



A GRID COMPUTING FRAMEWORK FOR COMMERCIAL SIMULATION PACKAGES

A thesis submitted for the degree of Doctor of Philosophy

by

Navonil Mustafee

**School of Information Systems, Computing and Mathematics
Brunel University**

May 2007

ABSTRACT

An increased need for collaborative research among different organizations, together with continuing advances in communication technology and computer hardware, has facilitated the development of distributed systems that can provide users non-trivial access to geographically dispersed computing resources (processors, storage, applications, data, instruments, etc.) that are administered in multiple computer domains. The term *grid computing* or *grids* is popularly used to refer to such distributed systems. A broader definition of grid computing includes the use of computing resources within an organization for running organization-specific applications. This research is in the context of using grid computing within an enterprise to maximize the use of available hardware and software resources for processing enterprise applications.

Large scale scientific simulations have traditionally been the primary benefactor of grid computing. The application of this technology to simulation in industry has, however, been negligible. This research investigates how grid technology can be effectively exploited by simulation practitioners using Windows-based commercially available simulation packages to model simulations in industry. These packages are commonly referred to as *Commercial Off-The-Shelf (COTS) Simulation Packages (CSPs)*.

The study identifies several higher level grid services that could be potentially used to support the practise of simulation in industry. It proposes a grid computing framework to investigate these services in the context of CSP-based simulations. This framework is called the *CSP-Grid Computing (CSP-GC) Framework*. Each identified higher level grid service in this framework is referred to as a CSP-specific service. A total of six case studies are presented to experimentally evaluate how grid computing technologies can be used together with unmodified simulation packages to support some of the CSP-specific services.

The contribution of this thesis is the CSP-GC framework that identifies how simulation practise in industry may benefit from the use of grid technology. A further contribution is the recognition of specific grid computing software (grid middleware) that can possibly be used together with existing CSPs to provide grid support. With its focus on end-users and end-user tools, it is intended that this research will encourage wider adoption of grid computing in the workplace and that simulation users will derive benefit from using this technology.

CONTENTS

ABSTRACT, CONTENTS AND ACKNOWLEDGEMENTS.....ii

ABSTRACT.....	ii
CONTENTS.....	iii
LIST OF FIGURES.....	vi
LIST OF TABLES.....	viii
LIST OF SCREENSHOTS	x
LIST OF GRAPHS.....	xi
ACKNOWLEDGEMENTS.....	xii
DECLARATION.....	xiii
GLOSSARY.....	xv

1 INTRODUCTION..... 1

1.1 Rationale and motivation	2
1.2 Aim and objectives	3
1.3 Research methods	3
1.4 Audience, scope and limitation of this research.....	4
1.5 Thesis structure	5
1.6 Chapter summary	8

2 GRID COMPUTING AND SIMULATION PACKAGES9

2.1 Introduction.....	9
2.2 Grid computing	10
2.3 Computer simulation.....	35
2.4 COTS Simulation Packages (CSPs)	36
2.5 Higher-level grid services for CSP-based simulation	40
2.6 Distributed simulation	54
2.7 Web-based simulation	63
2.8 Grid middleware and CSPs	71
2.9 Public-Resource Computing (PRC) middleware BOINC.....	73
2.10 Enterprise Desktop Grid Computing (EDGC) middleware Condor	77
2.11 Different approaches to using CSPs with desktop grids	87
2.12 Chapter summary	89

3 PROPOSING THE CSP-GC FRAMEWORK91

3.1 Introduction.....	91
3.2 The CSP-GC Framework	91
3.3 Grid-facilitated CSP-specific services	93
3.4 Investigation of CSP-specific services using BOINC and Condor	101
3.5 Suitability of BOINC and Condor for CSP-specific services.....	105

3.6	Suitability of BOINC and Condor for deployment in industry	106
3.7	Chapter summary	111
4	DEVELOPMENT OF DESKTOP GRIDS FOR WINDOWS	113
4.1	Introduction	113
4.2	WinGrid architecture	113
4.3	WinGrid-WS architecture	116
4.4	CSP-grid integration technology	117
4.5	Investigation of CSP-specific services using WinGrid and WinGrid-WS	124
4.6	Suitability of WinGrid and WinGrid-WS for CSP-specific services	126
4.7	Chapter summary	127
5	CASE STUDIES	129
5.1	Introduction	129
5.2	Criteria for hypothesis evaluation	129
5.3	CSP-GC framework investigation scenarios	131
5.4	BOINC case study for evaluation of SMMD task farming service	135
5.5	Condor case study for evaluation of MMD task farming service	143
5.6	Ford case study for evaluation of SMMD task farming service	151
5.7	IB case study for evaluation of workflow and SMMD task farming services	159
5.8	NBS case study for evaluation of distributed simulation service	172
5.9	Chapter summary	184
6	REVISITING THE CSP-GC FRAMEWORK	186
6.1	Introduction	186
6.2	Distributed simulation with SMMD and MMD task farming service	186
6.3	MU case study for evaluation of distributed simulation with task farming service ...	191
6.4	CSP-GC framework revisited	197
6.5	Evaluation of CSPs based on CSP-GC framework defined services	204
6.6	Chapter summary	207
7	SUMMARY AND CONCLUSION	208
7.1	Introduction	208
7.2	Research summary	208
7.3	Aims and objectives revisited	210
7.4	Contribution of this research	212
7.5	Further research	213

REFERENCES AND APPENDICES	214
REFERENCES.....	214
APPENDIX A: Vendor URLs.....	242
APPENDIX B: NBS case study - further discussion.....	250
APPENDIX C: BOINC case study - experiments and results.....	262
APPENDIX D: WinGrid user documentation (version 1.0).....	264

LIST OF FIGURES

Figure 1: Chapters and their purpose.....	7
Figure 2: Users' view of grid computing	14
Figure 3: Interaction between resource broker and grid resources.....	16
Figure 4: Example of remote steering in RealityGrid (Brooke et al., 2003).....	18
Figure 5: GT-4 container hosting user-defined web services (Foster, 2006).....	20
Figure 6: GT-4 architecture showing the different components (Foster, 2006).....	23
Figure 7: Conceptual view of users and service providers (OMII, 2006b)	26
Figure 8: Different forms of grid computing	31
Figure 9: Parallel computing using multiple CPUs (Barney, 2006)	42
Figure 10: Shared-memory (A) and distributed-memory (B) multiprocessor machines.....	43
Figure 11: Parallel computing using a DES CSP	44
Figure 12: Workflow using a DES CSP and a visualization application	48
Figure 13: Frequency of model re-use and its underlying complexity (Pidd, 2002)	51
Figure 14: Execution of events in a distributed simulation (Fujimoto, 1990).....	56
Figure 15: CSP-based distributed simulation using FAMAS (adapted from Boer, 2005)	61
Figure 16: Distributed simulation using Simul8 and CSPE-CMB middleware.....	62
Figure 17: Layered architecture of Internet Protocol (IP) stack.....	65
Figure 18: Communication between client and server programs.....	66
Figure 19: Remote S&A approach to web-based simulation using CSPs.....	67
Figure 20: Local S&A approach to web-based simulation using CSPs.....	68
Figure 21: Java data server approach to web-based simulation using CSPs.....	68
Figure 22: The "pull" model of PRC projects	74
Figure 23: The BOINC system	74
Figure 24: Multiple BOINC projects in an organization	76
Figure 25: Condor resource management architecture.....	79
Figure 26: Communication between different Condor processes	79
Figure 27: Graphical representation of diamond DAG (Frey, 2002)	85
Figure 28: Processing job using Condor MW.....	86
Figure 29: The CSP-GC framework	92
Figure 30: Middleware integration approach to providing distributed simulation service.....	99
Figure 31: Application integration approach to providing distributed simulation service.....	100
Figure 32: The "push" model implemented by WinGrid.....	114
Figure 33: WinGrid architecture.....	115
Figure 34: Architecture of WinGrid-WS	116
Figure 35: Interaction between WTC-WAadapter-CSP.....	118
Figure 36: Interaction between WJD-MAadapter-Excel	119
Figure 37: UML sequence diagram showing the interaction between WinGrid components	123
Figure 38: Execution of RAS application using BOINC.....	140

Figure 39: BOINC test network.....	142
Figure 40: Execution of RAS and AO applications on a Condor pool	145
Figure 41: Condor test pool	151
Figure 42: Integration architecture of WinGrid and First	156
Figure 43: Integration architecture of WinGrid and IRS-RBF application.....	164
Figure 44: IRS-RBF application workflow.....	166
Figure 45: Simplified model of the NBS supply chain with NBS PTI (left) and one hospital .	175
Figure 46: Conventional simulation approach with NBS PTI and four hospitals	175
Figure 47: NBS distributed simulation with NBS PTI and four hospitals	177
Figure 48: CSP Controller Middleware (CCM) architecture	180
Figure 49: Distributed simulation with SMMD task farming.....	187
Figure 50: Distributed simulation with MMMD task farming	187
Figure 51: Integration architecture of WinGrid and DPL	194
Figure 52: CSP-GC framework (modified).....	200
Figure 53: Chapters that meet the different objectives outlined in this thesis	211
Figure 54: CSP Controller Middleware (CCM) architecture	251
Figure 55: CCM-Next Event Request (NER) protocol	251
Figure 56: CCM-Time Advance Request (TAR) protocol.....	252
Figure 57: Time Management States of a Federate (adapted from Kuhl et al., 1999).....	258

LIST OF TABLES

Table 1: e-Science projects that use grid computing	11
Table 2: Examples of production grids	27
Table 3: Example of organizations that use grid computing middleware	29
Table 4: Comparing different forms of grid computing	32
Table 5: Survey of CSPs (extended from Swain's OR/MS survey of simulation tools).....	36
Table 6: CSPs that support parallel computation	45
Table 7: CSPs that provide support for task farming.....	46
Table 8: Potential applications of simulation in a networked environment (Robinson, 2005b)	46
Table 9: CSPs that support data source access	48
Table 10: CSPs that expose package functionality	49
Table 11: CSPs that support creation of reusable model components	52
Table 12: CSPs that facilitate model sharing	53
Table 13: Application areas of parallel and distributed simulation	55
Table 14: CSPs and distributed simulation support	63
Table 15: CSPs that provide support for web-based simulation	68
Table 16: Interaction between different Condor processes.....	80
Table 17: CSP-GC framework defined services and their descriptions	92
Table 18: Michael Flynn's classification of computer architectures	95
Table 19: Possible task farming scenarios with CSPs and desktop grids.....	96
Table 20: BOINC and Condor support for CSP-specific services	105
Table 21: BOINC, Condor and middleware deployment considerations.....	107
Table 22: Ideal middleware implementation for CSP-based simulation.....	111
Table 23: Interfaces used for communication between WTC and WA adapter	118
Table 24: Interfaces used for communication between WJD and MA adapter	119
Table 25: Interfaces used for communication between MA and MA adapter.....	120
Table 26: Interfaces used for communication between WA and WA adapter.....	121
Table 27: WinGrid and WinGrid-WS support for CSP-specific services	126
Table 28: Middleware support for CSP-specific services	126
Table 29: Criteria for hypothesis evaluation	131
Table 30: CSP-specific services that can be potentially implemented.....	132
Table 31: Case studies	133
Table 32: BOINC case study	135
Table 33: Condor case study.....	143
Table 34: Ford case study	152
Table 35: Investment bank case study	159
Table 36: Execution time for different products using the original IRS-RBF application.....	162
Table 37: Workunits to be processed by IRS and RBF simulations.....	168
Table 38: Configuration of WinGrid nodes	168

Table 39: NBS case study	173
Table 40: Grid middleware support for distributed simulation with task farming service	191
Table 41: Manufacturing unit case study.....	191
Table 42: Modified CSP-GC framework defined services and their descriptions	201
Table 43: Custom CSP support and grid middleware support for CSP-specific services.....	205
Table 44: Vendor URLs – support for parallel computing	242
Table 45: Vendor URLs – task farming support in CSPs	243
Table 46: Vendor URLs – data source access support in CSPs.....	244
Table 47: Vendor URLs – CSPs that expose package functionality	245
Table 48: Vendor URLs – reusable model components support in CSPs	246
Table 49: Vendor URLs – support for sharing models in CSPs	247
Table 50: Vendor URLs - distributed simulation support in CSPs.....	248
Table 51: Vendor URLs – support for web-based simulation.....	249
Table 52: Percentage performance increase of TAR over NER	254

LIST OF SCREENSHOTS

Screenshot 1: Group-based collaboration using Access Grid.....	20
Screenshot 2: Job submission web page for the NGS portal.....	70
Screenshot 3: Workflow editor in P-GRADE portal (adapted from Kiss, 2007).....	70
Screenshot 4: JVM related information output using Condor command “condor_status”.....	82
Screenshot 5: Web front-end to WinGrid-WS (Alstad, 2006).....	125
Screenshot 6: Range Accrual Swap (RAS) application.....	137
Screenshot 7: Setting user preference using menu provided by BOINC core client	138
Screenshot 8: Setting user preference using web interface.....	138
Screenshot 9: BOINC core client attached to multiple projects	139
Screenshot 10: Asian Options (AO) application	145
Screenshot 11: Job submit file for AO application.....	146
Screenshot 12: Job submit file for RAS application	147
Screenshot 13: Results from the simulation experiments	148
Screenshot 14: Condor jobs getting executed in temporary execution directory	148
Screenshot 15: AO and RAS applications execution over Condor pool	149
Screenshot 16: Status of job queue displayed using Condor command “condor_q”	150
Screenshot 17: Jobs removed from the queue using Condor command “condor_rm”	150
Screenshot 18: FIRST application main menu.....	153
Screenshot 19: Graph generated by FIRST using data returned by Witness	153
Screenshot 20: FIRST experimentation tool showing a list of experiments.....	155
Screenshot 21: MCS CSP Analytics Desktop application	160
Screenshot 22: WJD Application Specific Parameter (APS) tool for IRS-RBF application... ..	163
Screenshot 23: WinGrid WJD console showing execution of workflow in phases 1 to 3.....	167
Screenshot 24: Condor queue after submission of multiple instances of job-A and job-B ...	189
Screenshot 25: DES CSP Simul8 model “sourceA” (DPL application)	192
Screenshot 26: DES CSP Simul8 model “sourceB” (DPL application)	192
Screenshot 27: DES CSP Simul8 model “destC” (DPL application)	193
Screenshot 28: Excel-based Distributed Production Line-Experimentation Tool (DPL-ET) .	194
Screenshot 29: WinGrid console showing execution of distributed simulation federations .	195
Screenshot 30: HLA-RTI executive process executing federations EXP1 and EXP2	196

LIST OF GRAPHS

Graph 1: Time taken to execute FIRST application using different workloads	157
Graph 2: Time taken to execute the IRS-RBF application using different workloads	169
Graph 3: Total job assignments for IRS-RBF simulation.....	171
Graph 4: Job assignments for different phases of IRS-RBF simulation (4 nodes)	171
Graph 5: Job assignments for different phases of IRS-RBF simulation (8 nodes)	172
Graph 6: Execution time of NBS distributed simulation and NBS standalone simulation	181
Graph 7: Monthly execution time of NBS distributed and standalone simulations.....	182
Graph 8: RAS application results	262

ACKNOWLEDGEMENTS

It would not have been possible for me to complete this research without the help, support, advice and encouragement that I have received from my family, friends, teachers and my colleagues. Looking back, I would like to thank my mom (Sabita Mustafee) and dad (Dr. Tulsi Pada Mustafee) for their sacrifice which allowed me to get a good education in life; my brother (Indronil Mustafee) who persuaded me to do to a career shift from hospitality to computing and for teaching me the art of programming in C and C++ languages; my teachers from yesteryear who have instructed me; my friends in India for being there for me.

I would like to thank my supervisor Dr. Simon Taylor for his guidance and constructive criticisms throughout the course of my MSc. and PhD. studies. I gratefully acknowledge the motivation and the inspiration that I have received from him, time and time again. Thanks are due to my friends and colleagues Carole Bromley, Carolyn Bailey, Jon Saville, Meeta Talwalkar, Neela Rungien and Saptarshi Ghosh. They have unreservedly extended their personal support at very difficult times. Without their encouragement and help this research would have taken longer. Special thanks to Carolyn Bailey for her help with proofreading this thesis.

In the course of this study I have collaborated with fellow researchers and people in industry, and I would like to express my gratitude to Anders Alstad, Bjørn Larsen, Dr. David Bell, Prof. Eduardo Saliby, Jingri Zhang, John Ladbrook, Jonathan Berryman, Dr. Korina Katsaliaki, Dr. Mark Elder, Rahul Talwalkar, Robert Watson, Dr. Sally Brailsford and Dr. Steve Turner. They have contributed to this research in various ways.

I am obliged to Prof. Ray Paul for securing the funding that has allowed me to do this research in the first place. His unrelenting advice and teachings, on matters not limited to research alone, has facilitated my personal development in many ways. Thank you Professor, I hope that in future I will be worthy of the education you have imparted.

DECLARATION

Research from this thesis that has been published or is presently under review is given below.

Refereed journal publications

- Taylor, S. J. E., Turner, S. J., **Mustafee, N.**, Ahlander, H. and Ayani, R. (2005). COTS distributed simulation: a comparison of CMB and HLA interoperability approaches to type I interoperability reference model problems. *SIMULATION: Transactions of the Society of Modelling and Simulation International*. Volume 81(1): 33-43.

Journal papers under review

- Katsaliaki, K., **Mustafee, N.**, Taylor, S. J. E. and Brailsford, S. Comparing conventional and distributed approaches to simulation in complex supply-chain health systems. *Journal of the Operational Research Society*.

Refereed conference papers

- Zhang, J., **Mustafee, N.**, Saville, J. and Taylor, S. J. E. Integrating BOINC with Microsoft Excel: a case study. . In Proceedings of the 29th Information Technology Interfaces Conference (accepted).
- **Mustafee, N.**, Taylor, S. J. E., Katsaliaki, K. and Brailsford, S. (2007). Using CSPI distributed simulation standards for the analysis of a health supply chain. In *Proceedings of Simulation and Visualization 2007*, Schulze, T., Preim, B. and Schumann, H. (eds.), pp. 155-168. The Society for Modelling and Simulation International, SCS European Publishing House, Germany.
- **Mustafee, N.**, Taylor, S. J. E., Katsaliaki, K. and Brailsford, S. (2006). Distributed simulation with COTS simulation packages: a case study in health care supply chain simulation. In *Proceedings of the 37th Winter Simulation Conference*, Perrone, L. F., Wieland, F. P., Liu, J., Lawson, B. G., Nicol, D. M. and Fujimoto, R. M. (eds.), pp. 1136-1142. Winter Simulation Conference, USA.
- **Mustafee, N.**, Alstad, A., Larsen, B., Taylor, S. J. E. and Ladbrook, J. (2006). Grid-enabling FIRST: Speeding up simulation applications using WinGrid. In *Proceedings of the 10th International Symposium on Distributed Simulation and Real-Time Applications (DSRT 2006)*, Alba, E., Turner, S. J., Roberts, D. and Taylor, S. J. E. (eds.), pp. 157-164. IEEE Computer Society, Washington, DC, USA.
- Bell, D., **Mustafee, N.**, Taylor, S. J. E., de Cesare, S. and Lycett, M. (2006). A web services component discovery and deployment architecture for simulation model reuse. In *Proceedings of the 2006 European Simulation Interoperability Workshop (EURO SIW)*. 06E-SIW-047. Simulation Interoperability Standards Organization, Orlando, Florida, USA.
- **Mustafee, N.** and Taylor, S. J. E. (2006). Using a desktop grid to support simulation modelling. In *Proceedings of the 28th Information Technology Interfaces Conference (ITI*

2006), Stiffler, V.L. and Dobric, V. H. (eds.), pp. 557-562. IEEE Computer Society, Washington, DC, USA.

- **Mustafee, N.** and Taylor, S.J.E. (2006). Investigating distributed simulation with COTS simulation packages: experiences with Simul8 and the HLA. In *Proceedings of the 2006 Operational Research Society Simulation Workshop (SW06)*, Garnett, J., Brailsford, S., Robinson, S. and Taylor, S. (eds.), pp. 33-42. Operational Research Society, Birmingham, UK.
- Brailsford, S., Katsaliaki, K., **Mustafee, N.** and Taylor, S. J. E. (2006). Modelling very large complex systems using distributed simulation: a pilot study in a healthcare setting. In *Proceedings of the 2006 Operational Research Society Simulation Workshop (SW06)*, Garnett, J., Brailsford, S., Robinson, S. and Taylor, S. (eds.), pp. 257-262. Operational Research Society, Birmingham, UK.

GLOSSARY

BOINC: Berkeley Open Infrastructure for Network Computing (BOINC) is a desktop grid middleware that was primarily created for Public-Resource Computing (PRC).

BOINC application client: The user application that is executed by BOINC core client.

BOINC core client: The BOINC client side middleware that is installed on different grid nodes. The BOINC core client executes different user-developed BOINC application clients.

BOINC-PAC: BOINC-Proxy Application Client (BOINC-PAC). BOINC application client that has client side dependencies. For example, the BOINC application client may invoke operations on Excel, Simul8, etc. that are installed on a local resource.

BOINC-RAC: BOINC-Runtime Application Client (BOINC-RAC). BOINC application client that has no client-side dependencies. Only BOINC core client needs to be pre-installed on each client computer.

Condor: Condor is a grid middleware that is supported on Windows platform. Condor is an Enterprise Desktop Grid Computing (EDGC) middleware.

Condor DAGMan: Condor Directed Acyclic Graph Manager (DAGMan). A component of Condor which supports execution of workflows.

Condor Java Execution Environment: Condor middleware can execute Java programs through the Condor Java Execution Environment. Only PCs that have the Java Runtime Environment (JRE) installed can be a part of this environment.

Condor MW: Condor Master Worker (MW). MW is a C++ library that can be used to create task farming applications for execution over the Condor pool.

Condor Pool: A collection of computers that are installed with the Condor middleware and that process Condor jobs.

COTS: Commercial, Off-The-Shelf (COTS). This term is used to refer to software applications that can be purchased from software vendors.

CSP: COTS Simulation Package (CSP). In this thesis the term CSP is used to refer to simulation packages for both Discrete-Event Simulation (DES) and Monte Carlo Simulation (MCS).

DES: Discrete-Event Simulation (DES).

EDGC: Enterprise Desktop Grid Computing (EDGC). It refers to a grid infrastructure that is confined to an institutional boundary, where the spare processing capacities of an enterprise's desktop PCs are used to support the execution of the enterprise's applications.

HLA: The High Level Architecture (HLA) is an IEEE standard for distributed simulation.

HLA-RTI: The High Level Architecture-Run Time Infrastructure (HLA-RTI) is distributed simulation middleware that implements the interface specifications outlined by the HLA standard.

Job-parallel application: An application that uses standard grid mechanisms to submit a batch of jobs for processing. If a user submits multiple instances of the same job for processing, then it is also referred to as job-parallel execution.

MA: Master Application (MA). In WinGrid terminology, a MA is an Excel-based application that lists experiment parameters for batch simulations. It can also be used to display the results of the different simulations.

Manager federate: In HLA-based distributed simulation, the HLA federate which co-ordinates the other federates during the execution of a distributed simulation.

Master computer, Master process, Master: The master process in the master-worker distributed computing architecture. The grid node over which the master process runs is sometimes referred to as the master computer.

MCS: Monte Carlo Simulation (MCS).

Middleware: A software program that interfaces between two or more programs. The term is also used to refer to software that enables communication between distributed computing resources.

MMMD: Multiple Model Multiple Data (MMMD) is a form of task farming. It refers to the concurrent execution of different CSP models using different experiment parameters over multiple processors.

MPI: Message Passing Interface (MPI). Used in the context of parallel programming.

Node, Grid node: A computing resource that is a part of a grid infrastructure, e.g., desktop PCs.

P2P: Peer-to-Peer (P2P). It refers to a non-centralized infrastructure for file sharing over the Internet (such as, KaZaA).

PRC: Public-Resource Computing (PRC). This refers to the use of millions of volunteer computers for scientific processing (such as, SETI@Home project).

PVM: Parallel Virtual Machine (PVM). Used in the context of parallel programming.

Rtiexec: rtiexec.exe is the HLA-RTI program.

SMMD: Single Model Multiple Data (SMMD) is a form of task farming. It refers to the concurrent execution of one CSP model using different experiment parameters over multiple processors.

Socket communication: A form of communication between two processes executing on different computers.

Task-parallel application: An application in which one process acts as the master and is responsible for directing and coordinating the computations being executed on the workers.

WA: Worker Application (WA). In WinGrid, the unmodified CSPs are referred to as WA.

WinGrid: WinGrid, or the desktop grid for Windows, is a desktop grid middleware that was implemented by the author during the course of this study.

WJD: WinGrid Job Dispatcher (WJD). This is the WinGrid job scheduler that runs on only one computer. It is responsible for allocating jobs to different WTCs.

WMS: Workflow Management System (WMS) (such as, Condor DAGMan).

Worker computer, Worker process, Worker: The worker process in the master-worker distributed computing architecture. The grid node over which the worker process runs is sometimes referred to as the worker computer.

WTC: WinGrid Thin Client (WTC). This refers to the WinGrid software component that is installed on different WinGrid nodes. WTC runs a server socket to listen for job requests that may be coming from the WJD.

1 INTRODUCTION

Grid computing has the potential to provide users on-demand access to computational resources, just as power grids provide users with consistent, pervasive, dependable and transparent access to electricity, irrespective of its source (Baker et al., 2002). Simulation modelling is an Operational Research (OR) technique that can benefit from this, as computing power can be a bottleneck to the development of simulation (Robinson, 2005a). Discrete-Event simulation is arguably the most frequently used classical OR technique that is applied across a range of industries like manufacturing, travel, finance and healthcare, among others (Hollocks, 2006). Commercially available discrete-event simulation packages are generally used to model such simulations (Taylor et al., 2005b). Monte Carlo simulation is yet another OR technique that is extensively used in application areas like finance and insurance (Herzog and Lord, 2002). Commercially available spreadsheet applications, spreadsheet add-ins and Monte Carlo simulation packages are often used for modelling Monte Carlo simulations in industry (Swain, 2007). The term *Commercial Off-The-Shelf (COTS) Simulation Packages (CSPs)* is used in this thesis to refer to software used for modelling both Discrete-Event and Monte Carlo simulations. Discrete-Event simulation and Monte Carlo simulation are henceforth referred to as DES and MCS respectively. The focus of this research is on investigating how simulation users in industry using such CSPs can benefit from grid computing.

The hypothesis presented in this thesis is that ***grid computing will benefit CSP-based simulation practice in industry***. The hypothesis is considered important because it looks at grid computing from the end-users' perspective (and thus the focus on end-user simulation software), wherein the end-users are not expected to be IT specialists. As will be seen from the literature review, the end user adoption of grid computing technologies in the work place has been extremely limited. This adds further significance to this hypothesis. As the scope of this research is limited to the practice of simulation in industry, the end-users are simulation practitioners and the tools used are CSPs. This research is arguably the first attempt to undertake a study of CSPs in the context of grid computing.

This research proposes a grid computing framework to evaluate the hypothesis presented in this thesis. This framework is called the *COTS Simulation Package-Grid Computing (CSP-GC) Framework* and it provides a logical structure for evaluation of the hypothesis. CSP-GC framework is built through a review of the field of grid computing. This review identifies some of the higher level grid services that could possibly be used to support CSP-based simulation in industry. This framework is then evaluated by developing case studies and through case study experimentation. Finally, the hypothesis is either supported or rejected.

This research is considered as end-user oriented because, apart from proposing the CSP-GC framework that identifies the possible uses of grid computing for CSP-based simulation in industry, it also informs those engaged in such simulations of existing grid computing middleware that they could possibly use. It is hoped that this would encourage the adoption of grid computing among simulation practitioners in industry.

1.1 Rationale and motivation

The rationale of this thesis is based on the recognition that the development in simulation has been closely allied to the advances in the field of computing (Robinson, 2005a). It can therefore be expected that simulation software will continue to rely on the latest advances in computing to support increasingly large and complex simulations (Pidd and Carvalho, 2006). Grid computing is arguably the latest advancement in the field of distributed computing. The rationale of this thesis is that, as previous developments in computing have been adopted by the simulation users and they have benefited from it, similarly grid computing technologies provide an opportunity to further the practise of simulation in industry.

This research is motivated by the advances being made in the field of grid computing and the advantages being derived by various disciplines through the adoption of grid computing technologies. Simulation modelling is a problem solving methodology that has arguably gained the most from using grid computing to conduct scientific simulations in disciplines like particle physics, climatology, astrophysics and medicine, among others. Simulation is also widely used in industry to aid decision making. It is, therefore, considered to be a logical next step to investigate how simulation practice in industry can benefit from grid computing.

A further motivation of this research is the low adoption rate of grid computing outside of academic and research domains. At present a major proportion of grid users comprises researchers (physicists, biologists, climatologists, etc. – they can be considered as the primary stakeholder of the applications running on the grid) and computer specialists with programming skills (they usually provide IT support to the primary stakeholders). This is not unexpected as the majority of applications using grid computing are research applications. The adoption of grid computing technologies by employees at their work place has been minimal. One important reason for this is, although the employees are experts in their own discipline they generally do not have the necessary technical skills that are required to work with present generation grids. A possible means to increase adoption is to incorporate grid support in software applications that are used by the end-users to perform their day-to-day jobs. Simulation practitioners in industry usually create simulations using CSPs. It was therefore considered appropriate to focus on these simulation tools and to propose a grid computing framework which investigates how the CSPs can benefit from grid computing technologies.

1.2 Aim and objectives

The aim of this thesis is to *investigate how grid computing will benefit CSP-based simulation practice in industry*. Towards this aim the following four objectives will be met.

- **Objective 1:** *State the hypothesis and identify what grid computing has to offer*

The hypothesis has already been presented in this chapter. The hypothesis states that CSP-based simulation practice in industry will gain from using grid computing technologies. Through literature survey, the latest developments in grid computing are examined; the potential of using grid technologies are recognised; and several higher level grid services that could be used to support the CSPs are identified.

- **Objective 2:** *Propose the CSP-GC framework and identify grid computing middleware that can potentially support the framework*

To provide a logical structure for evaluation of the hypothesis, the CSP-GC framework is proposed. The framework identifies several grid-facilitated CSP-specific services that can be potentially provided through the use of grid computing. These CSP-specific services are in turn based on higher level grid services (objective 1). Through literature review, specific grid computing middleware are identified that could possibly support the CSP-specific services outlined by the CSP-GC framework.

- **Objective 3:** *Experimentally test the CSP-GC framework*

Case studies are developed to experimentally test a subset of these middleware (identified in objective 2) in relation to their support for some of the CSP-specific services identified by the CSP-GC framework.

- **Objective 4:** *Evaluate CSP-GC framework and test the hypothesis*

The CSP-GC framework is evaluated based on the discussions on grid middleware in relation to CSP-specific services (objective 2) and the results of the case study experimentation (objective 3). Based on this evaluation, the hypothesis is either accepted or rejected.

1.3 Research methods

Empirical research has been conducted in this study to experimentally investigate how grid computing middleware can be used with existing CSPs for the benefit of the simulation end-users. Empirical research method in computer science generally follows four distinct steps – hypothesis generation, method identification, result compilation and conclusion (Johnson, 2003). In the hypothesis generation stage the idea to be investigated is explicitly stated. The techniques that would be used to examine the hypothesis are then identified in the method identification stage. Experimentation is generally one of the methods used during this stage,

as empirical research in computer science stresses the repeatability of results. The results of the experiments are presented in the result compilation stage, based on which conclusions are drawn and the hypothesis is either supported or rejected. A short discussion of these different stages in the context of this research is presented below. The specific chapters in which these stages have been used are also indicated.

The hypothesis presented in this research is that grid computing will benefit CSP-based simulation practice in industry (chapter 1). The methods that have been used in order to progressively establish this hypothesis are as follows.

- *Literature review* of grid computing and CSP-based simulation in industry. This is done in order to investigate how grid computing technologies can be used to support CSP-based simulation in industry (chapter 2).
- A *framework* that would provide a logical structure for evaluation of the hypothesis (chapter 3).
- *Case studies* to experimentally evaluate the framework (chapter 5). A total of six real-world and hypothetical case studies have been presented in this research.

The results of the experiments are then presented (chapter 5). Conclusions are finally drawn on the basis of these results and grid-specific discussions in the earlier chapters, and the hypothesis is either accepted or rejected (chapter 6).

This research has also led to the development of a grid computing middleware that is specifically targeted at the CSPs (chapter 4). Some aspects of design research have been used during the development of this artefact. In short, design research uses existing knowledge in a problem area to suggest solutions that are implementable in the form of software artefacts (Vaishnavi and Kuechler, 2006). These artefacts are then evaluated based on a set of criteria. Artefact development and evaluation are both iterative processes, and each iteration adds to knowledge in the problem domain. The problem area in this research is the application of grid computing to CSP-based simulations. The artefact that is developed is a grid computing middleware (WinGrid) that can support CSPs.

1.4 Audience, scope and limitation of this research

This research has been written with the following audience in mind.

- *Simulation practitioners* who use CSPs to model simulations in industry. It is expected that this research would inform them of existing grid computing technologies that they could benefit from.
- *Researchers in grid computing* may find the end-user driven “grid at the workplace” approach to grid computing that is presented in this research as a facilitator for wider adoption of grid computing in the enterprise. This can encourage development of grid

computing software that is specifically targeted at software tools used by end-users in their workplace.

- The *CSP vendors* may consider grid-enabling their existing simulation products for the benefit of their customers. It has to be added, however, that the focus of this thesis is on using general purpose grid computing solutions with CSP packages. Implementing customized grid computing middleware that supports software packages developed by one particular vendor may be an intermediate solution. However, in the long run it is hoped that software vendors (not limited to CSP vendors alone), researchers in grid computing, developers of both open source and commercial grid computing middleware, standard creation bodies, end users, among others, will work together to create standards that would facilitate software applications to utilize multiple computing resources, made available through grid middleware, for processing end-user computation jobs.
- *Researchers in distributed simulation* may find the sections pertaining to CSP-based distributed simulation using IEEE 1516 HLA standard interesting. They may be encouraged to adopt an approach similar to the one presented in this research for their own research projects. The *CSP vendors* that are perhaps interested in incorporating package-level support for distributed simulation in future may also benefit from this research.

The scope of this research is limited to investigating four specific grid computing middleware (BOINC, Condor, WinGrid and WinGrid-WS) in the context of providing certain grid-facilitated higher level services to Windows-based DES and MCS packages. Furthermore, these packages should be accessible by external applications through well-defined interfaces that are exposed by the DES and the MCS CSPs.

The limitation of this research is that it only evaluates grid technologies that are freely available or those that have been implemented during the course of this research. Furthermore, although this research is targeted at end-users who are considered experts in simulation modelling but not necessarily in information technology, practical implementation of the CSP-grid integration solutions presented in this thesis will only be possible if the end-users have programming knowledge (Java and Visual Basic) and are familiar with grid middleware. However, it is hoped in the future the CSP-grid integration solutions will become transparent to the user.

1.5 Thesis structure

This thesis is structured into 7 chapters. This chapter (**chapter 1**) has presented the research hypothesis, has identified the aim and objectives of this research, the research method to be used, the intended audience and finally its scope and limitations.

Chapter two of this thesis reviews the literature in the field of grid computing and presents an overview of CSP-based simulation in industry. The objective of this chapter is to examine how

grid computing technologies can be used to support CSP-based simulation in industry. To this end, this chapter identifies six higher level grid services that could potentially be used together with CSPs. Furthermore, it identifies different forms of grid computing and specific grid computing middleware that can potentially be used in the enterprise environment.

Chapter three builds on the higher level grid services identified in the previous chapter and proposes the CSP-GC framework. This framework provides a logical structure for evaluation of the hypothesis. The framework refers to each higher level grid service as a grid-facilitated CSP-specific service that could be potentially supported using grid computing. The grid computing middleware identified in chapter two are then evaluated with regards to each CSP-specific service. The chapter concludes by arguing the need for a grid middleware that is specifically implemented, based on identified “ideal” middleware implementation requirements, to support CSP-based simulation in industry.

Chapter four discusses the architecture of a grid middleware (WinGrid) that is implemented during the course of this research. WinGrid incorporates the “ideal” middleware implementation requirements (which were identified in chapter three) and is specifically aimed at CSPs. Finally, WinGrid is examined in relation to the CSP-GC framework defined services to investigate whether it can support some of these services.

Chapter five investigates whether the grid-facilitated, CSP-specific solutions identified in chapters 3 and 4 are implementable in practice. This is done by designing case studies that experiment with grid middleware and CSPs. This is considered important because this thesis is end-user oriented and it attempts to present the simulation user with solutions that can be implemented at their workplace. The criteria for evaluating the CSP-specific services are also presented in this chapter. As case studies are grouped under one or more of these CSP-GC framework defined services, the evaluation criteria outlined for each service can be considered as the evaluation criteria for the respective case studies under it. A total of five real-world and hypothetical case studies are presented in this chapter.

Chapter six evaluates the CSP-GC framework based on the results of the case study experimentation (chapter 5) and the discussions pertaining to middleware support for CSP-GC framework defined services (chapters 3 and 4). The hypothesis presented in this research is accepted or rejected based on the evaluation of the CSP-GC framework.

Chapter seven is the final chapter of this thesis. It provides a summary of the research and discusses its contribution. It highlights how the aim and the objectives of this research have been met. The chapter concludes by suggesting future areas of research in this field.

Figure 1 shows the purpose of each chapter and how they are related to each other.

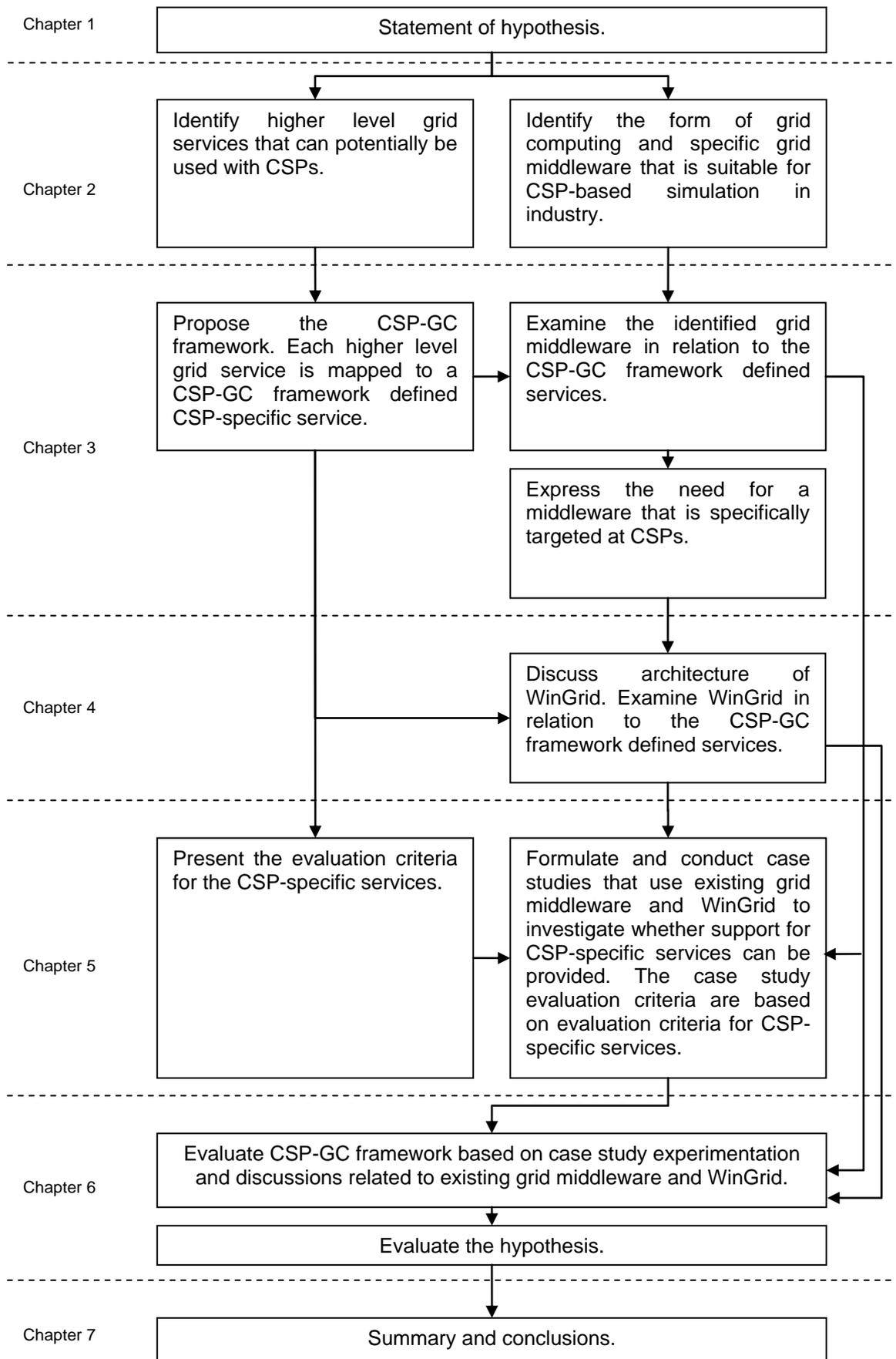


Figure 1: Chapters and their purpose

1.6 Chapter summary

This chapter has presented the motivation and the rationale for conducting this research (section 1.1), the aim and objectives (section 1.2) and the research methods that will be used (section 1.3). It has further identified the audience, scope and limitations of this work (section 1.4). Finally, this chapter has given an overview of the structure of this thesis (section 1.5).

The next chapter is the literature review chapter. The purpose of this chapter is to identify how grid computing technologies can be used to support commercial simulation packages that are widely used in industry. To this end, this chapter identifies higher level grid services and specific grid middleware that can be used in the context of simulation in industry. This chapter also presents an overview of CSP-based simulation in industry.

2 GRID COMPUTING AND SIMULATION PACKAGES

2.1 Introduction

The research presented in this thesis hypothesises that simulation practice in industry can benefit from the use of grid computing. This chapter provides the context to this hypothesis through a discussion on grid computing and the CSPs that are generally used to model simulations in industry. This in turn identifies some of the potential benefits that could be accrued by using grid computing technologies together with the CSPs to further the practise of simulation in industry.

Section 2.2 of this chapter conducts a literature review on grid computing. It focuses on the definition of the term “grid”, discusses its predominant use in scientific projects, describes some of the uses of this technology (basic grid services and higher-level grid services), gives an overview of grid computing middleware and production-level grids being used around the world, and finally concludes with a discussion on different forms of grid computing.

This chapter then discusses simulation from an industry perspective and focuses on simulation tools that are commonly used to model such simulations (sections 2.3 and 2.4). Grid computing necessitates the use of multiple computing resources that are connected over the network. Thus, for grid computing to offer any practical benefit to simulation practitioners it is imperative that they have access to multiple networked PCs within their organization. Informed by the discussion on grid computing and CSPs in previous sections, section 2.5 identifies four higher-level grid services that can be potentially used to support the commercial simulation packages. This section further examines the extent to which the CSPs support functionality similar to those provided by the higher-level grid services through custom solutions.

This chapter discusses two specific forms of simulation that may gain from use of grid computing. These are, distributed simulation (section 2.6) and web-based simulation (section 2.7). The extent to which the CSPs support distributed simulation and web-based simulation through custom solutions are also discussed. This chapter then identifies two specific forms of grid computing that could be potentially used with unmodified CSPs (section 2.8), namely Public Resource Computing (PRC) and Enterprise Desktop Grid Computing (EDGC), and discusses PRC middleware BOINC (section 2.9) and EDGC middleware Condor (section 2.10) in detail. The chapter concludes with presenting three different approaches to using simulation tools together with grid computing software and identifies one of them to be most appropriate for this research (section 2.11).

2.2 Grid computing

The grid vision of providing users continuous access to computing resources, similar to public utility services like electricity and telephone, can be traced back to the *Multics* (Multiplexed Information and Computing Service) system that arguably discussed this in the context of time-sharing of a CPU among jobs of several users (Corbato and Vyssotsky, 1965). The term “grid computing” was itself preceded by the term *metacomputing* which also advocated transparent user access to distributed and heterogeneous computing resources by linking such resources by software and an underlying network (Smarr and Catlett, 1992).

Grid computing (or Grids) was first defined by Ian Foster and Carl Kesselman in their book “*The Grid: The Blueprint for a New Computing Infrastructure*” as a hardware and software infrastructure that provides access to high-end computational resources (Foster and Kesselman, 1998). It was further stated that this access should be dependable, consistent, pervasive and inexpensive. This definition of grid computing has since been modified twice by the grid veterans; once by Foster, Kesselman and Tuecke in their paper titled “*Anatomy of the Grid*” (Foster et al., 2001), and again by Foster and Kesselman with the publication of the second edition of their book “*The Grid: The Blueprint for a New Computing Infrastructure*” (Foster and Kesselman, 2004).

In Foster et al. (2001) grid computing has been distinguished from conventional distributed computing by its focus on large-scale resource sharing, innovative applications and high-performance orientation, with the objective of coordinated resource sharing and problem solving in dynamic multi-institutional virtual organizations. A virtual organization is defined as a group of individuals and/or institutions engaged in some joint task who share resources by following clearly stated sharing rules. These rules define what is shared, who is allowed to share and the condition under which sharing occurs. Unlike the previous definition, which seems to suggest that access to High Performance Computing (HPC) resources in supercomputing centres could be termed as grid computing, this definition lays special emphasis on collaborative resource sharing between organizations whose resources are generally under different administrative domains. It further clarifies the nature of sharing in the grid environment to include not only file exchange but rather direct access to computers, software, data, and other resources (attached computer peripherals, remote instruments like sensors, etc.).

In Foster and Kesselman (2004) grid has been defined as a system that coordinates distributed resources using standard, open, general-purpose protocols and interfaces with the aim of delivering non-trivial qualities of service. The three key elements that are highlighted in this definition are:

1. A grid provides *coordinated resource sharing* within an organization and among virtual organizations (VOs) and addresses issues of security, VO membership, sharing policy,

payment for use of resources, etc. that arise in such cross-organizational settings. The VO element in this definition is not new. It was introduced in Foster et al. (2001).

2. A grid is built using *standard, open, general-purpose protocols and interfaces* that address fundamental issues such as authentication, authorization, resource discovery and resource access. This is an important new element of the definition. It highlights that for any distributed system to be a part of the grid it must implement the “inter-grid” protocols and standards that are gradually being created by grid-standards creation communities like the *Open Grid Forum* (Open Grid Forum, 2007). This would encourage both open source and commercial distributed systems to interoperate effectively across organizations and thereby realize the grid vision.
3. A grid delivers *nontrivial qualities of service (QoS)* relating to throughput, availability, response time, resource co-allocation, etc., such that the utility derived from the grid infrastructure is significantly greater than what would have been derived if resources were used in isolation. QoS is an important new element introduced in this definition, although earlier definitions have implicitly indicated at it.

Re-definition of the term “grid computing” twice over the period of nearly 5 years suggests that this is still an evolving field. However, all the three definitions are consistent in terms of their focus on large-scale computing. Thus, Foster and Kesselman (1998) mention “access to high-end computational resources”, Foster et al. (2001) refer to “large-scale resource sharing” and, finally, Foster and Kesselman (2004) highlight “delivery of nontrivial QoS”. This focus on large scale computing makes grid computing an enabling technology for *eScience* (Hey and Trefethen, 2002). *e-Science* is large scale science that is increasingly being carried out through global collaborations, and which requires access to very large data sets and computing resources distributed across a wide geographical area (National e-Science Centre, 2001). Some of the *e-Science* projects using grid technology are presented in table 1 below.

Table 1: e-Science projects that use grid computing

e-Science Project	Disciple	Details	Reference
LHC e-Science project, CERN (Geneva)	Particle physics	The LHC (Large Hadron Collider) project features a high-luminosity accelerator and four state-of-the-art particle physics collision detectors (ALICE, ATLAS, CMS, LHCb). The four LHC experiments, named after the four collision detectors, are designed to be able to study particle physics under conditions well beyond any other previous experiment. When the LHC becomes operational in 2007 it will produce roughly 15 Petabytes (15 million Gigabytes) of data annually. The data will be accessed and analysed by thousands of scientists (ATLAS alone has about 1700 scientific collaborators from more than 150 institutions). <i>Author's Comment: As of October 2006, the LHC collaboration consists of scientists and resource providers in 40 countries.</i>	(Lamanna, 2004) and (LCG, 2007a)
NEES e-Science project, USA	Earthquake engineering	The NEES (Network for Earthquake Engineering Simulation) project links earthquake researchers across the U.S. with leading-edge computing	(Spencer et al., 2004)

e-Science Project	Disciple	Details	Reference
		resources and research equipment like supercomputers, data storage, networks, visualization displays, sensors and instruments, application code, among others. This allows collaborative teams (including remote participants) to plan, perform, and publish their experiments.	
ESG e-Science project, USA	Climatology	In the ESG (Earth System Grid) project, global climate models are used to simulate climate, and experiments are executed continuously on an array of distributed supercomputers. The resulting data archive, spread over several sites, currently contains upwards of 100 TB of simulation data. The ESG project is a collaborative interdisciplinary project.	(Bernholdt, 2005)
BIRN e-Science project, USA	Medical	The BIRN (Biomedical Informatics Research Network) project is establishing an information technology infrastructure that will pool together research facilities, instrumentation resources, domain expertise and regional information to better tackle diseases.	(Ellisman and Peltier, 2004)

The adoption of grid computing outside e-Science projects has been limited. There are only a few examples in the literature of the use of grids in industry for inter-organizational collaborative work (i.e., access to shared VO resources for day to day operations of an organization) or collaborative research. Arguably, this is best illustrated by the fact that the majority of the research papers related to “grid applications” that are listed on the website of Globus Alliance (Globus Alliance, 2007b), a well recognised community of organizations and individuals that are involved in the research and development of grid computing technologies, are about the use of grid computing in e-Science projects.

One exception to this is the Distributed Aircraft Maintenance Environment (DAME) project that has developed a distributed aircraft engine diagnosis environment as a proof of concept demonstration for Grid computing (Jackson, 2003). This project has three industrial partners (Rolls-Royce plc, Data Systems and Solutions, and Cybula) and four academic partners (Universities of York, Leeds, Sheffield and Oxford). DAME is designed to use grid computing to store terabytes of engine sensor data, which are generated by aircraft fleets during flight, in distributed data repositories and to make them accessible for engine health monitoring services. Other ways in which grid computing technologies have been used in this project can be found from the cited paper.

This section of the thesis has defined grid computing and has highlighted its prevalence in e-Science projects. Before concluding, it is worth adding that the concept of running user applications using multiple distributed resources has been around for as long as computer networks itself. For example, distributed systems like the Resource Sharing Executive (RSEXEC) system (Forsdick et al., 1978), the National Software Works network operating system (Forsdick et al., 1978), the V distributed operating system (Cheriton, 1988), the Amoeba distributed operating system (Tanenbaum et al., 1990), Legion (Grimshaw and Wulf, 1996), the Uniform Interface to Computing Resources (UNICORE) system (Almond and

Snelling, 1999), among others, have been in existence for decades; however, there are some key differences between the current approach to grids and the former approaches (Schopf and Nitzberg, 2002).

- Grid computing facilitates use of heterogeneous hardware and software resources. For example, different hardware architectures running different operating systems and different versions of applications can all be combined together to form the grid. This is being made possible through the development of standardized, interoperable grid protocols and interfaces, and the implementation of the same in the form of grid computing middleware for different hardware architectures and operating systems. The previous approaches generally provided non-interoperable and custom solutions (Foster and Kesselman, 2004).
- The grid approach is about resource sharing, and unlike previous approaches that concentrated mainly on sharing computers and networks, it also focuses on the sharing of data, specialized instruments, applications, etc.
- The grid approach advocates site autonomy for the different administrative domains that collectively provide resources for grid computing. Thus, each administrative domain has complete control over its local resources, policies governing use of such resources, the user accounts that are maintained, etc.
- The grid approach focuses on the users. It enables them to select resources that are best suited to fulfil the requirements of their applications. The previous approaches were mainly driven by the requirements of the resource providers, for example, to maximize utilization and throughput.

The next section looks at grid computing from the point of view of those involved in executing their applications over the grid – the grid users (subsequently referred to only as the users).

2.2.1 Grid computing from the perspective of the users

The users perceive distributed grid resources as one single system that is capable of processing their computation and data intensive jobs. This is graphically illustrated in figure 2. By logging into one computer (which can be an office computer, a personal laptop, etc.), the users expect to seamlessly access the underlying grid resources like computing clusters, disk arrays, applications, instruments, databases, etc. This section presents an overview, from the point of view of the users, of the services that can be provided by grids and the grid-specific mechanisms that are involved in accessing them.

Baker et al. (2002) identify the following five *basic grid services* that can be provided by grids.

- **Computation Services:** These services allow user jobs (these can be considered as executable programs written by the user) to be run on distributed computational resources. A grid providing computational services is often called a *Computational Grid*.

- **Data Services:** These services provide secured access to datasets. In order to create the illusion of a mass storage, these datasets can be replicated, catalogued or even stored in different locations. The processing of the datasets is normally carried out using computational grids. *Data Grids* is a term that is used to define computational grids that process massive datasets.
- **Application Services:** These services provide access to remote software and libraries. They build on computational and data services that are provided by the grid.
- **Information Services:** These services use the computational, data and application services to present data with meaning (i.e., information). For example, the output generated by the simulation can be visualized.
- **Knowledge Services:** Knowledge can be defined as information applied to achieve a goal, solve a problem or execute a decision. Data grids can be used to mine for knowledge using data that is present in the databases.

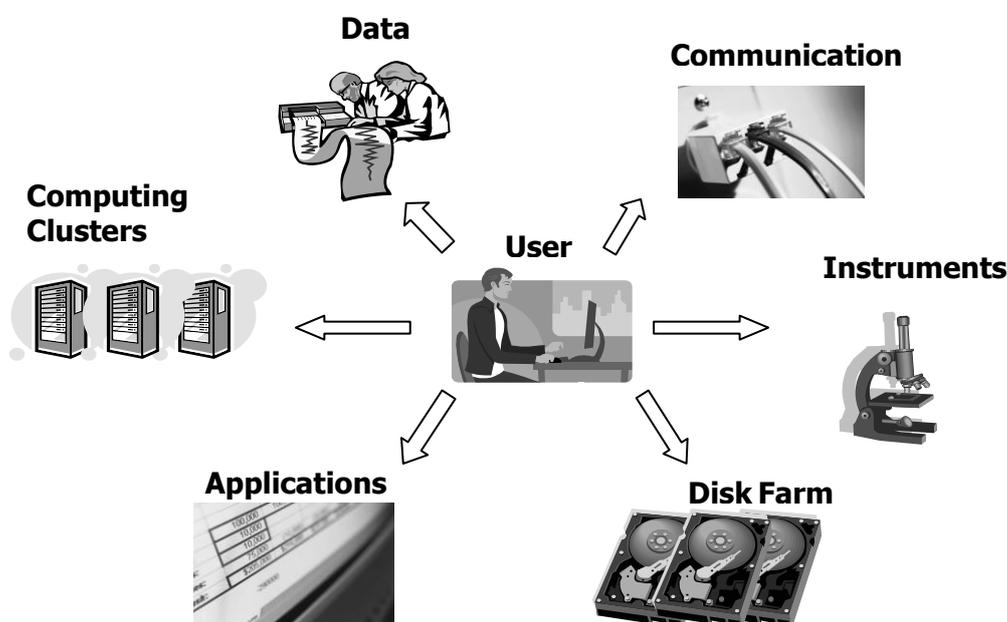


Figure 2: Users' view of grid computing

The users will have to interact with the grid system using grid-specific mechanisms in order to access and utilize the services that have been identified above. These grid-defined mechanisms are referred to as *core grid mechanisms* in this thesis. This thesis looks only at the procedures for accessing computation services, as this introduces some important core grid mechanisms (scheduling, brokering, etc.) that will be used in the subsequent discussions. However, most of these mechanisms are also usually used for accessing the other grid services that have been described in this section.

The users generally access computation service through *job submission*. Jobs generally consist of executable code and associated data, wherein the code acts on the data to produce some output. Jobs are submitted to the grid by the user using a local computer (also referred to as the job submission node). The interaction between the users and the grid in

successful execution of the user jobs will involve all or some of the following core grid mechanisms:

- **Authentication and single sign-on:** Users gain access to grid resources by authenticating themselves to just one resource. This is made possible through the generation of short-lived (usually 12-24 hours) *proxy-certificates* that enable dynamic assignment of new user identity certificates for each new resource accessed by the user or by a user program (Welch et al., 2003).
- **Authorization:** Users can access only those resources for which they have permission. Two grid-defined mechanisms for authorization are *gridmap-files* and *Virtual Organization Membership Service (VOMS)* (Alfieri et al., 2005). Authentication and authorization are grid security mechanisms.
- **Grid information service:** Information pertaining to grid resources is maintained by the grid information service (Czajkowski et al., 2001). This information is continually updated to reflect the availability of resources. Other information like the configurations of the machines (e.g., number of CPUs, RAM), the software available (e.g., MPI libraries, Java runtime environment), etc. are also generally kept.
- **Resource discovery:** Through the grid information service the users discover available resources for running their jobs. Resource discovery is not necessary if the users have already decided on the resource over which to execute their jobs.
- **Resource allocation:** User jobs are allocated to resources that have been discovered and that are considered appropriate for the execution of the jobs.
- **Job submission:** User jobs are submitted on the allocated resource. This is normally achieved through batch submission systems like Portable Batch System (PBS) (Bayucan et al., 1999), Load Sharing Facility (LSF) (Zhou, 1992), LoadLeveler (Kannan, 2001), etc. running on the local computation resource. A local batch submission system, on the one hand, allows the administrator of a resource to define policies with regard to its use for the execution of different jobs; and on the other hand it provides a mechanism that ensures that the user job will have access to resources required to complete its execution (Bayucan et al., 1999).
- **Data staging:** User data is moved from the job submitting node to the computation node, as local access to data at the computing node generally reduces execution time.
- **Job monitoring:** Users can monitor the progress of their jobs.
- **Output retrieval:** The outputs of the computations are retrieved by the users.
- **Resource brokering:** Resource discovery and job submission can be done on behalf of the users by a Resource Broker (RB) component of the grid system, if present. The RB is responsible for matching job requirements with resource capabilities and for assigning jobs to the resources accordingly (Berlich et al., 2005). Thus, the RB allows the submission of user jobs to different local batch submission systems that are running on various grid resources (figure 3). *Nimrod/G* is an example of a RB that has extensively been used over grids for parametric computing (task farming) by applications in the field

of bio-informatics, operations research, etc. (Buyya et al., 2000). The concept of RB has relevance to subsequent discussions.

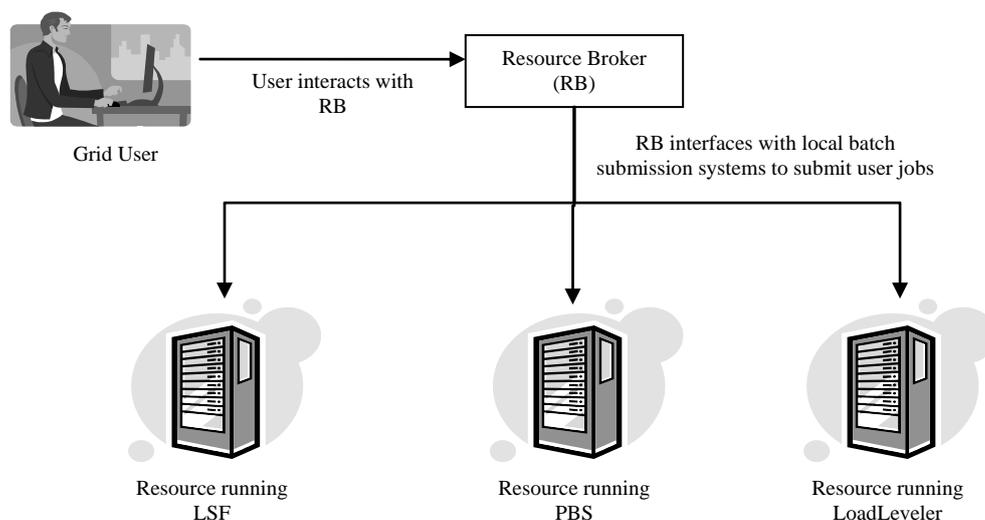


Figure 3: Interaction between resource broker and grid resources

This section of the thesis has presented an overview of basic grid services and the means to access them, using core grid mechanisms, from the point of view of the grid users. The next section will discuss some of the higher-level grid services that built on the basic grid services and provide the users with higher-level functionality.

2.2.2 Higher-level grid services

The basic grid services outlined in section 2.2.1 (for example, computation service, data service, application service, etc.) can be used to offer higher level, grid-supported functionality to the user applications. Using multiple grid nodes that are installed with Parallel Virtual Machine (PVM) (Geist et. al, 1994) and / or parallel computing libraries based on Message Passing Interface (MPI) (Argonne National Laboratory, 2007), the user is generally able to execute parallel applications over the grid. For example, Huang et al. (2006) have implemented a grid-based parallel visualization service to visualize massive datasets of scientific data in parallel. They have used the MPICH-G2 (Karonis et al., 2003) implementation of MPI over Globus middleware (discussed in section 2.2.3.1) for parallel execution of their application. Thus, it can be argued that they utilize three basic grid services, namely, computational service (for parallel processing), data service (to make available scientific datasets) and application service (for accessing MPICH-G2 libraries installed over different grid nodes), to provide a high-level information visualization service that abstracts the underlying basic grid services. Grid computing middleware that provide parallel computation support to user applications include Globus, Condor (discussed in sections 2.2.3.2 and 2.10) and InteGrade (Goldchleger et al., 2004). Grid support for executing parallel applications will henceforth be referred to as *parallel computation service*.

Grid computing provides access to multiple computing resources and therefore it is generally possible to execute different applications over various grid nodes. This is different from parallel computation using grids (described in the earlier paragraph) where one application is executed co-operatively by multiple grid resources. The ability to run different applications concurrently over grids facilitates the execution of applications that are based on the *master-worker* distributed computing architecture. This architecture (also referred to as task farming architecture) consists of one master entity and multiple workers entities, wherein the master entity decomposes the problem into small tasks, distributes these tasks among multiple worker processes and gathers the partial results to produce the final result of the computation; and the worker entities receive messages from the master with the next task (or request next task from the master), process the task and send back the result to the master (Heymann et al., 2000). Goux et al. (2000) describe a software framework called *MW* that allows users to parallelize computations using the master-worker paradigm on the computational grid. *MW* interfaces the user application (the user application consists of two separate components, namely master program and worker program) with underlying grid middleware. Thus, the user applications use the *MW* software framework to draw on computation resources required for their execution. The *AppLeS (Application Level Scheduling) Master-Worker Application Template*, or *AMWAT*, is yet another software framework that targets deployment of small and medium-scale master-worker applications (Berman et al., 2003). Grid support for executing master-worker type applications will subsequently be referred to as *task farming service*.

Computational steering service is yet another high-level service that can be composed of basic grid services. Unlike traditional non-interactive programs that are executed over the grid, computational steering provides a way for the users to interact with grid applications while they are running (Brooke et al., 2003). This allows a user to steer the execution of a remote application based on the intermediate outputs being generated by it. Computational steering usually necessitates concurrent execution of two or more programs over the grid, wherein one program (client) provides the interface to steer the execution of one or more remote programs. For example, the *gViz* e-Science project has demonstrated the use of *gViz computational steering library* in an environment disaster simulation and visualization application, where a client program is used to manipulate the wind directions while the simulation and visualization components are running over the grid (Brodie et al., 2004). Similarly, the RealityGrid project has implemented a *computational-steering library* and a *steering client*, where the steering client is used to steer one or more software components (Brooke et al., 2003). The software components, including the RealityGrid steering client, utilize the RealityGrid computational steering library for this purpose. Figure 4 shows an example where the client is being used to computationally steer a simulation and a visualization component.

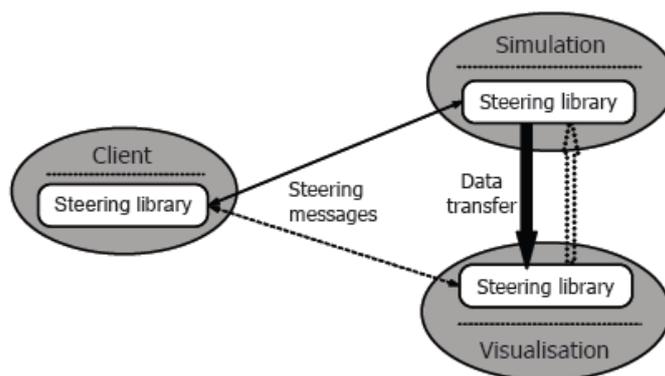


Figure 4: Example of remote steering in RealityGrid (Brooke et al., 2003)

Grid computing technologies can be used to integrate previously uncoupled resources and applications such as sensor networks, High Performance Computing (HPC) resources, simulation and visualization applications, distributed datasets, command and control systems, etc. This is referred to as grid-facilitated *integration service* in this thesis. The FireGrid project, for example, utilizes the integration capability of the grid to develop real time fire emergency response systems (Berry et al., 2005). It uses the computation service, a basic grid service, to gain on-demand access to HPC resources and to run computational fluid dynamics fire models using the provisioned resources. The simulations are steered using data sent over wireless sensors networks (pre-deployed at the location of fire emergency) and the results of the computations are input to a real-time command and control (C^2) system. The C^2 system is used for emergency response and evacuation planning. The FireGrid also utilizes the knowledge service, another basic grid service outlined by Baker et al. (2002), for mining data pertaining to key events.

A grid portal is a web-based application that is enhanced with the necessary software to enable it to communicate with grid services made available by the grid middleware (Novotny, 2002). It provides the users with higher-level abstraction to the underlying grid services. The web browsers provide an easy-to-use, graphical environment through which the users can interact with the grid middleware. Furthermore, grid portals make it possible for the users to access grids from virtually any computer that is connected through the Internet. Examples of grid portals include the P-GRADE (Parallel Grid Run-time and Application Development Environment) portal (Németh et al., 2004), the NGS (National Grid Service) portal (Yang et al., 2005), the GENIUS (Grid Enabled web eNvironment for site Independent User job Submission) grid portal (Barbera et al., 2003) and the Legion grid portal (Natrajan et al., 2002). The use of grid portals to enable convenient access to grid middleware is subsequently referred to as *grid portal service*.

The applications that are executed over grid resources can have dependencies among them. For example, the output of one application can be the input to another application (sequential dependency). Such dependencies between applications can be maintained using workflows

and workflow management systems. Workflows are concerned with the automation of procedures whereby files and data are passed between applications following a defined set of rules to achieve an overall goal; and workflow management systems are responsible for defining, managing and executing such workflows over computational resources (Yu and Buyya, 2006). Examples of workflow management systems include Condor DAGMan (discussed in section 2.10.4), Taverna (Oinn et al., 2004), Pegasus (Deelman et al., 2004) and Gridbus workflow enactment engine (Yu and Buyya, 2004). Grid support for executing workflows will subsequently be referred to as *workflow service*.

Grid computing facilitates collaboration among VOs. This collaboration can take various forms. At the most basic level it can be collaboration through co-operative use of grid resources. Table 1 in section 2.2 lists four such examples of collaborative resource sharing in e-Science projects, namely, LHC (Lamanna, 2004), NEES (Spencer et al., 2004), ESG (Bernholdt, 2005) and BIRN (Ellisman and Peltier, 2004).

Collaboration in the grid environment can take the form of users publishing their user-developed web services (think of these as user applications that can be accessed using standard Internet protocols and open standards) for other users to access. Web services are a web-based technology that is increasingly being used to implement Service Oriented Architectures (SOA) (Mahmoud, 2005). Web services support machine-to-machine interaction over a network using SOAP messages sent over Hyper Text Transfer Protocol (HTTP) and web-related standards (World Wide Web Consortium, 2004). SOAP is a lightweight Extensible Markup Language (XML)-based protocol for exchange of information in a decentralized, distributed environment (World Wide Web Consortium, 2000). OGSA (Open Grid Services Architecture)-complaint grid middleware like GT-4 usually provide containers (hosting environments) to host the user-developed web services, and provide mechanisms for service providers to register their web services through use of service registries (service publication), mechanisms for service consumers to search for services in the registries (service discovery) and mechanisms to invoke the services when a suitable match is found (service invocation). Both OGSA and GT-4 are further discussed in section 2.2.3.1.

Figure 5 shows the GT-4 container hosting both user-developed web services (“custom web services” and “custom WSRF [Web Services Resource Framework] web services”) and GT-4 developed web services (“GT-4 WSRF web services”). It depicts the service registry as “registry administration” and the applications used by service consumers to access both the user-developed and the GT-4 developed web services as “user applications”. The reader is referred to Globus Alliance (2007a) for an overview on SOA, web services and WSRF.

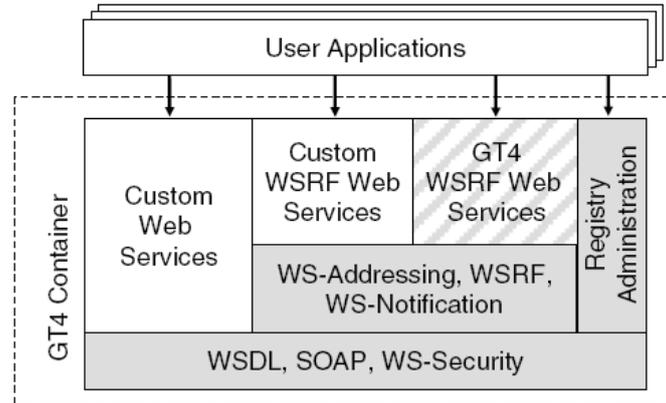


Figure 5: GT-4 container hosting user-defined web services (Foster, 2006)

Another form of grid-facilitated collaboration could be virtual meeting support provided through integration of audio, video and messaging capabilities with grid middleware. An example of this is the *Access Grid Collaboration System*. This is based on technology developed by the Argonne National Laboratory's (ANL) Futures Laboratory Group (FLG). Access grid is primarily meant for group-to-group human interaction through the use of interactive presentation and software environments, remote visualization environments, large-format multimedia displays, among others (Stevens and FLG, 2004). Screenshot 1 below shows Access Grid being used for an interactive virtual meeting. In this thesis, the use of grid technology to facilitate collaboration is referred to as *collaboration service*.



Screenshot 1: Group-based collaboration using Access Grid

This section has discussed some of the higher-level grid services that can be accessed by users through use of grid computing. The services discussed were parallel computation service, task farming service, computational steering service, integration service, grid portal

service, workflow service and collaboration service. Most of these services built on the basic grid services that were discussed in section 2.2.1.

The basic and the higher-level grid services can be provided through use of grid computing software. This software is commonly referred to as grid computing middleware and is discussed next.

2.2.3 Grid middleware

A grid middleware is a distributed computing software that integrates network-connected computing resources (computer clusters, data servers, standalone PCs, sensor networks, etc.), that may span multiple administrative domains, with the objective of making the combined resource pool available to user applications for number crunching, remote data access, remote application access, among others. A grid middleware is what makes grid computing possible. With multiple VOs involved in joint research collaborations, issues pertaining to security (authentication and authorization), resource management, job monitoring, secure file transfers, etc. are of paramount importance. Thus, in addition to making available a seamless distributed computing infrastructure to cater to the computing needs of the grid user, the grid middleware usually provides mechanisms for security, job submission, job monitoring, resource management and file transfers, among others. This section gives an overview of grid middleware that are commonly installed on distributed computing resources to create an underlying infrastructure for grid computing. The operating system support for each middleware is also highlighted.

2.2.3.1 Globus middleware

The origin of Globus middleware can arguably be traced back to 1995, when 17 supercomputing centres, data centres and virtual reality laboratories across North America were linked together through the *I-WAY* network to demonstrate distributed execution of a number of supercomputing applications (Berlich et al., 2005). A management and application programming environment called *I-Soft* was developed as part the *I-WAY* experiments and was deployed at most of the 17 *I-WAY* sites (Foster et al., 1996). *I-Soft* can thus be considered as the precursor to Globus. Globus has since come a long way with the current version of the Globus middleware being version 4.

The Globus middleware is an open architecture and an open source set of services and software libraries, developed in consultation with the user community, which supports grids and grid applications (Foster et al., 2002). It implements a set of components (based on standard grid protocols and interfaces) that provide basic grid services like authentication, resource discovery, resource access, resource management, data management, communication, etc., and a set of software libraries, both of which facilitate the construction of more sophisticated grid middleware. As such, Globus is regarded more as a toolkit for the development of other grid middleware rather than a ready-to-use grid solution (Berlich et al.,

2005). Globus is thus referred to as Globus Toolkit (GT) in different versions of the middleware, viz., GT-2, GT-4, etc. The majority of the middleware discussed later in this section either includes components from Globus or are an extension of Globus itself. Subsequent discussions on Globus are extensively referenced from Foster and Kesselman (2004), unless otherwise stated.

A few of the grid protocols that are implemented by Globus and its purpose are described next.

- The Grid Security Infrastructure (GSI) protocol supports single sign-on user authentication.
- The Grid Resource Allocation and Management (GRAM) protocol is for allocation and management of user jobs on remote resources.
- The Monitoring and Discovery Service (MDS-2) provides a framework for discovering and accessing information like server configuration information, networks status, etc.
- The GridFTP protocol is an extension of the popular File Transfer Protocol (FTP) protocol and supports partial and parallel file access.

It has to be added that some of these protocols like GridFTP and GSI were first defined and implemented by Globus version 2 (GT-2), before they were subsequently reviewed within the standards bodies and recognised as standards. This is hardly surprising because from 1997 onwards GT-2 was generally considered the de facto standard for grid computing because of its focus on reusability and interoperability with other grid systems. A community-wide grid protocol standardization effort started in around 2001 with the emergence of the Global Grid Forum, now called the Open Grid Forum (Open Grid Forum, 2007). This ultimately produced the *Open Grid Services Architecture (OGSA)* - a service oriented framework, defined by a set of community-developed standards, for the development of grid middleware. OGSA builds on concepts and technologies from both the grid and web services communities with the objective of providing an extensible set of grid services that VOs can aggregate in various ways (Foster et al., 2002). It is widely believed that OGSA-based grid middleware will encourage the adoption of grid computing technology in industry and will facilitate the development of grid-based commercial applications. Globus toolkit versions 3 and 4 (GT-3, GT-4) are both based on OGSA. A short overview of GT-4 is presented next.

GT-4 provides the following sets of components (Foster, 2006).

- A set of Globus-developed web services implementation of core grid services for resource management (like WSRF implementation of GRAM), data access and movement (Reliable File Transfer [RFT], OGSA-DAI [Antonioletti et al., 2005]), replication management (Data Replication Server [DRS]), monitoring and discovery service (Index, Trigger, WebMDS.), credential management (Delegation, SimpleCA) and instrument management (Globus Teleoperations Control Protocol [GTCP]).

- A set of Globus-developed non-web services implementations of core grid services for resource management (GRAM), data access and movement (GridFTP), replication management (Replica Location Service [RLS]), monitoring and discovery service (MDS-2) and credential management (MyProxy).
- Three different containers, viz., Java container, Python container and C container, to host user-developed services written in Java, Python and C respectively. These containers provide implementations of security, management, discovery, state management, and other mechanisms frequently required when building user-defined services.
- A set of client libraries that allow user programs in Java, Python and C to invoke operations on both Globus-developed and user-developed services.

The GT-4 architecture is shown in figure 6 below. The figure shows only some of the components described above. More information on the individual components of GT-4 can be found in the “GT-4 administration guide” (Globus Alliance, 2005).

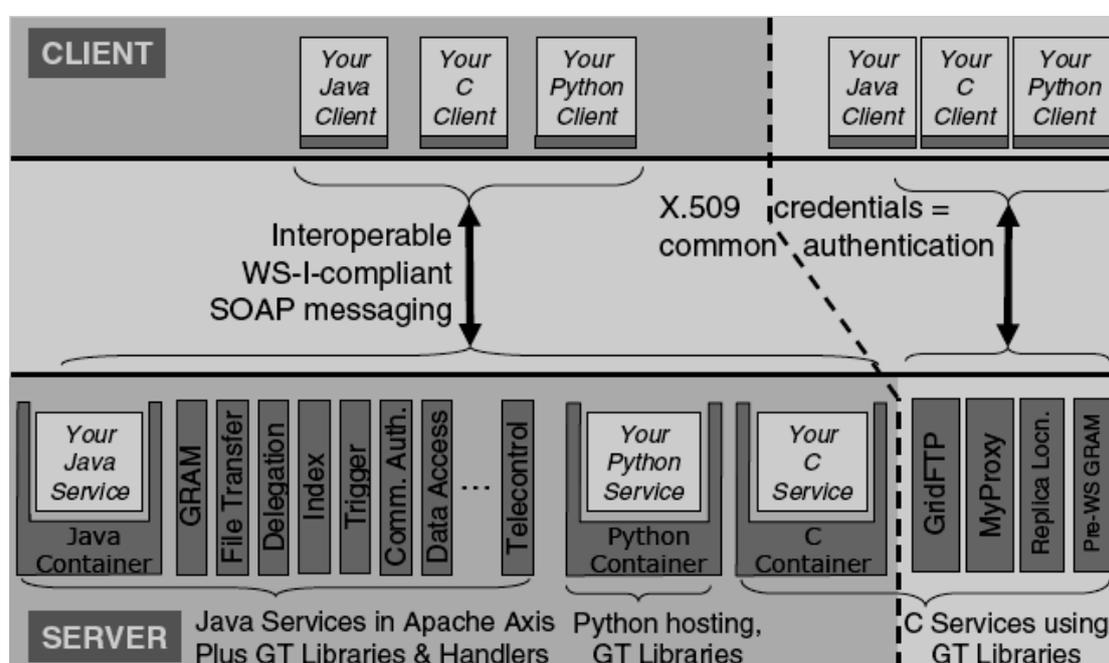


Figure 6: GT-4 architecture showing the different components (Foster, 2006)

GT-4 is supported on UNIX, Linux and Windows operating systems. However, not all components can be installed on Windows. For example, neither the pre-web services implementations of the resource management component of GT-4 (GRAM), nor the WSRF implementations GRAM can be installed on a Windows system. Furthermore, the non-web services GT-4 implementations for security (MyProxy), file transfer (GridFTP), replication, and information service (MDS-2) can only be run on UNIX and Linux platforms (Globus Alliance, 2005).

2.2.3.2 Condor middleware

Condor is a job scheduling system that is designed to maximize the utilization of collections of networked PCs, referred to as a Condor Pool, through identification of idle resources and

scheduling background user jobs on them (Litzkow et al., 1988). Although Condor was originally designed to harness unutilized CPU-cycles from non-dedicated PCs within an organization, the same design can be used to manage dedicated compute clusters. Using the *Condor-G* extension to Globus it is possible to operate Condor across organizational boundaries (Berlich et al., 2005).

Condor is supported on UNIX, Linux and Windows platforms. However, like Globus, not all components of Condor can be installed on a Windows machine. For example, Condor does not support several Condor execution environments like standard universe, PVM universe, GT-4 grid type, LSF grid type, etc. on Windows (Condor Version 6.9.1 Manual, 2007a). The reader is referred to section 2.10 for a detailed discussion on Condor.

2.2.3.3 European Data Grid (EDG) middleware

The EU-funded European Data Grid (EDG) project was a three year project (2001-2004) that was started with the goal of developing technological infrastructure for facilitation of e-Science collaborations in Europe. The grid computing middleware developed during this project is commonly referred to as the EDG middleware. The EDG middleware itself is based on GT-2, but in addition to Globus-supported standard grid features like grid security infrastructure, grid information service, resource discovery and monitoring, job submission and management, etc., it extends Globus to offer high functionality middleware services like resource brokering and replication management (Berlich et al., 2005). Resource brokering and replication management services are implemented using the *Resource Broker (RB)* and *Replication Management Tools (RMT)* respectively, both of which are integrated with the EDG middleware. Through the RB component, EDG middleware implements the “push” middleware architecture wherein the RB periodically polls the computing resources to find out the load levels and decide on whether new jobs are to be assigned to the resources (Berlich et al., 2005).

After the completion of the EDG project in 2004, some of the EDG middleware components, notably RB and RMT, have been further developed as part of other EU-funded grid projects like the Enabling Grids for E-science (EGEE) project (see section 2.2.4). The EDG middleware has only been tested on *RedHat Linux 7.3* (EDG WP6 Integration Team, 2003).

For subsequent discussions in this thesis relating to grid middleware, a distinction between “pull” and “push” middleware architecture is now presented. “pull” and “push” are two different methods (models, approaches, architectures, mechanisms) for scheduling tasks (jobs) on resources (Hantz and Guyennet, 2005). The tasks are scheduled by a middleware component that can be referred to by various names, for example, job scheduler, workload management system, task dispatcher, master process, etc. For the purpose of this research it is sufficient to view the task scheduling component as an integrated part of the grid middleware. In a “pull” model the computing resources request jobs from a central resource which maintains the job

queue; whereas in a “push” model one central resource schedules jobs on the available resources and tries to centrally optimize the allocation of jobs between the resources (Garonne et al., 2004). In the decentralized “pull” model the system state information is maintained by each resource, whereas in the centralized “push” model state information of all the resources is maintained at a central resource (Garonne et al., 2005).

2.2.3.4 Virtual Data Toolkit (VDT) middleware

Virtual Data Toolkit (VDT) is a grid middleware primarily meant for the US Open Science Grid (see section 2.2.4). It is a combined package of various grid middleware components, including Globus and Condor, and other utilities. The goal of VDT is to provide users with a middleware that is thoroughly tested, simple to install and maintain, and easy to use. The latest version of VDT (version 1.6.1) supports only Linux-based platforms like *Debian Linux*, *Fedora Core Linux*, *RedHat Enterprise Linux*, *Rocks Linux*, *Scientific Linux* and *SUSE Linux* (Virtual Data Toolkit, 2007). More information on the individual VDT components can be found from the cited reference.

2.2.3.5 gLite middleware

The development of gLite middleware is being supported by the European Commission funded EGEE project. gLite is primarily being developed for the LHC Computation Grid (LCG) and the EGEE grids (see section 2.2.4). Twelve academic and industrial partners are involved in the development of gLite. These include the European Organization for Nuclear Research (CERN), the National Institute of Nuclear Physics (INFN, Italy), National Center for Scientific Research (CNRS, France), Council for the Central Laboratory of the Research Councils (CCLRC, UK), and National Institute for Nuclear Physics and High Energy Physics (NIKHEF, The Netherlands).

The *gLite-3* middleware (the latest version of gLite) uses components developed from several other grid projects like Globus, Condor and EDG. *gLite-3* is based on the web services architecture and its underlying computing resources are referred to as Computing Elements, or gLite CE for short. On one hand, *gLite-3* middleware supports the “pull” architecture that empowers the gLite CEs to decide the best time to start a grid job; on the other hand, a RB can be used to “push” jobs just as EDG middleware (Berlich et al., 2005). Another middleware which uses the “pull” architecture for its RB is AliEn (a middleware primarily developed for LHC ALICE experiment – see section 2.2.4). Because of its “pull” implementation the AliEn RB does not need to know the status of all resources in the system (Saiz et al., 2003). *GLite-3* middleware is presently supported only on the *Scientific Linux* operating system (Burke et al., 2007).

2.2.3.6 LCG-2 middleware

LCG-2 is the middleware for the LCG and the EGEE grids. It is a precursor to the gLite middleware, and is being gradually replaced by gLite on both these production grids. The

operating systems supported by LCG are *Red Hat 7.3* and *Scientific Linux 3* (Peris et al., 2005).

2.2.3.7 OMII middleware

The Open Middleware Infrastructure Institute (OMII), based in the University of Southampton and established as part of the five year (starting from late 2001) £250 million UK e-Science core program, is mainly responsible for ensuring “production-level” quality standards for grid middleware components being delivered by various UK e-Science projects, ensuring that the components are well documented and maintained in a middleware repository, undertaking integration testing of these UK developed middleware components for interoperability with components produced outside of the UK, and for testing the components to ensure interoperability with open grid and web services standards. (Atkinson et al., 2005).

In order to achieve “production-level” quality of middleware components, OMII works jointly with the e-Science project teams in all phases of software development and/or employs its own pool of software engineers to work on the software artifacts after they have been delivered by the grid projects. Some of these components are collectively released as a combined, quality assured, easy to install OMII software release. This software is also referred to as the OMII middleware and it presently consists of two specific releases, viz., *OMII server release* and *OMII client release*. The OMII grid middleware is open source and can be downloaded from the OMII website <<http://www.omii.ac.uk/>> for deployment by the users. Some of the software components that are part of the OMII middleware are GridSAM job submission and monitoring service, Taverna workflow tool (Oinn et al., 2004), BPEL workflow editor and execution engine, application hosting environment, etc. More details of these software components can be found on the OMII website.

The client and the server parts of OMII middleware are installed on the computers of the grid clients (users) and grid service providers respectively. The client typically accesses the computation resources and applications made available by the grid service provider through the OMII client, via the OMII server (figure 7).

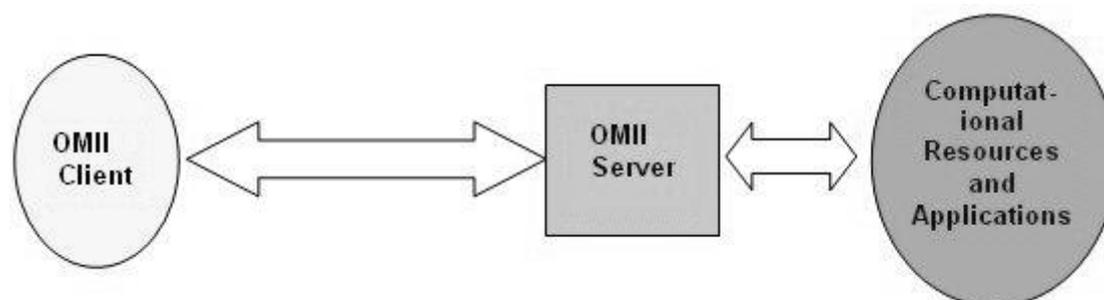


Figure 7: Conceptual view of users and service providers (OMII, 2006b)

The client part of OMII middleware can be installed on different distributions of Linux, Windows and Apple Macintosh operating systems. However, the server part can only be

installed on Linux flavor operating systems and on Apple Macintosh (OMII, 2006a). Both the client and the server parts require Java to be pre-installed on the target machines.

2.2.4 Production grids

Production grids can be defined as grid computing infrastructures that have transitioned from being “research and development” test beds to being fully-functional grid environments, offering users round-the-clock availability at sustained throughput levels. Production grids are usually supported by a team that is responsible for the day-to-day maintenance of the grid (including upgrading software), solving technical problems associated with the grid, helping users through help-desk support, creating user documents, conducting training courses for knowledge dissemination purposes, among others. This section gives an overview of some of the largest production grids in the world and highlights the grid middleware running on them. This information is presented in table 2 below.

Table 2: Examples of production grids

Grid Name	Purpose	Infrastructure	Grid Middleware	Reference
LCG (LHC Computing Grid)	The purpose of LCG is to provide computation and storage resources for four LHC particle physics experiments, viz., ALICE, ATLAS, CMS, LHCb, at the European Organization for Nuclear Research (CERN) near Geneva.	The LHC infrastructure is arranged in a four-tier hierarchy. Tier 0 is located at CERN and is responsible for the storage of all raw data. Tier 1 centres are supercomputing facilities that complement Tier 0’s capacity and act as data distribution centres for Tier 2. Tier 2 centres provide facilities to analyze data. Tier 3 centers consist of physicists and other users who access data from their PCs through the Tier 2 centres. The LHC infrastructure comprises of resources from other national and international production grids like EGEE and Grid3 (see below). At present the LCG grid spans over 200 sites around the world and has access to more than 30,000 CPUs and 20 PB of data storage capacity. <i>Author’s Comment: As of October 2006, LHC consisted of 12 Tier-0 and Tier-1 centres and 38 Tier-2 centres. When the LHC becomes operational in 2007 it will require 100K of today’s fastest CPU’s.</i>	LCG-2 / gLite	(Lamanna, 2004), (LCG, 2007a), (LCG, 2007b) and (Burke et al., 2007)
OSG (Open Science Grid), USA	Research in bioinformatics, medical imaging, nanotechnology, physics, etc.	50 sites across United States, Asia and South America. Note: OSG infrastructure is also used for the ATLAS and CMS experiments.	VDT	(Open Science Grid, 2007)
DOE Science Grid , USA	Scientific computing in multiple disciplines across DOE (US Department of Energy).	Aims to provide access to advanced resources at multiple DOE resource sites: initially, computers and storage systems at Argonne National Laboratory (ANL), Lawrence Berkeley National Laboratory (LBNL), National Energy Research Scientific Computing Centre (NERSC), Oak Ridge National Laboratory (ORNL), and Pacific Northwest National Laboratory (PNNL); In time, the DOE Science Grid hopes to incorporate other resource types (e.g., networks) and resources at other laboratories and universities.	SciDAC Collaboratory Software Environment	(Johnston, 2001)

Grid Name	Purpose	Infrastructure	Grid Middleware	Reference
		Author's Comment: DOE Science Grid is now operational.		
IPG (NASA Information Power Grid), USA	Provides support for NASA's scientific and engineering communities.	The IPG will interconnect major computing and data resources at multiple NASA sites. It will provide access to around 300 CPUs and 30-100 Terabytes of storage and is connected through a network of at least 100 Mbits/s. Author's Comment: IPG is now operational.	Globus	(Johnston, 1999)
Tera Grid, USA	Research in genomics, earthquake studies, cosmology, climate and atmospheric simulations, biology, etc.	As of 2003, the Tera Grid infrastructure consists of the National Center for Supercomputing Applications (NCSA), the San Diego Supercomputer Center (SDSC), ANL, California Institute of Technology (Caltech) Center for Advanced Computing Research, and the Pittsburgh Supercomputing Center (PSC).	Globus	(Reed, 2003)
Grid3, USA	Resources are used for high energy physics simulations and for data analyses in bio-chemistry, astronomy, etc.	25 sites across the US and Korea collectively provide more than 2000 CPUs. Note: Grid3 also provides resources for ATLAS and CMS experiments at CERN.	VDT	(Grid3, 2007)
NAREGI (National Research Grid Initiative), Japan	All areas of science and technology. Large-scale nanoscience simulations.	The National Institute of Informatics (NIN) and the Institute of Molecular Science (IMS) aim to operate a dedicated NAREGI test bed with 18 teraflops of computing power distributed over 3000 processors.	NAREGI Middleware	(Matsuoka, 2005)
EGEE (Enabling Grids for E-science) Grid	EGEE Grid infrastructure is ideal for any scientific research.	The EGEE project involves over 90 partner institutions across Europe, Asia and the United States and provides access to over 20,000 CPU and 5 Petabytes of storage.	LCG-2 / gLite	(EGEE, 2007)
EDG (European Data Grid)	Provided intensive computation and analysis of shared large-scale databases across distributed scientific communities (e.g., high energy physics, earth sciences, bio-Informatics, etc.)	CERN, INFN (Italy), CNRS (France), Particle Physics and Astronomy Research Council (UK), NIKHEF (Netherlands) and European Space Agency (ESA). Author's Comment: The EU Data Grid has been superseded by EGEE grid in March, 2004.	EDG Middleware	(Segal, 2000) and (EU-DataGrid, 2004)
NGS (National Grid Service), UK	Production use of computational and data grid resources in all branches of academic research.	NGS provides access to over 2,000 processors and over 36 TB storage capacities. These resources are provided by the Universities of Manchester, Leeds, Oxford and the Rutherford Appleton Laboratory (RAL). The two High Performance Computing (HPC) service providers are UK National HPC Service (CSAR) and the CCLRC Daresbury Laboratory. Author's Comment: The NGS resource base is gradually increasing with more Universities contributing their clusters.	Globus	(Yang et al., 2005)

As can be seen from the above table, most of these production grids have a resource base spanning multiple VOs. These production grids are mainly being used for e-Science projects. It has been noted earlier that there are very few examples of multiple VO-based grid computing in industry. However, it is also true that grid computing middleware like Globus is gradually being introduced within enterprises for processing enterprise-related applications. In this scheme the organizations seek to leverage their existing computing resources using grid middleware. Collaborations, if any, are limited to intra-organizational resource sharing and problem solving. Some of the organizations that use grid computing middleware for their day-to-day operations or integrate these middleware within their own application are listed in table 3 below.

Table 3: Example of organizations that use grid computing middleware

Company: Application	Description	Middleware	Reference
SAP R/3: Internet Pricing and Configurator (IPC), Workforce Management (WFM) and Advanced Planner and Optimizer (APO)	IPC, WFM and APO applications are part of SAP's R/3 product line and are designed to support large numbers of requests generated by interactive clients using Web browsers or from batch processes. Each client request is dispatched to one of a number of worker processes. SAP has modified these applications to use Globus components to discover and reserve the resources used to host those worker processes, and to execute, monitor, and remove the worker processes on those resources.	Globus	(Foster, 2005)
GlobeExplorer: GlobeExplorer	The data portrayed in the maps served by GlobeExplorer originate from multiple sources, e.g. population data, data on street networks, aerial images, satellite Imagery, etc. Globus provides the technology required to integrate data from such heterogeneous resource base.	Globus	(Gentzsch, 2004)
Planet Earth: Butterfly Grid	Butterfly Grid supports massive multiplayer (MMP) games via an on-demand service. Globus is used for staging and maintenance of code; for scheduling, monitoring and termination of processes; and as a distributed monitoring framework using Globus' Monitoring and Discovery Service (MDS-2). Globus Security Infrastructure (GSI) is used for single sign-on into multiple clusters. <i>Author's comments: IBM is also a partner to this project</i>	Globus	(Levine and Wirt, 2004)

The question that has to be asked is: can the use of grid middleware within an organization be termed as grid computing? The grid computing definition (Foster and Kesselman, 2004) the author has been following stipulates collaborative problem solving among VOs and, consequently, across administrative domains. Going by this definition the use of grid computing middleware to access multiple resources within the same organization may not qualify as grid computing. However, there is little agreement over what the term grid computing actually means and there is not one, all-accepted, definition of grid computing. For example, Baker et al. (2002) mention that the "cooperative use of geographically distributed resources unified to act as a single powerful computer" is known by several names such as "metacomputing, scalable computing, global computing, Internet computing, and more recently peer-to-peer or Grid computing" and Luther et al. (2005) refer to enterprise desktop

grid computing, public distributed computing and peer-to-peer computing as different names for Internet computing. However, as will be seen from the discussion presented in the next section, grid computing, enterprise desktop grid computing and Internet / peer-to-peer / public resource computing generally have a different set of objectives that determine the design architecture of their underlying middleware technologies.

2.2.5 Different forms of grid computing

The discussion on grid computing, until this point, has shown that grid infrastructures, middleware and applications have traditionally been geared towards dedicated, centralized, high performance clusters (like Beowulf clusters [Beowulf.org, 2007]) and super computers running on UNIX and Linux flavour operating systems (a notable exception being Condor middleware). This form of grid computing will henceforth be referred to as *cluster-based grid computing*. With the advent of *Microsoft Windows Compute Cluster Server 2003* (Microsoft WCCS, 2007) for parallel HPC (the OS includes Microsoft's implementation of Message Passing Interface – MS-MPI) in 2006, it is expected that grid computing middleware specifically targeted at Windows-based operating systems will be developed in future.

Cluster-based grid computing can be contrasted with *desktop-based grid computing* which refers to the aggregation of non-dedicated, de-centralized, commodity PCs connected through a network and running (mostly) the Microsoft Windows operating system. Middleware for cluster-based grid computing severely limits the ability to effectively utilize the vast majority of Windows-based resources that are common place in both enterprise and home environments, and therefore development of middleware for desktop-based grid computing is important with the growing industry interest in grids (Luther et al., 2005).

Desktop grid computing or desktop grids addresses the potential of harvesting the idle computing resources of desktop PCs for processing of parallel, multi-parameter applications which consist of a lot of instances of the same computation with its own input parameters (Choi et al., 2004). This definition fits with the original design objectives of Condor and it is therefore considered appropriate to regard it as a desktop grid middleware. The idea of harvesting unused CPU cycles has been around for decades with programs such as PARC (Xerox Palo Alto Research Center) WORM, a program that replicated itself on networked PCs and used the idle resources for computation, being developed as early as the 1970s (Chetty and Buyya, 2002).

The desktop grid resources can be part of the same local area network (LAN) or can be geographically dispersed and connected via a global network such as the Internet. Studies have shown that desktop PCs can be under utilized by as much as 75% of the time (Mutka, 1992). This coupled with the widespread availability of desktop computers and the fact that the power of network, storage and computing resources is projected to double every 9, 12,

and 18 months respectively (Casanova, 2002), represents an enormous computing resource. In this thesis the use of a desktop grid within the enterprise is termed as *Enterprise-wide Desktop Grid Computing (EDGC)*. Thus, EDGC refers to a grid infrastructure that is confined to an institutional boundary, where the spare processing capacities of an enterprise's desktop PCs are used to support the execution of the enterprise's applications (Chien et al., 2003). User participation in such a grid is not usually voluntary and is governed by enterprise policy. Applications like Condor, Platform LSF (Zhou, 1992), Entropia DCGrid (Kondo et al., 2004), United Devices GridMP (United Devices, 2007) and Digipede Network (Digipede Technologies, 2006) are all examples of EDGC.

Like EDGC, Internet computing seeks to provide resource virtualization through the aggregation of idle CPU cycles of desktop PCs. But unlike EDGC, where the desktop resources are generally connected to the corporate LAN and used to process enterprise applications, Internet computing infrastructure consists of volunteer resources connected over the Internet and is used either for scientific computation or for the execution of applications from which the user can derive some benefit (for example, sharing music files). This research distinguishes between two forms of Internet computing - Public Resource Computing (PRC) and Peer-to-Peer Computing (P2P) - based on whether the underlying desktop grid infrastructure is used for solving scientific problems or for deriving some user benefit respectively. The different forms of grid computing are shown in figure 8. PRC and P2P computing are described next.

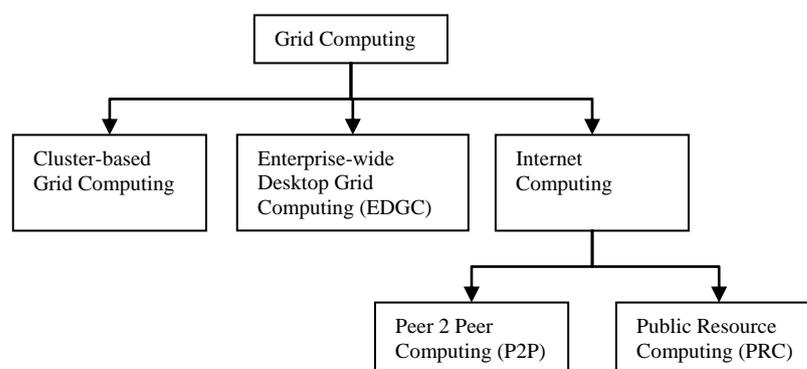


Figure 8: Different forms of grid computing

PRC refers to the utilization of millions of desktop computers primarily to do scientific research (Anderson, 2004). The participants of PRC projects are volunteers who contribute their PCs to science-oriented projects like SETI@home (Anderson et al., 2002) and Climateprediction.net (Christensen et al., 2005). Berkeley Open Infrastructure for Network Computing (BOINC) (BOINC, 2007b) is arguably the most widely used PRC middleware that enables the project participants to download work units from BOINC servers, process them and upload the results back to the servers. The majority of the PRC middleware is supported on Windows. This is not unexpected as PRC projects depend on volunteer computing

resources, and the bulk of these resources presently run on the Windows operating system. The participants of a PRC project are unable to use the underlying desktop grid infrastructure, of which they themselves are part of, to perform their own computations.

P2P computing refers to a non-centralized infrastructure for file sharing over the Internet. P2P networks are created with the resources of the volunteer users (peers), who derive the benefit from such networks as it allows them download files that are shared by other peers. As P2P computing is voluntary, the middleware for such systems should ideally have mechanisms to organize the ad-hoc and dynamic peers in such a way that they can co-operate to provide file sharing services to the P2P community; for example, the P2P middleware should have mechanisms to quickly and efficiently locate files that are distributed among peers (Saroiu et al., 2002). Some of the popular P2P file sharing systems are Gnutella (Sun et al., 2006), KaZaA (Good and Krekelberg, 2003) and in the past, Napster (Giesler and Pohlmann, 2003). They are all supported under the Windows operating system.

Unlike cluster-based grid computing whose user base is generally limited to participants of e-Science projects (or those general users who have in-depth knowledge of grid middleware like Globus) and like PRC / P2P computing whose user base is substantially larger and is comprised of general users contributing their computing resources, enterprise-wide desktop grid computing encourages wider employee participation through resource contribution. Unlike PRC that does not allow the project participants to use the underlying infrastructure to solve their own problems and like cluster-based grid computing that allows users to execute their applications, enterprise-wide desktop grid users can utilize the aggregate resources to process their enterprise-specific jobs. Comparisons, based on multiple criteria, between cluster-based grid computing, desktop-based grid computing and PRC / P2P computing are presented in table 4 below. Some of the differences between cluster-based grid computing and PRC / P2P computing have been referenced from Foster and Iamnitchi (2003).

Table 4: Comparing different forms of grid computing

Comparison based on:	Cluster-based Grid Computing	Enterprise-wide Desktop Grid Computing	P2P / Public Resource Computing
Objective	Pooling of resources that are distributed among VOs and the coordinated use of such resources.	Pooling of resources that are distributed in an enterprise. The coordinated use of such resources by the employees.	Pooling of resources that are available at the edges of the Internet and the coordinated use of such resources.
Grid computing middleware	Globus, Condor, LCG-2, gLite, OMII, Virtual Data Toolkit, etc.	Condor, Entropia DCGrid, Digipede Network, GridMP, etc. <i>Author's Comment: The focus here is on Windows-based middleware.</i>	The middleware is usually specific to a P2P or PRC application. For example, Gnutella and KaZaA P2P file sharing middleware, PRC middleware for Folding@Home project (Pande, 2007). One exception is BOINC which is used for multiple PRC projects like SETI@Home and Climateprediction.net.
Applications	Production grids can be used by many scientific applications,	Enterprise-wide desktop grids are used for the processing of	Each P2P and PRC application normally has its own overlay

Comparison based on:	Cluster-based Grid Computing	Enterprise-wide Desktop Grid Computing	P2P / Public Resource Computing
	spanning multiple projects.	enterprise applications.	network which is geared for that specific application. One exception is BOINC which is used for multiple PRC projects
Target community	Targeted at research communities. Presently, commercial interest in grid computing is on the increase.	Enterprise-wide desktop grids are primarily meant for use by the employees.	PC owners are the target community. They use P2P and PRC applications to either donate their resources for scientific computation (e.g., SETI@Home) or to derive some benefit from it (e.g., file sharing using KaZaA)
User base	Limited to those taking part in collaborative projects.	Usually limited to employees within an organization.	User base can span to hundreds of thousands of users.
Resources	Computing resources used are mostly powerful cluster computers running on UNIX and Linux OS. In addition, resources in a grid can also consist of scientific instruments. Local pools of desktop PCs can be joined into an administrative domain using technologies like Condor, and can be integrated into larger grids.	Resources can be both cluster computers and desktop computers available in an organization. The clusters tend to be dedicated resources. The desktop PCs are mainly non-dedicated employee PCs.	Leverages commodity desktop PCs (running on Windows, Unix, Linux, Macintosh, etc. OS) that have intermittent Internet connection.
Resource administration and sharing	Resources are administered in accordance with well-defined policies. Resource sharing criteria are decided by the VOs. The individual user normally has no control over which VO resources can be shared and / or accessed.	An organization-wide sharing policy may be imposed on the use of employees' PCs. Alternatively the employees can be empowered to take resource sharing decisions pertaining to their PCs.	The end-users (PC owners) are usually the resource administrators and they decide whether to share their resources.
Communication infrastructure	Grid resources are connected over the Internet and proprietary networks, e.g. LambdaRail network (NLR, 2007). Centralized administration of such resources in VOs makes it possible for it to be identified by static IP addresses or through Domain Name Service (DNS) servers.	Enterprise-wide desktop grid computing normally takes place within the confines of the corporate Intranet. The firewall prevents unauthorized access to the grid from external sources.	P2P and PRC resources are connected over the Internet. Increasingly, P2P systems are designed to work independently from DNS and offer significant or total autonomy from central servers. For example, the first-generation centralized P2P systems, like Napster have evolved to second-generation flooding-based P2P systems like Gnutella file sharing. PRC projects usually depend on central servers.
Trust	Resources and users are trusted. This is made possible through Certification Authorities (CA) that issue digital certificates for both resources and their users.	Since access to the grid is usually provided only to the organization's employees, and since unauthorized remote access is prevented through corporate firewalls, both users and resources are trusted.	Makes no assumptions on trust. Thus, files shared (in the case of P2P computing) and results returned (in the case of PRC) have to be verified.
Quality of Service (QoS)	Designed to deliver non-trivial QoS. Well-defined policies for resource sharing accounts for higher QoS.	The desktop PCs in an enterprise grid are generally non-dedicated resources and are geared-up for High Throughput Computing (HTC), i.e., it focuses on delivering large amounts of processing capacity over long periods of time.	Less concerned with QoS as P2P and PRC networks are characterized by few providers and many consumers. P2P and PRC users normally have to be provided with incentives to encourage sharing.
Services	Offers many services, e.g.,	Because enterprise grids are	Offers only limited services like

Comparison based on:	Cluster-based Grid Computing	Enterprise-wide Desktop Grid Computing	P2P / Public Resource Computing
	authentication and authorization, resource discovery, job scheduling, information services. These services can be used by a host of grid applications.	generally secure, the applications running on them can be provided with access to databases, files in shared directories, third-party applications like CSPs, etc.	access to disk space (Gnutella) and compute cycles (SETI@Home). However, these are NOT a generic set of services that can be used by P2P or PRC applications (see protocols below).
Protocols	OGSA is an effort towards the standardization of grid protocols and interfaces. This enables interoperability between different grid middleware. Furthermore, applications can use these standard protocols and interfaces for service discovery, data query, remote execution and monitoring, etc.	The middleware installed on enterprise-wide desktop grids usually have their own protocols, e.g., Condor.	Protocols in P2P and PRC are generally application specific. Thus, if a user runs multiple P2P / PRC applications, each application will run its own protocols over its own overlay network.

The discussion in this section has compared the different forms of grid computing. From Ian Foster's three-point definition of grid (Foster and Kesselman, 2004) - viz., non-centralized resource sharing, use of standard and general purpose protocols and interfaces, and delivery of non-trivial QoS - only the characteristics of cluster-based grid computing fits the definition of grids. However, enterprise-wide desktop grids (like Condor, Entropia DCGrid, United Devices GridMP) and P2P systems (like Gnutella) that do not implement standard grid interfaces and protocols can still be considered as *first-generation grids* because they integrate distributed resources in the absence of centralized control and offer "interesting qualities of services" (Foster, 2002). This research does not distinguish between different generations of grids, and uses the term "grid computing" to refer to cluster-based grid computing, EDGC, PRC and P2P computing, unless explicitly stated.

This section of the thesis has conducted a literature review on grid computing. For subsequent simulation-specific discussions on grids, the following three observations that were made in the course of this literature review are important:

- Grid computing allows users to access higher-level grids services like parallel computation service, task farming service, workflow service, etc.
- Cluster-based grid computing middleware like GT-4, VDT, gLite, etc. are primarily targeted at Unix and Linux flavour operating systems
- Middleware for EDGC, PRC and P2P forms of grid computing are available for Windows operating system.

The next two sections of the thesis present a brief overview of computer simulation (section 2.3) and the COTS simulation packages that are commonly used to model simulations in industry (section 2.4).

2.3 Computer simulation

A computer simulation uses the power of computers to conduct experiments with models that represent systems of interest (Pidd, 2004a). Experimenting with the computer model enables us to know more about the system under scrutiny and to evaluate various strategies for the operation of the system (Shannon, 1998). Computer simulations are generally used because they are cheaper than building (and discarding) real systems; they assist in the identification of problems in the underlying system and allow testing of different scenarios in an attempt to resolve them; allow faster than real-time experimentation; provide a means to depict the behaviour of systems under development; involve lower costs compared to experimenting with real systems; facilitate the replication of experiments; and provide a safe environment for studying dangerous situations like combat scenarios, natural disasters and evacuation strategies (Brooks et al., 2001; Pidd, 2004a).

Computer simulation can be applied in a wide range of application domains for a variety of purposes. A few of these are discussed here. In *manufacturing* computer simulation can be used to increase productivity by achieving a better operating balance among resources (Zimmers and Brinker, 1978). Simulation can be used for assessing the performance of asset and liability portfolios in the *finance and insurance* sectors (Herzog and Lord, 2003). In the *military* it can be applied to support training, analysis, acquisition, mission rehearsal and for testing and evaluation (Page and Smith, 1998). In *healthcare*, simulation can be used to model the highly uncertain nature of illness (e.g., bird flu epidemics) and to represent the complexity of subsystem interactions (e.g., interaction of blood supply chains with hospitals) (Lowery, 1998). It can be used for the study of *human-centred systems* through integration of human performance models with system performance models (Laughery, 1998). It can be applied to *Business Process Re-engineering (BPR)* as simulation can model the interaction between the various business process elements and can provide quantitative estimates of the impact that a process redesign is likely to have on key performance measures (Bhaskar et al., 1994).

This thesis investigates the application of grid computing to support the practice of DES and MCS in industry, particularly in manufacturing (DES), healthcare (DES) and finance (MCS) application areas. In a DES the behaviour of a model, and hence the system state, changes at an instant of time (Brooks et al., 2001). Two approaches that can be used to control the flow of time in a DES are the *Time Slicing* approach, where time is moved forward in equal time intervals, and the *Next-Event* approach, where time is moved at variable time increments from event to event, i.e., from one state change to the next state change (Pidd, 2004a). MCS, on the other hand, is a simulation procedure that uses a sequence of random numbers according to probabilities assumed to be associated with a source of uncertainty, for example, stock prices, interest rates, exchange rates or commodity prices (Chance, 2004). Commercial software packages are widely used in industry to facilitate DES and MCS (Tewoldeberhan et al., 2002; Swain, 2003), and are discussed in the next section.

2.4 COTS Simulation Packages (CSPs)

In the context of simulation practice in industry, although programming languages may be used to build simulations in certain circumstances, models are generally created using commercially available simulation packages (Robinson, 2005b). Visual Interactive Modelling Systems (VIMS) usually refer to DES software that enable users to create models in a graphical environment through an interactive “click-and-drag” selection of pre-defined simulation objects (entry points, queues, workstations, resources, etc.) and linking them together to represent the underlying logical interactions between the entities they represent (Pidd, 2004a). Examples of VIMS include commercially available DES packages like Witness (Lanner group), Simul8 (Simul8 corporation), AnyLogic (XJ technologies) and Arena (Rockwell automation). Similarly, MCS may be modelled in a visual environment using spreadsheet software like Excel (Microsoft), Lotus 1-2-3 (IBM, formerly Lotus Software); spreadsheet add-ins, for example @Risk (Palisade Corporation), Crystal Ball (Decisioneering); or through MC-specific simulation packages such as Analytica (Lumina Decision Systems) and Analytics (SunGard).

In this thesis the term *COTS Simulation Package (CSP)* is used to represent commercially available software for both DES and MCS. Thus, spreadsheets and spreadsheet add-ins for MCS are also regarded as CSPs. The term *DES CSP* or *MCS CSP* is used in cases where CSP specific to DES or MCS need to be distinguished.

Swain (2005) has made a comprehensive survey of commercially available simulation tools based on the information provided by vendors in response to a questionnaire requesting product information. It is the seventh biennial survey of simulation software for DES and related products (MCS software, distribution fitting software, etc.) and is published by the *Institute for Operations Research and Management Science (INFORMS)*. This list presently consists of 56 CSPs and features the most well known CSP vendors and their products (Swain, 2007). Table 5 below lists the CSPs by their type (i.e., MCS CSP or DES CSP) and the platform they are supported on (i.e., Windows, UNIX, Linux or Apple Macintosh). The tools that are neither MCS CSP nor DES CSP are highlighted in the table with a gray background. The information on supporting platforms has been taken from the OR/MS survey itself and the CSP type classification information was gathered from the CSP vendor website. Classification based on both CSP-type and platform-type is important for subsequent discussions.

Table 5: Survey of CSPs (extended from Swain’s OR/MS survey of simulation tools)

No	Software	Vendor	Windows	UNIX	Linux	Mac	CSP Type (MCS/DES)
1	@RISK	Palisade Corporation	1				MCS CSP
2	AgenaRisk	Agena	1	1	1		MCS CSP
3	Analytica	Lumina Decision Systems, Inc	1			1	MCS CSP

No	Software	Vendor	Windows	UNIX	Linux	Mac	CSP Type (MCS/DES)
4	AnyLogic 6.0	XJ Technologies	1	1	1	1	DES CSP
5	Arena	Rockwell Automation	1				DES CSP
6	AutoMod	BrooksSoftware	1				DES CSP
7	AutoSched AP	BrooksSoftware	1				DES CSP
8	Crystal Ball Professional	Decisioneering	1				MCS CSP
9	Crystal Ball Standard	Decisioneering	1				MCS CSP
10	CSIM 19	Mesquite Software	1	1	1	1	DES CSP
11	DecisionPro	Vanguard Software Corporation	1				MCS CSP
12	DecisionScript	Vanguard Software Corporation	1				MCS CSP
13	eM-Plant	UGS	1				DES CSP
14	Enterprise Dynamics Simulation Software	Production Modelling Corporation (PMC)					Not a MCS / DES CSP (PMC appear to be simulation consultants, not CSP vendors)
15	Enterprise Dynamics Studio	Incontrol Enterprise Dynamics	1				DES CSP
16	ExpertFit	Averill M. Law					Not a MCS / DES CSP (distribution fitting software)
17	Extend Industry	Imagine That, Inc.	1				DES CSP
18	Extend OR	Imagine That, Inc.	1			1	DES CSP
19	Extend Suite	Imagine That, Inc.	1				DES CSP
20	Flexsim	Flexsim Software Products, Inc.	1				DES CSP
21	ForeTell-DSS	DecisionPath, Inc.					Not a MCS / DES CSP (system dynamics software)
22	GAUSS matrix programming language	Aptech Systems, Inc.					Not a MCS / DES CSP (It is a programming language that can be used for simulation)
23	GoldSim Monte Carlo	GoldSim Technology Group	1				MCS CSP
24	Lean MAST	CMS Research Inc	1				DES CSP
25	Lean-Modeler	Visual8	1				DES CSP
26	MAST	CMS Research Inc	1				DES CSP
27	Micro Saint Sharp Version	Micro Analysis & Design	1				DES CSP
28	mystrategy	Global Strategy Dynamics Ltd	1				MCS CSP (for strategy planning)
29	Portfolio Simulator	ProModel Corporation	1				DES CSP
30	Process Simulator	ProModel Corporation	1				DES CSP (plug-in to Microsoft Visio)
31	ProcessModel Version 5.1	ProcessModel, Inc.	1				DES CSP
32	Project Simulator	ProModel Corporation	1				DES CSP (add-in to Microsoft Project)

No	Software	Vendor	Windows	UNIX	Linux	Mac	CSP Type (MCS/DES)
33	ProModel Optimization Suite	ProModel Corporation					Not a MCS / DES CSP (<i>OptQuest and SimRunner are optimization software add-ons to other ProModel products</i>)
34	PSM++ Simulation System (<i>old version: PASION</i>)	Stanislaw Raczynski					Not a MCS / DES CSP (<i>It is a simulation language</i>)
35	Quantitative Methods Software (QMS)	QuantMethods	1	1	1	1	MCS CSP
36	SAIL	CMS Research Inc	1				DES CSP
37	SAS/OR	SAS Institute Inc.	1	1	1		DES CSP
38	SCIMOD, Techno Corr, Techno Pas, Profimax, etc.	Techno Software International (TSI)					Not a MCS / DES CSP (<i>PMC appear to be simulation consultants, not CSP vendors</i>)
39	ServiceModel Optimization Suite	ProModel Corporation	1				DES CSP
40	ShowFlow 2	Webb Systems Limited	1				DES CSP
41	SIGMA	Custom Simulations					Not a MCS / DES CSP (<i>It is a simulation language</i>)
42	Simcad Pro	CreateASoft, Inc.	1				DES CSP
43	SIMPROCESS	CACI Products Company	1	1	1		DES CSP
44	SIMSCRIPT II.5	CACI Products Company					Not a MCS / DES CSP (<i>It is a simulation language</i>)
45	SIMUL8 Professional	SIMUL8 Corporation	1				DES CSP
46	SIMUL8 Standard	SIMUL8 Corporation	1				DES CSP
47	SLIM	MJC Limited	1	1			DES CSP (<i>for modelling supply chains</i>)
48	Stat::Fit	Geer Mountain Software Corp.					Not a MCS / DES CSP (<i>distribution fitting software</i>)
49	Supply Chain Builder	Simulation Dynamics, Inc.	1				DES CSP (<i>for modelling supply chains</i>)
50	Supply Chain Guru	LLamasoft	1				DES CSP (<i>for modelling supply chains</i>)
51	Systemflow 3D Animator	Systemflow Simulations, Inc.					Not a MCS / DES CSP (<i>It is a 3D simulation animator</i>)
52	TreeAge Pro Suite	TreeAge Software, Inc.	1		1	1	MCS CSP
53	Visual Simulation Environment (VSE)	Orca Computer, Inc.	1				DES CSP
54	WebGPSS (micro-GPSS)	AcobiaFlux AB	1				DES CSP

No	Software	Vendor	Windows	UNIX	Linux	Mac	CSP Type (MCS/DES)
55	WITNESS 2006	Lanner Group	1				DES CSP
56	XLSim	AnalyCorp Inc.	1				MCS CSP
	TOTAL MCS/DES CSPs		45	7	7	6	
	Supporting Platform (%)		100.00	15.56	15.56	13.33	

Information presented in Table 5 show that of the 56 simulation software and related products that have been surveyed, 12 are MCS CSPs, 33 are DES CSPs, 4 are simulation / programming languages (GAUSS matrix programming language, PSM++, SIGMA, SIMSCRIPT II.5), 2 are distribution fitting software (ExpertFit, Stat::Fit), 1 is an optimization suite (ProModel Optimization Suite), 2 appear to be simulation consultants (Production Modelling Corporation, Techno Software International), 1 is a systems dynamic software (ForeTell-DSS), and finally, 1 is a 3-D simulation visualization software (Systemflow 3D Animator). Some CSPs support multiple simulation approaches. For example, AnyLogic, Flexsim, Extend Industry and Extend OR support both DES and continuous simulation, and AnyLogic further provides system dynamics and agent-based simulation capabilities. However, for the purpose of this research, classification only on the basis of MCS and DES CSP is considered.

Spreadsheet applications like Microsoft Excel and IBM Lotus1-2-3 have not been included in Swain's survey, but will nonetheless be considered as MCS CSPs since they can be used to model MCS. SunGard Analytics software is used in banking and finance for Monte Carlo-based credit risk simulations, and this too will be considered as MCS CSP. These products have been specifically mentioned because some of the case studies that are discussed later in the thesis have used MCS applications built using Excel and Analytics.

As stated earlier, the CSP-type categorization has been completed by the author based on an extensive review of product information that is published on the vendor websites. As such, there may be errors in the classification due to incomplete (or exaggerated) product descriptions made available by the vendors or due to an inadvertent error on the part of the author. But in the most part this classification is considered valid by the author.

Of the total 45 CSPs (12 MCS CSPs and 33 DES CSPs) that have been identified from Swain's survey, all the CSPs are supported in the Windows platform, 15.56% (approx.) are supported in UNIX and Linux platforms, and only 13.33% (approx.) are supported under the Apple Macintosh Operating System. Furthermore, Excel and Analytics are supported only on the Windows platform. As will be discussed later in this thesis, platform support for CSPs is important when considering different grid technologies that can be potentially used with existing CSPs. Swain's survey has been widely cited in simulation literature (e.g., Pidd, 2004a; Boer, 2005; Ryde, 2005; Pidd and Carvalho, 2006), and in this research it is used to

investigate the extent of CSP support, through custom vendor implementations, for some identified uses of grid technology in the context of CSP-based simulation.

The next section investigates the higher-level grid services, described earlier in section 2.2.2, in the context of CSP-based simulation. The purpose here is to identify the higher-level grid services which could be potentially used together with CSPs.

2.5 Higher-level grid services for CSP-based simulation

Before continuing further it is worth considering if there is an end-user demand for grid technology for CSP-based simulation or, as has been pointed out earlier - “grid is a solution in search for a problem” (Schopf and Nitzberg, 2002), it is being investigated in this research to explore technology-driven possibilities. It is arguable that the suggestion of using multiple networked computers to execute simulations faster is appealing to practitioners, although they may not be aware of the term “grid computing”. This argument is further strengthened by the observations made by the author during his interactions with simulation end-users. Thus, in the case of distributed experimentation at least, there appears to be some user demand for distributed systems that can support execution of CSP-based simulations on multiple computers. However, as has been discussed in section 2.2.2, the potential of executing experiments in parallel over a network of computers (task farming service) is but one of multiple higher-level services that can be provided through use of grids. The majority of simulation users may be unaware of these grid-facilitate services, and from this perspective grid computing can be seen as providing a technology-driven impetus to facilitate its possible adoption for CSP-based simulation in industry.

Robinson (2005b) has distinguished between demand-led and technology-led innovation in simulation practice. This distinction is rephrased to show its relevance to CSP-based simulation modelling. A demand-led innovation occurs when the functionality provided by the CSPs lag behind the requirements of the simulation practitioners, and the implementation of which would aid current simulation practice. On the other hand, a technology-led innovation occurs when research and development move ahead of the requirements of the CSP users, and which has the potential to change and improve the current simulation practice. In this thesis, the demand-led and technology-led innovations are considered in the context of using multiple networked computers to support CSP users in industry.

The wide prevalence of CSPs suggests that (1) simulation practitioners using these packages are constrained by the functionality provided by the package vendors, and (2) vendors will generally have to become involved in further development of their packages to provide any new features. CSP vendors usually have limited manpower and budget at their disposal, and they might not consider incorporating support for a particular feature unless there is sufficient demand for it (Ryde 2005). Thus, it is more likely that vendors will, first and foremost, be

interested in supporting demand-led requirements before considering technology-led feature support.

This thesis does not differentiate between demand-led and technology-led impetus to further the practice of simulation in industry. It is considered that, from the perspective of the simulation practitioner, what is important is getting all the CSP support required to complete the task at hand. What is seen as a demand-led innovation by one user may be regarded as a technology-led intervention by the other, and vice-versa. For example, in the context of simulation practice in industry, distributed multiple replication (distributed experimentation) and distributed model execution (distributed simulation) have been shown to be demand-led and technology-led interventions respectively (Robinson, 2005b). Although it is generally considered to be true that the demand for distributed experimentation is greater than the need for distributed simulation (and the author agrees with this), from the perspective of the simulation modeller such characterization may not be necessary. A user who generally develops small models that require only a few minutes to execute on a PC may feel that distributed experimentation is not necessary. Thus, he may consider it as a technology-led innovation. On the other hand, a user who is involved in creating large and complex models that require hours to execute on a single PC might be interested in distributing the execution of the model onto multiple computers to decrease runtime, and may also see the benefits of distributed experimentation. In this case the requirement for distributed simulation and distributed experimentation support in CSPs can be seen as demand-led. In this thesis, both the demand-led and technology-led innovations, in the context of using multiple computers for simulation, are seen as potential areas for application of grid computing technologies.

The next four sections of this thesis discuss four higher-level grid services that can be potentially used together with CSPs. The four services are parallel computation service (section 2.5.1), task farming service (section 2.5.2), workflow service (section 2.5.3) and collaboration service (section 2.5.4). The grid portal service is discussed in section 2.7 in the context of web-based simulation. Grid-facilitated integration service is not investigated because CSPs seldom need integration with physical systems, heterogeneous distributed databases, etc. Similarly, computational steering service is not considered appropriate for further investigation because the user will generally need to access the remotely running graphical CSP interface to computationally steer the simulation, and grid middleware do not generally support such remote visualization of user applications that are being executed over various grid nodes. However, a groupware like Microsoft NetMeeting can be used to provide such access (Taylor, 2000). A discussion on the higher-level grid services can be found in section 2.2.2.

2.5.1 Parallel computation service

The Journal of Simulation's (JoS) survey on the future for DES takes note of the present and the expected future trends for creating increasingly large models (Taylor and Robinson,

2006). CSPs, although suitable for most simulations that are modelled in industry, may however be unable to simulate such large and complex models (Pidd, 2004b). Arguably, one reason for this is, the larger the model, the greater the processing power and memory required to simulate the model. Simulation is a computationally intensive technology that has benefitted from increasing processor speeds made possible through advances in computer science; and with ever increasing processing speeds, the CSPs, in future, will possibly provide features that may not presently seem possible (for example, dramatic decrease in model runtime, execution of increasingly large and complex models, etc.) (Hollocks, 2006). However, it is also true that with more processing power available the simulation user may tend to develop even larger and more complicated models simply because it is possible to do so (Robinson, 2005a). This, in turn, may again mean that CSPs will not be able to support execution of some user models because of their sheer size and complexity. Thus, in some cases at least, there may be a need for more computation power to support the practice of simulation in industry. One way to facilitate the execution of a large model using existing computing resources is through development of CSPs that support parallel computing (i.e., utilizing multiple processors to speed up the execution of one simulation). The grid-facilitated higher-level parallel computation service can then be used to execute such CSPs.

Parallel computing is the concurrent use of multiple processors to solve a computational problem and generally involves the following three steps (Barney, 2006).

- Breaking down a problem into sub-parts that can be solved concurrently.
- Breaking down the sub-parts into a series of CPU instructions.
- Executing the instructions from each sub-part concurrently over different CPU's.

The three steps that are described above are graphically illustrated in figure 9 below.

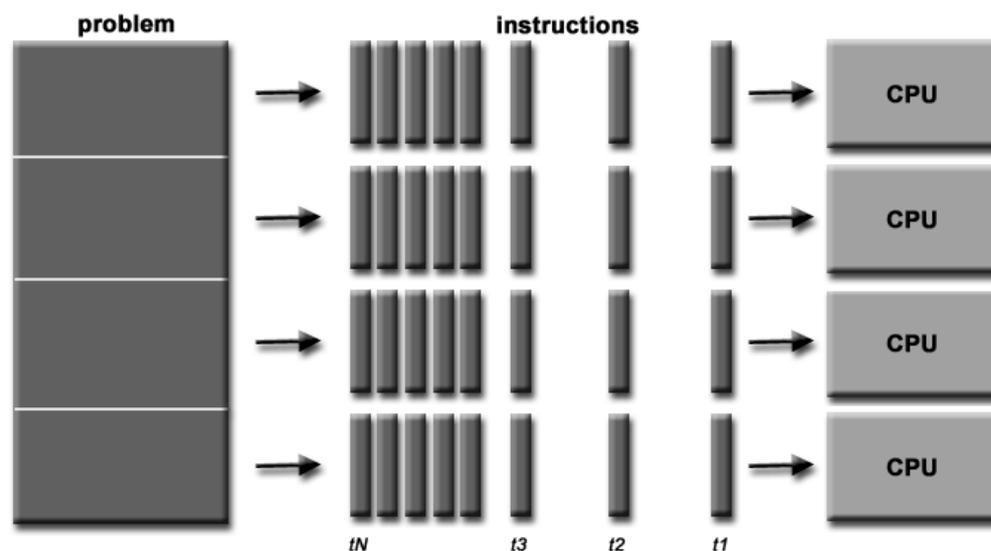


Figure 9: Parallel computing using multiple CPUs (Barney, 2006)

The set of processors that take part in such parallel computation can be referred to as a parallel computer, and may include parallel supercomputers that have thousands of

processors, networks of commodity PCs, shared-memory multiprocessors, distributed-memory multiprocessors and embedded systems (Foster, 1995). Parallel computing is generally used to speed up the execution of a computation, to solve problems that might be impossible with traditional sequential computers, to provide concurrency, to take advantage of non-local resources, among others (Barney, 2006).

In a parallel computation the processes (instructions) being executed on separate processors may need to communicate with each other. Two dominant forms of such inter-process communication are shared variables and message passing (Fujimoto, 1999b). A shared-memory multiprocessor system can provide executing processes shared access to variables using shared memory that is present in such systems. In the case of distributed-memory multiprocessor systems and networks of PCs, where there is no access to shared memory, the communication between the executing processes is usually accomplished by sending messages that are based on message passing standards like MPI. PVM also has an explicit message-passing model for such inter-process communication. The typical architecture of a shared-memory and a distributed-memory multiprocessor system is shown in figure 10 (adapted from Barney, 2006). The architecture of a parallel computer comprising of a network of PCs is similar to the distributed-memory multiprocessor, but with one key difference. The technology used for interconnecting the different workstation nodes is based on standard networking technology like Ethernet, and not on customized high speed interconnection switches as it the case with distributed-memory multiprocessor computers. The discussion on parallel computing in this thesis is in the context of using network of PCs for parallel computation.

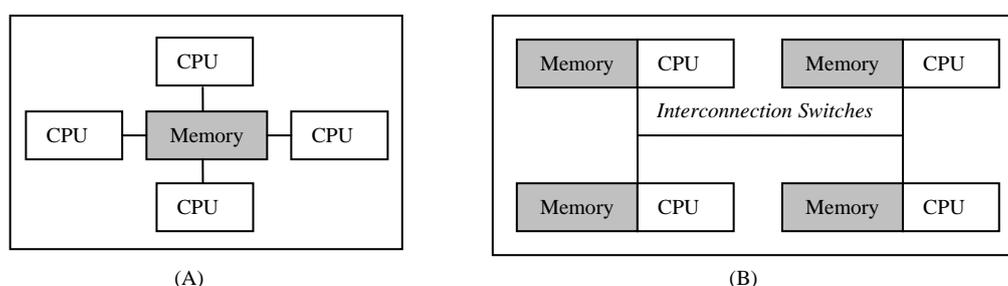


Figure 10: Shared-memory (A) and distributed-memory (B) multiprocessor machines

As has been stated earlier, a network of PCs built using commodity hardware, software and networking technologies can run parallel programs using message passing mechanisms like the MPI and PVM. A MPI program generally creates a fixed set of processes, one for each processor, which may execute different programs (multiple instructions) or the same program (single instruction) and communicate with other processes by calling library routines to send and receive messages (Foster, 1995). The processes generally have access to different sets of data (multiple data). Similarly in PVM, a collection of tasks (the unit of parallelism in PVM is called a task) communicate with each other by sending messages and cooperatively solve a

computation through data decomposition (multiple data), functional decomposition (multiple instructions) or a combination of both (Geist et al., 1994). Thus, MPI and PVM both support *Single Program Multiple Data (SPMD)* and *Multiple Program Multiple Data (MPMD)* parallel programming models.

For a MCS or DES CSP to support parallel computation over a network of PCs with distributed memory, the underlying simulation package will generally have to support message passing mechanisms like those discussed above. This requires intervention from the CSP vendor and may involve a total redesigning and implementation of the software. A MCS CSP may implement a SPMD parallel processing model where each processor executes the same model but with different random number streams. On the other hand, a MPMD parallel processing model may have to be implemented by a DES CSP where different sub-parts of the model are executed over different processors (this may require Parallel Discrete Event Simulation (PDES) algorithms – discussed in section 2.6.2) or different sub-components of the CSP (e.g., simulation executive, visualization sub-component, statistics collection, interpreter for user-defined code, etc.) access multiple distributed processors through a processor abstraction layer (shown in figure 11 below).

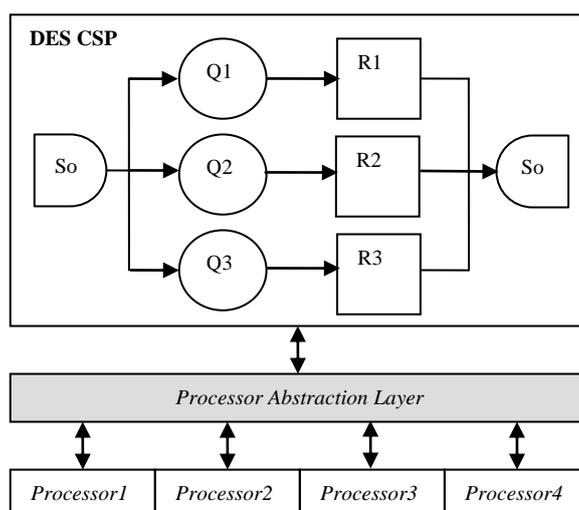


Figure 11: Parallel computing using a DES CSP

The CSPs that utilize multiple CPUs for simulation are listed in table 6. The table shows that only two MCS CSPs enable parallel simulation execution over multiple-processor machines. DES CSP Simul8 is not considered to provide parallel computation support for the reasons noted in table 6 (next page). Furthermore, none of the 45 CSPs surveyed support parallel computation over distributed processors. One reason for this may be that it is arguable as to what extent a general purpose simulation application like the CSPs can provide parallel simulation support. The overwhelming experience in parallel software development is that high application performance requires load-balancing, communication, and synchronization techniques that are often application specific (Nicol and Heidelberg, 1996). Nevertheless, with improved technology and programming models it may be possible to parallelize CSP

applications to utilize processors distributed over the network efficiently, and as such parallel computation service is considered as a potential higher-level grid service that could benefit CSP-based simulation modelling.

Table 6: CSPs that support parallel computation

Software	Vendor	Features	Information Source (Appendix A.1)
@Risk Industrial	Palisade Corporation	The @RISKAccelerator integrated in @Risk Industrial speeds-up MCS by using all CPUs in a single, multiple-CPU machine.	Vendor website
Simul8 Professional and Standard Editions	SIMUL8 Corporation	Simul8 can use up to four processors in a multi-CPU machine to conduct trials. AUTHOR'S COMMENT: <i>This cannot be considered as parallel computation because each trial runs a separate Simul8 process. Furthermore, running one trial using 4 CPUs will generally not give any performance benefit compared to, say, running one trial on a single CPU machine (with identical processor, RAM, etc. specifications).</i>	Discussion with vendor & Simul8 newsletter
TreeAge Pro	TreeAge Software, Inc.	To support complex and lengthy MCS, TreeAge Pro can use up to eight processors on a single computer.	Vendor website

There are some examples in literature where parallel simulators and optimizers have been implemented for solving specific problems. For example, Mccoy and Deng (1999) have implemented a high-performance, parallel, molecular-dynamics software package that includes features like asynchronous message passing, dynamic load balancing, mechanisms for data caching, etc. Mutalik et al. (1992) have implemented a parallel simulated annealing algorithm and a parallel genetic algorithm for solving combinatorial optimization problems on shared memory multiprocessor systems and distributed memory systems. Their approach uses message passing for communication between processes running on multiple CPUs is through message passing. Yau (1999) describe the AKAROA package for parallel steady-state stochastic simulation of high-speed and integrated-services communication networks. The package can be used on multiprocessor systems and heterogeneous computer networks. Elmroth et al. (1999) have implemented a parallel version of TOUGH2 (Transport Of Unsaturated Groundwater and Heat version 2) simulation package, a widely used software for studying ground water flow related problems such as nuclear waste isolation, geothermal reservoir engineering, etc., to solve a set of coupled mass and energy balance equations using a finite volume method in parallel.

2.5.2 Task farming service

The practice of simulation can gain from an increased availability of computation power (Robinson, 2005a). Grid facilitated task farming service uses multiple grid resources to execute simulation experiments in parallel. Task farming involves distributing (farming) the simulation experiments (tasks) over different PCs that are part of the grid infrastructure and using the computational service and application service (basic grid services – section 2.2.1) to execute the simulation over the grid nodes. The CSPs that support some form of task farming are presented in table 7.

Table 7: CSPs that provide support for task farming

Software	Vendor	Features	Information Source (Appendix A.2)
GoldSim Monte Carlo	GoldSim Technology Group	The GoldSim <i>Distributed Processing Module</i> is an add-on module that allows users to combine the power of multiple computers connected over a network to carry out MCS.	Vendor website
SIMPROCESS	CACI Products Company	The <i>Remote Plot Capability</i> of SIMPROCESS allows the user to set up multiple computers to present the plots while the simulation is running on another computer. NOTE: <i>SIMPROCESS does not distribute the simulation workload but only the visualization aspects and, as such, offers only limited task farming features.</i>	Vendor website
Simul8 Professional and Standard Editions	SIMUL8 Corporation	Simul8's parallel processing feature allows the user to spread the execution of trials across two or more networked computers that have Simul8 installed. The PCs only use spare CPU cycles to execute the models. A network drive that can be accessed by all the PCs should be made available for parallel processing to work. NOTE: <i>Simul8 does not presently provide task farming support for multiple models (referred to as Multiple Model Multiple Data (MMMD) task farming - section 3.3.2).</i>	Vendor website (Simul8 newsletter)
Vanguard Studio (DecisionPro)	Vanguard Software Corporation	Vanguard's <i>Grid Computing Add-in</i> give users the ability to run large MCS by dividing the simulation task between many computers on the Enterprise Grid.	Vendor website

Of the 45 CSPs that have been surveyed only 2 MCS CSPs and 1 DES CSP support task farming. DES CSP SIMPROCESS only provides limited task farming features for reasons noted in the above table. There are some examples in literature that have used task farming architecture, consisting of one master computer and multiple worker computers, to execute simulation experiments faster. For example, Marr et al. (2000) have used the *SimManager* (master process) to execute parallel simulation studies over multiple *Engines* (worker processes) using the Java-based Silk simulation system (Kilgore, 2000). Yücesan et al. (1998) describe a project that aims to distribute DES experiments over the Internet with a view on simulation optimization. The system they implement is called the PDESSS (Parallel Discrete-Event Simulation Support System). Mustafee and Taylor (2006) have implemented a task farming system that support concurrent execution of multiple instances of different Simul8 models (MMMD task farming - section 3.3.2).

Robinson (2005b) has discussed some of the potential applications of simulation in a networked environment (referred to as distributed simulation in his paper) under four specific categories, viz., model execution, data management, experimentation and project processes. This is presented in table 8 below.

Table 8: Potential applications of simulation in a networked environment (Robinson, 2005b)

Category	Potential Application	Description
Model execution	Distributing model execution	Splitting the execution of a large model across a series of computers
	Linking separate models	Running separate models concurrently across a series of computers
Data management	Linking to database and other software	Linking models to remote databases and other software
	Linking to real-time systems	Linking models to remote real-time systems

Experimentation	Gaming	Distributed users interacting with a simulation.
	Distributed multiple replications	Distributing replications across a series of computers to speed up execution
	Distributed multiple scenarios	Distributing experimental scenarios across a series of computers to speed-up experimentation
Project processes	Sharing models	Giving distributed users access to the same simulation model
	Application sharing	Giving distributed users access to the same simulation software
	Virtual meetings	Remote meetings between modellers and users during model development
	Searching for model components	Searching for and downloading components for model building

As can be seen from the table above, two potential applications of simulation under the “experimentation category” are distributed multiple replications and distributed multiple scenarios. In this thesis distributed multiple replication and distributed multiple scenarios are referred to as Single Model Multiple Data (SMMD) task farming and Multiple Model Multiple Data (MMMD) task farming respectively (discussed in section 3.3.2). SMMD task farming refers to the execution of one model using different experiment parameters over multiple processors. MMMD task farming, on the other hand, refers to multiple SMMD experiments being executed concurrently over the grid. Simulation, being a computationally intensive OR technique that usually requires multiple experimentation runs with varying parameters, can potentially gain from the use of additional computing resources being made available through the task farming service. As such, task farming is considered as a potential higher-level grid service that could benefit the practise of CSP-based simulation in industry.

2.5.3 Workflow service

Grid-facilitated workflow service has the potential to link CSPs with other software applications through use of workflow management systems (WMSs). The reader is referred to section 2.2.2 for examples of WMSs. The WMS is usually responsible for executing different applications over the grid in a phased manner based on dependencies between executing programs. The dependency is generally in the form of data, wherein data output from one application serves as the input to a different application. The applications usually run on different grid nodes and the responsibility of transferring data between the nodes is generally with the WMS and the underlying grid middleware.

An example of a workflow using a CSP and an external application can be the visualization of a model by the latter from the simulation output of the former. For example, a visualization application like *Systemflow 3D Animator* can be used to animate the output of a simulation in 3-D graphics, provided a time stamped event log is generated by the DES CSP (Systemflow Simulations, 2006). This is shown in figure 12.



Figure 12: Workflow using a DES CSP and a visualization application

Looking at the above example in the context of grid-facilitated workflow service, it may be possible to use a WMS to specify the dependencies between the DES CSP and the Systemflow 3D Animator. This would enable the WMS to execute both the applications over (possibly) different grid resources in a phased manner (i.e., the execution of DES CSP and Systemflow 3D Animator is sequential). The WMS would also be responsible for transferring data output by DES CSP to the grid node running the Systemflow 3D Animator.

It can be argued that linking CSPs to data sources (databases, spreadsheets, etc.) also represent a form of workflow because CSPs and the data sources are different applications and the former may be dependent on data from the latter (to populate variable values, etc.). And as most CSPs provide means to access databases, spreadsheets and files (Robinson, 2005a), it can be assumed that workflow support is already present in most simulation packages (some of the MCS and DES CSPs that support data source access are presented in table 9 below). Here the communication is generally one way, i.e., the CSPs performs read and write operations on data sources.

Table 9: CSPs that support data source access

Software	Vendor	Features	Info. Source (Appendix A.3)
AnyLogic 6.0	XJ Technologies	AnyLogic models can dynamically read and write data to spreadsheets, databases, Enterprise Resource Planning (ERP) or Customer Relationship Management (CRM) systems.	Vendor website
Arena	Rockwell Automation	Provides ActiveX Data Object (ADO) and Open DataBase Connectivity (ODBC) access to Oracle, Access, Excel, etc.	Vendor website
Enterprise Dynamics Studio	Incontrol Enterprise Dynamics	Provides ODBC access to databases.	Vendor website
GoldSim Monte Carlo	GoldSim Technology Group	Enables exchange of data between any ODBC-compliant database and GoldSim model	Vendor website
Simprocess	CACI Products Company	SIMPROCESS has the capability to provide simulation models as callable services through the use web services. This capability provides simulation-on-demand to applications within the enterprise.	Vendor website
Vanguard Studio (DecisionPro)	Vanguard Software Corporation	Vanguard's <i>Web Services Add-ins</i> allows inter-connection between the Vanguard models and other enterprise systems. For example, by applying the Web Services Add-ins Vanguard models can be built that pull real-time data from an Enterprise Resource Planning (ERP) system through a web service, performs a Monte Carlo cash flow simulation and then push the results back into the same ERP system through another web service.	Vendor website

Software	Vendor	Features	Info. Source (Appendix A.3)
WITNESS 2006	Lanner Group	Witness can access spreadsheets and databases like Oracle, SQL Server, Access, etc.	Vendor website

Furthermore, it can be argued that CSP-based workflow implementation is not limited to linking CSPs with data sources alone, and it may be possible to write code that interfaces the CSPs with one or more applications (optimization software, data analysis software, visualization application, etc.). Interfacing simulation software with external programs generally requires application-level support to facilitate inter-process communication between the executing programs. COM (Component Object Model) is one such technology that allows different software programs to communicate with each other by means of interfaces (Gray et al., 1998). The MCS and DES CSPs that expose package functionality through Application Programming Interfaces (APIs), COM, Object Linking Embedding (OLE) (Gani and Picuri, 1995) and similar technologies are presented in table 10 below. Such access should ideally also be provided by the external applications (with which the CSPs are being linked) to facilitate two-way communication.

Table 10: CSPs that expose package functionality

Software	Vendor	Features	Info. Source (Appendix A.4)
AgenaRisk Enterprise Edition	AgenaRisk	<i>Agena API</i> provides Java routines that allow users to create, edit and execute AgenaRisk models.	Vendor website
Simprocess	CACI Products Company	Provides <i>external application call</i> and <i>remote application call</i> support that enables the user to write java modules to interface Simprocess with applications running on the same computer or other computers over the network respectively.	Vendor website
Simcad Pro	CreateASoft, Inc.	Includes a <i>Visual Basic scripting engine</i> that makes it possible for Simcad Pro to interfaces with custom and off-the-shelf applications.	Vendor website
Crystal Ball Professional and Premium Editions	Decisioneering	Crystal Ball Professional and Premium Editions include a <i>Developer Kit</i> that consists of macro command and method libraries that can be called from within a VBA program or from any other language outside of Excel that supports OLE 2 automation.	Vendor website
GoldSim	GoldSim Technology Group	A Dynamic Link Library (DLL) makes it possible to link an external computer program directly to GoldSim.	Vendor website
Extend Industry, Extend OR and Extend Suite	Imagine That, Inc.	Extend supports the component object model (COM/ActiveX) and makes it possible to control an application from within Extend, or have it control Extend.	Vendor website
Enterprise Dynamics Studio	Incontrol Enterprise Dynamics	Allows creation of simulation solutions that can act as stand-alone applications or solutions that are embedded with other systems.	Vendor website
Analytica	Lumina Decision Systems, Inc	Enterprise-level features including OLE linking.	Vendor website
Witness	Lanner	The <i>SIMBA SDK</i> (Software Developer's Kit) includes a COM enabled version of WITNESS that can be used by external applications. It also includes ActiveX libraries that enable WITNESS displays inside other products that support such objects (e.g. Microsoft Excel) or other programmed interfaces.	Vendor website

Software	Vendor	Features	Info. Source (Appendix A.4)
@Risk Professional	Palisade Corporation	The <i>Excel Developer Kit (XDK)</i> automates and customizes @RISK for Excel through a complete library of commands and functions. @RISK for Excel can be added to any custom application.	Vendor website
Enterprise Dynamics	Production Modelling Corporation	Enterprise Dynamics uses open architecture, supporting major industry standards and can be easily connected or integrated with other software systems and components.	Vendor website
ProModel	ProModel Corporation	Using Microsoft Visual Basic (or any other ActiveX-enabled language), ProModel can be executed from another application.	Vendor website
Arena	Rockwell Automation	Provides ActiveX controls, Visual Basic for Applications (VBA), ActiveX object model for external control.	Vendor website
Simul8 Standard and Professional Editions	Simul8 Corp	Provides a standard Windows COM interface that allows any application that can use COM to drive SIMUL8.	Vendor website
eM-Plant	UGS	eM-Plant has an open system architecture that supports multiple interfaces and integration capacities like ActiveX, Sockets, etc.	Vendor Website
AnyLogic	XJ Technologies	AnyLogic can interoperate with software written in Java or other languages (via Java Native Interface). External programs can be called from anywhere in the model, and vice versa. Simulation models can be called from external program using the open API of AnyLogic simulation engine.	Vendor website

Finally, it may be possible to link simulations to real-time systems (Robinson, 2005b). Linking simulations to physical real-time systems can facilitate symbiotic simulation. A symbiotic simulation system consists of a simulation model interacting with the physical system in a mutually beneficial way, with the former benefitting from continued access to the latest data and the automatic validation of the simulation outputs, and the latter benefitting from optimized performance obtained from the analysis of simulation experiments (Low et al., 2005). Communication between the CSP and the physical system may be achieved using open interfaces. The CSPs that expose package functionality have already been listed in table 10. The physical system (through associated software) should generally provide similar access to the CSPs to facilitate two-way communication.

For the purpose of this research, linking CSPs to data sources, applications and real-time systems is not considered a workflow because there is usually no overarching mechanism (like WMS in case of grids) that (1) controls phased execution of the different applications and (2) is responsible for transferring data between the applications. Furthermore, grid-facilitated workflow service is designed to work on distributed resources, and linking CSPs to data sources etc. may only work if the applications are installed on the same computer.

2.5.4 Collaboration service

Grid-facilitated collaboration service provides mechanisms that could potentially allow different users to mutually access each other's applications. In the context of CSP-based simulation, this service will be discussed in relation to (1) simulation model reuse through model sharing between different users, and (2) sharing CSPs and individual models for co-

operative model development. Another form of collaboration that has been identified in section 2.2.2 involves interactions among remote grid users through integrated support for virtual meetings. The importance of communication among the simulation modellers and the problem owners in conducting a successful simulation study cannot be overstated, and therefore virtual meeting support for CSP-based simulation will also be discussed.

Simulation model reuse refers to the creation of new models using pre-existing modelling artefacts like portions of simulation code, simulation components and complete models in itself, with the purpose of reducing the time and cost for model development (Robinson et al., 2004). Model reuse is a form of collaboration because models created by one modeller may be reused by others. An extension of model reusability is the concept of separate development and user groups, whereby models are developed and validated by one group and then used to specify simulations by another group (Bortscheller and Saulnier, 1992). Pidd (2002) distinguishes between four different types of model reuse that can be applied to simulation, viz., *code scavenging* (reusing existing code), *function reuse* (reusing functions that provide specific functionalities), *component reuse* (reusing encapsulated simulation modules that provide a well-defined interface for communication with other such modules) and *full model reuse* (reusing a pre-existing model). Figure 13 indicates the frequency of model reuse and the complexity that is associated with the four forms of model reuse.

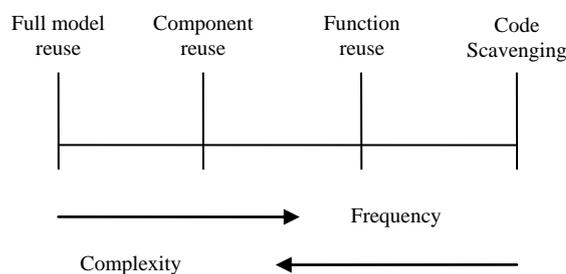


Figure 13: Frequency of model re-use and its underlying complexity (Pidd, 2002)

In the context of DES CSPs simulation models may be reused in the following ways (Paul and Taylor, 2002).

- Through reuse of basic modelling components like queues and workstations that are included in the DES CSPs. In Pidd's classification this can be referred to as fine-grained *component reuse*.
- Through reuse of subsystem models that may be available through a model library or that have been previously developed by the modeller. In Pidd's classification this can be referred to as coarse-grained *component reuse*.
- Through reuse of similar models that have been developed previously with appropriate changes. In Pidd's classification this is referred to as *full model reuse*.

In a networked environment simulation model reuse will generally involve searching and downloading model components (also existing models) for model building (Robinson, 2005b). For simulation practitioners to benefit from such an approach the search-and-download features should ideally be integrated with the CSPs. The search feature could potentially allow discovery of CSP model components through an inter-organizational repository of models, and the download feature could make it possible to load the model into a CSP, modify it according to the requirements of the new model and then execute it (Bell et al., 2006). Table 11 below lists the CSPs that allow creation of reusable modelling components.

Table 11: CSPs that support creation of reusable model components

Software	Vendor	Features	Information Source (Appendix A.5)
Crystal Ball Standard and Professional Editions	Decisioneering	Crystal Ball 7 supports collaboration by allowing multiple users to subscribe to distributions that have been created by other users.	Vendor website
Extend Industry, Extend OR and Extend Suite	Imagine That, Inc.	<i>Extend blocks</i> are components that are the building blocks for other models. It is possible to create, reuse and distribute these blocks through the Extend library.	Vendor website
Micro Saint Sharp Version 2.1	Micro Analysis & Design	Micro Saint allows creation of reusable modelling components.	Vendor website
Visual Simulation Environment (VSE)	Orca Computer, Inc.	Users can reuse model components from a library. They can create (and sell) their own library of reusable models and model components developed for a specific problem domain.	Vendor website
Arena	Rockwell Automation	With Arena Professional users can develop custom templates that consist of libraries of modelling objects.	Vendor website
eM-Plant	UGS	eM-Plant helps to create models using libraries of standard and specialized components. The users can also extend the library with their own objects.	Vendor website
Vanguard Studio (DecisionPro)	Vanguard Software Corporation	<i>Vanguard Library Server</i> makes models available to other model builders as components. Multiple components, each of which may be maintained independently, can be linked together for analysis. The Vanguard server manages all interaction between the components.	Vendor website
AnyLogic	XJ Technologies	The native Java environment of AnyLogic supports extensibility including custom java code, external libraries, and external data sources.	Vendor website

Of the eight MCS and DES CSPs that have been listed above only two appear to facilitate simulation reuse in a networked environment. MCS CSP Crystal Ball (Version 7.0) has a *distribution gallery* that allows multiple users to subscribe to distributions that have been created by others. Similarly MCS CSP DecisionPro, which is a sub-system of the web-based Vanguard Studio, makes model components available to users.

Sharing CSPs and individual models enable different users to access the same simulation software and/or the same simulation model for model building purposes. Obviously, this is a form of collaboration because multiple simulation users are involved. In this thesis sharing CSP applications is discussed in the context of web-based simulation in section 2.7. CSPs that enable joint simulation development through model sharing are listed in table 12.

Table 12: CSPs that facilitate model sharing

Software	Vendor	Features	Information Source (Appendix A.6)
AnyLogic	XJ Technologies	AnyLogic 6 allows the use of version control software (namely CVS) from the model development environment to facilitate multiple modellers to work on a large project. Functions to share, commit and update models are available from the project tree view provided by the CSP.	Vendor website

As can be seen from the table above, only one out of the 45 CSPs that have been surveyed appear to support model sharing for the purposes of cooperative model development. However, it may be also possible to facilitate joint development of models using other techniques like merging several model files together from various developers (Ryde, 2005). Packages like Simul8 and ProModel offer such capabilities.

Virtual meetings may encourage frequent interactions between the simulation modellers and problem stakeholders. Through a survey of simulation consultants and their respective clients, Robinson and Pidd (1988) have observed that three important factors related to the success of a simulation study were (1) regular communication between the clients and the consultants, (2) regular meetings between the clients and the consultants and (3) teamwork. All three factors are bound together by the common requirement of communication. In a distributed environment such communication can be achieved through virtual meetings. A CSP that supports this form of collaboration would generally require integrating audio, video and messaging capabilities along with the package. At present there are no CSPs that integrate virtual meeting capabilities along with their packages.

The discussions in this section have shown that CSP-based simulation modelling may gain from the use of grid-facilitated collaboration service, as this higher-level grid service can potentially provide mechanisms for reusing model components, can facilitate model sharing for joint development and provide support for virtual meetings. It is interesting to note that the three possible applications of grid-facilitated collaborative service in the context of CSP-based simulation (namely, model reuse, model sharing and virtual meeting) have been included in Robinson's classification of potential applications of simulation in a networked environment (Robinson, 2005b) (table 8).

Section 2.5 of this thesis has discussed four higher-level grid services in relation to CSPs. The four services were parallel computation service (section 2.5.1), task farming service (section 2.5.2), workflow service (section 2.5.3) and collaboration service (section 2.5.4). The next two sections of this thesis describe two specific forms of simulation, namely, distributed simulation (section 2.6) and web-based simulation (section 2.7). Both these forms of simulation involve the use of multiple computing resources, and as such it will be interesting to investigate them in the context of grid computing. Although there are no higher-level grid

services for distributed simulation and web-based simulation, it may be possible to define simulation-specific higher level grid services which would allow both these forms of simulation to be included in the grid computing framework for CSPs that will be proposed in this thesis.

Distributed simulation is discussed next. It will include an overview of distributed simulation and its application areas (section 2.6.1), distributed simulation theory and conservative synchronization algorithm (section 2.6.2), middleware for distributed simulation and HLA-based simulations using DES CSPs (section 2.6.3) and, finally, a discussion on grid-facilitated distributed simulation service (section 2.6.4).

2.6 Distributed simulation

Distributed Simulation generally refers to the execution of a DES comprising two or more individual models, each of which runs on a separate processor. These processors can be a part of a multiprocessor computer or may belong to multiple PCs that are connected over the network. Parallel Discrete Event Simulation (PDES) usually refers to the execution of such distributed DES on parallel and distributed machines (Page and Nance, 1994).

Some of the reasons for using distributed simulations are as follows (Fujimoto, 1999a; Fujimoto, 2003).

- Distributed simulation can facilitate model reuse by “hooking together” existing simulations into a single simulation environment. It is usually far more economical to link existing simulations to create distributed simulation environments than to create new models within the context of a single tool or piece of software.
- A large simulation may have memory and processing requirements that cannot be provided by a single system. Distributing the simulation execution across multiple machines may allow the memory and processors of many computer systems to be utilized. Thus, distributed simulation may enable large simulations to be executed that could not be executed on a single computer.
- Executing simulations on a set of geographically distributed computers facilitates wider user participation in the simulation experiments. This also alleviates the cost and time that is normally associated with bringing participants to one physical place for conducting a joint simulation exercise.

In the context of PDES, Fujimoto (2001) distinguishes between parallel and distributed simulation based on the frequency of interactions between processors during the simulation execution. A parallel simulation is defined as running a simulation on a tightly coupled computer with multiple central processing units (CPUs) where the communication between the CPUs can be very frequent (e.g., thousands of times per second). A distributed simulation, on the other hand, is defined as executing simulations on multiple processors over loosely coupled systems (e.g., a network of PCs) where the interactions take more time (e.g.,

milliseconds or more) and occur less often. Sometimes the terms parallel simulation and distributed simulation are used interchangeably (Reynolds, 1988). In one of his more recent papers, Fujimoto (2003) uses the term distributed simulation to refer to both the parallel and distributed variants of PDES. The rationale presented is that, although historically, the terms “distributed simulation” and “parallel simulation” referred to geographically distributed simulations and simulations on tightly coupled parallel computers respectively, new distributed computing paradigms like clusters of workstations and grid computing has made this distinction less obvious. This research takes a similar view and therefore does not distinguish between the parallel and distributed variants of PDES. The terms distributed simulation and PDES will henceforth be used to refer to the execution of distributed simulation on both multiprocessor machines and over network of PCs.

2.6.1 Application areas of PDES

Some of the current and potential application areas for PDES are presented in table 13 below (Fujimoto, 1999b).

Table 13: Application areas of parallel and distributed simulation

Applications	Type of simulation
Military applications	Analytical war game simulations are performed to evaluate different strategies for war. These simulations are typically composed of individual models that represent different military divisions and use PDES algorithms (discussed in section 2.6.2) for synchronization of the models. Another application of PDES in the military is for training, and test and evaluation (T&E). These are conducted in distributed virtual environments (DVE) where both humans (<i>human-in-the-loop</i>) and devices (<i>hardware-in-the-loop</i>) take part in the simulation. Note: Unlike traditional distributed analytic simulations, DVE simulations are executed as per wall clock time. Furthermore, they usually incorporate rich 3-D graphics that gives users the look and feel of being embedded in the system being modelled.
Telecommunication networks	Analytical PDES have been used widely to evaluate networking hardware, software, protocols and services in the telecommunication industry.
Social interactions and business collaborations	Distributed virtual environments allow people to interact socially and to develop business collaborations on the Internet. Note: This was identified as a potential application area of distributed simulation in 1999, but today it has become a reality with popular Internet-based 3-D social networks like <i>Second Life</i> (Linden Research, 2007).
Medical application (potential area)	Computer generated virtual environments have been created both for doctors (to practice surgical techniques) and for patients (to treat various phobias). However, most of this work is currently limited to non-distributed virtual environments.
Transportation (potential area)	PDES can reduce the time taken to experiment with different strategies for responding to unexpected events like congestion resulting from weather conditions, etc. This will help take decisions faster.

Although the table lists only some of the application areas of distributed simulation, the fact that CSP-based simulation has not been identified as either a current or potential distributed simulation application area may seem to suggest that there is very little work done in the area of CSP-based distributed simulation. To further validate this observation, the DES CSPs will be examined with regards to in-built support for distributed simulation in section 2.6.4.

2.6.2 PDES theory

A simulation has to process events in increasing timestamp order. Failure to do so will result in *causality errors*. A causality error occurs when a simulation has processed an event with timestamp $T1$ and subsequently receives another event with timestamp $T2$, wherein $T1 > T2$. Since the execution of the event with time stamp $T1$ will have normally changed the state variables that will be used by the event with timestamp $T2$, this would amount to simulating a system in which the future could affect the past (Fujimoto, 1990). For a serial simulator that has only one event list and one logical clock it is fairly easy to avoid causality errors. In the case of distributed simulation, the avoidance of causality is a lot more difficult because it has to deal with multiple event lists and multiple logical clocks that are assigned to various processors. The reason for this is explained below.

The system being modelled (e.g., a factory) may be composed of a number of physical processes (e.g., distinct manufacturing units within the factory). In a distributed simulation, each physical process is usually mapped to a logical simulation process running on a separate machine. All the interactions between the physical processes (e.g., material movement from one unit of a factory to another) are modelled as messages that are exchanged between their corresponding logical processes. Each message will have a time stamp associated with it.

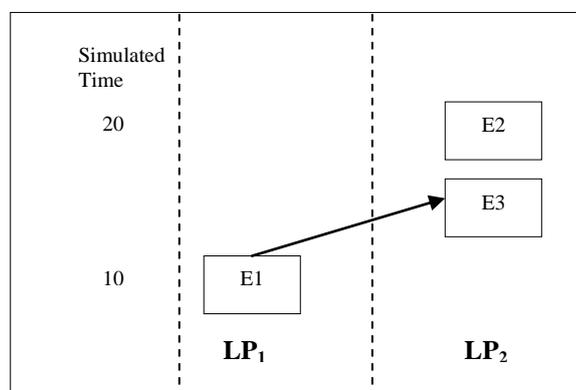


Figure 14: Execution of events in a distributed simulation (Fujimoto, 1990)

In the figure 14 above, the simulation represents a physical system that has two physical processes, say, PP_1 and PP_2 . Logical simulation processes LP_1 and LP_2 model the two physical processes. Each of these logical processes have their own simulation executive, simulation clock and an event list. During simulation initialisation the event lists of both LP_1 and LP_2 are populated with the events $E1$ and $E2$ respectively. The timestamps for $E1$ and $E2$ are 10 and 20 respectively. It will be possible for LP_1 to process event $E1$ without any causality error since the timestamp of $E1 <$ timestamp of $E2$. But LP_2 will not be able to execute event $E2$ at time 20 because causality error may then occur. The reason for this is that execution of $E1$ might schedule another event $E3$ for LP_2 at time 15. In such a case, if

LP_2 had been allowed to execute E_2 at simulated time 20 then it would have resulted in a causality error because the time stamp of $E_3 <$ the time stamp of E_2 . Different synchronization protocols are proposed for distributed simulation that prevent or correct such causality errors.

Synchronization protocols are one of the most important research areas of distributed simulation. They can be broadly divided into *conservative (pessimistic) protocols* and *optimistic protocols*. In a conservative protocol a processor is never allowed to process an event out of order; whereas in an optimistic protocol a processor is allowed to process an event out of order, provided it can revert back to its previous state in the case of a causality error (Nicol and Heidelberger, 1996). A pessimistic protocol like *Chandy-Misra-Bryant* (Chandy and Misra, 1979; Bryant 1977) implements the conservative synchronization protocol. Synchronization here is achieved through propagation of "null" messages (Chandy and Misra, 1979) or through *deadlock detection and recovery mechanisms* (Chandy and Misra, 1981). An optimistic synchronization protocol like *Virtual Time*, and its implementation called the *Time Warp mechanism*, executes events without considering the event time ordering (Jefferson, 1985). It has to save its state frequently so that a rollback to a previous state can occur when an event with a time stamp less than the current simulation time is received. There have also been several attempts to combine both conservative and optimistic approaches (e.g., *Local Time Warp*) in order to provide more efficient synchronization schemes (Rajaei et al., 1993). However, for the understanding of CSP-based PDES, the discussion that has been presented on pessimistic and optimistic synchronization protocols will suffice.

Based on the literature review of PDES algorithms it is possible to draw the following two conclusions:

- (1) Conservative and optimistic algorithms like *Chandy-Misra-Bryant* and *Virtual Time* are required for the execution of distributed simulations.
- (2) A DES CSP has to implement synchronization protocols, based on the conservative and optimistic synchronization algorithms, to provide support for distributed simulation.

2.6.3 Distributed simulation middleware

A distributed simulation middleware is a software component that implements the PDES algorithms to achieve synchronization between the individual running simulations. The next four sections of this thesis review four such middleware, viz., HLA-RTI, FAMAS, GRIDS and CSPE-CMB, that can be used to facilitate distributed execution of CSP-based simulations. Distributed simulation middleware like Aggregate Level Simulation Protocol (ALSP) (Fischer et al., 1994) and Distributed Interactive Simulation (DIS) (Miller and Thorpe, 1995) have been used widely in defence training simulations. However, there has been no reported application of these technologies to CSP-based simulations. As such they fall outside the scope of this research and will not be discussed further.

2.6.3.1 High Level Architecture (IEEE 1516.2000)

The High Level Architecture (HLA) (IEEE 1516, 2000) was originally proposed to address the need for interoperability between existing and new simulations within the U.S Department of Defense (DoD). This came from the need to reduce the cost of training military personnel by reusing computer simulations linked via a network. In the HLA, a distributed simulation is called a *federation*, and each individual simulation is referred to as a *federate*. A HLA *Runtime Infrastructure* (HLA-RTI) is a distributed simulation middleware, conforming to the HLA standards, that provides facilities to enable federates to interact with one another, as well as to control and manage the simulation.

The HLA is composed of four parts: a set of compliance rules (IEEE 1516.0, 2000), the Federate Interface Specification (FIS) (IEEE 1516.1, 2000), the Object Model Template (OMT) (IEEE 1516.2, 2000), and the Federate Development Process (FEDEP) (IEEE 1516.3, 2003). The rules are a set of ten basic conventions that define the responsibilities of both federates and the federation in the context of their relationship with the HLA-RTI. The FIS is an application interface standard which defines how federates interact within the federation. The FIS standard is implemented by the HLA-RTI. The HLA-RTI, thus, forms a base into which existing simulations (federates) can be "plugged into" to form a large distributed simulation (Fujimoto and Weatherly, 1996). There are several implementations of HLA-RTI available, for example, DMSO HLA-RTI (US Department of Defense Modelling and Simulation Office, 1999) and Pitch pRTI (Karlsson and Olsson, 2001). The OMT provides a common presentation format for HLA federates. FEDEP defines the recommended practice processes and procedures that should be followed by users of the HLA to develop and execute their federations.

For models created using CSPs to interoperate using the HLA standard, some of the FIS-defined interfaces have to be implemented. The FIS organises the communication between federates and the HLA-RTI into six different management groups. These are:

- Federation management: HLA-RTI calls for the creation and deletion of a federation, the joining and resigning of federates from the federation, etc.
- Declaration management: These pertain to the publication and subscription of messages between federates.
- Object management: Calls that relate to the sending and receiving of messages to and from federates.
- Ownership management: Calls for transfer of an object and attribute ownership.
- Time management: These provide synchronization services.
- Data distribution: For efficient routing of data between federates.

Mustafee and Taylor (2006a) have shown that a HLA-based CSP interoperability solution is possible by using services defined in at least four of these six management groups, viz.,

federation management, declaration management, object management and time management.

The time management component of the HLA supports interoperability among federates that use different time management mechanisms. These include federates executing simulations using both conservative and optimistic synchronization protocols (Fujimoto and Weatherly, 1996). One possible way through which time advance is coordinated in the HLA federation is now explained. A federate must explicitly request authorization from the HLA-RTI to advance simulation time to T . The HLA-RTI will grant permission to advance to T only when it can guarantee that all messages with a time stamp less than T have been delivered to the federate. This is the conservative synchronization protocol in action. Several services are also provided within HLA for the inclusion of optimistic federates such as those using the *Time Warp* synchronization protocol (Dahmann et al., 1997).

Almost all research in CSP interoperability using the HLA is concerned with conservative synchronization. This is probably because an optimistic approach is considered more complex as there is a need to save and restore system states periodically. Although it may be possible to save and restore the simulation system state by invoking a “file save” and a subsequent “file open” operation (through exposed CSP interfaces), this may drastically affect the performance of the simulation as both “file save” and “file open” operations are Input / Output (I/O) operations on persistent storage (not memory). Wang et al. (2004) have proposed the use of a HLA-based middleware called *rollback controller* for optimistic synchronization of CSP-based federates. However, at the time of writing, there have been no reported investigations pertaining to the integration of the rollback controller with a commercial simulation package. The subsequent discussions in this section focus on conservative CSP-based simulations using the HLA.

The problem of creating distributed simulations consisting of CSPs using the HLA was first addressed in Straßburger et al. (1998). CSPs can be perceived of as standalone “black box” packages that expose simple interfaces that are used to control the package and to access the model stored within the package. The main problem is therefore the manner in which the HLA-RTI software is interfaced to the CSP. Some examples of early HLA-related work are now presented. The IMS MISSION project attempted to use distributed simulation and CSPs within large decision support environments in manufacturing supply chains (McLean and Riddick, 2000). Individual research projects developed different, but incompatible approaches to the use of the HLA in support of distributed simulation with CSPs AnyLogic (Borshchev et al., 2002), AutoSched (Gan et al., 2005), Witness (Taylor et al., 2003); and simulation languages MODSIM III (Johnson, 1999), SLX (Straßburger et al., 1998), among others. Interoperability of models created using heterogeneous CSPs have been studied by Hibino et

al. (2002), whereby three commercial manufacturing simulators (QUEST, SIMPLE++ and GAROPS) have been interfaced with HLA using an adapter-based approach.

Building on the lessons learnt from these work, a standardization effort, described in Taylor et al. (2006b), specifically addressing the problems of HLA-based distributed simulation and CSPs began in 2002. This has led to the development of a suite of CSP Interoperability (CSPI) standards under the Simulation Interoperability Standards Organization's (SISO) CSPI Product Development Group (CSPI PDG). The CSPI PDG's standards are intended to provide guidance on how specific requirements of HLA-based distributed simulation can be supported with CSPs. These standards provide a set of Interoperability Reference Models (IRM) that describe different distributed simulation requirements, a set of Data Exchange Representations (DER) that are used to define the format of data exchanged between the models, and a set of Interoperability Frameworks (IF) that specify the architecture and protocol used to integrate the CSP with the HLA-RTI and exchange data in a time synchronized manner. Currently, there are six IRMs that describe the distributed simulation requirements for different scenarios; one DER and one IF (Taylor et al., 2006a).

Recent work on CSPI standards include Wang et al. (2006) who study possible implementations of the Type II IRM (synchronous entity passing); Taylor et al. (2005a) who investigate the use of distributed simulation in engine manufacturing; Gan et al. (2005) who investigate the use of distributed simulation in semiconductor manufacturing; Mustafee et al. (2006b) who report on using distributed simulation to model the supply chain of blood. The use of these standards within a semiconductor manufacturing decision support environment is discussed in Lendermann et al. (2005).

It is evident from this literature review that a lot of research in CSP interoperability is focussed on using HLA-RTI middleware. This is to be expected since HLA is an IEEE standard for distributed simulation and facilitates modular federation development using well-defined FIS. It is expected that the evolving CSPI PDG standards will encourage further research on using HLA to achieve CSP interoperability.

2.6.3.2 Generic Runtime Infrastructure for Distributed Simulation (GRIDS)

The Generic Runtime Infrastructure for Distributed Simulation (GRIDS) was first proposed as an architecture for studying bandwidth reduction techniques for distributed real-time simulations (Taylor et al., 1999). The GRIDS was then extended to provide distributed simulation environment for CSP interoperability (Taylor et al., 2001). Unlike the fixed functionality advocated by HLA, GRIDS was designed to provide only basic functionality for interoperation of different federates and a mechanism which would allow addition of extra services on an "on-demand" basis (Taylor et al., 2002). GRIDS was thus extensible. This extensibility was made possible by *Thin Agents* that were designed to provide additional services like different time synchronization algorithms, message filtering and security.

GRIDS was proposed because it is felt that HLA-RTI, designed primarily for interoperating military simulations, provided services that would possibly never be used in industry. It was felt that a lighter alternative supporting extensibility would be more suitable for distributed simulation in industry.

2.6.3.3 FAMAS.MV2 0.2 "Simulation Backbone"

FAMAS.MV2 0.2 "Simulation Backbone" is an architecture for linking different simulation models created as part of FAMAS.MV2 research programme (Veeke et al., 2002). It has also been proposed as a lightweight architecture for coupling of simulation models built using CSPs (Boer, 2005). In comparison to HLA, Famas provides only a limited set of subsystems (figure 15) for CSP interoperability (thus the term lightweight).

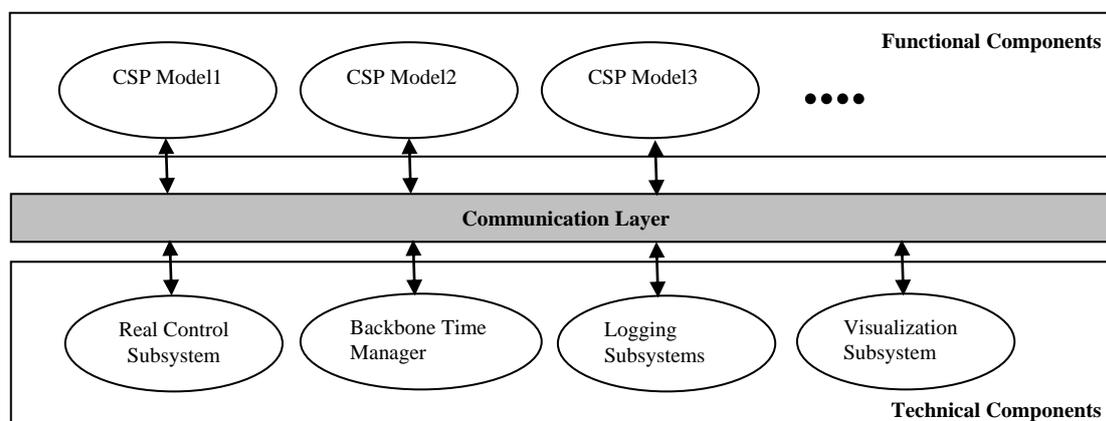


Figure 15: CSP-based distributed simulation using FAMAS (adapted from Boer, 2005)

The centralized Backbone Time Manager (BBTM) is probably the most important of these subsystems and is responsible for synchronization of several DES models running on multiple computers. It uses the conservative synchronization protocol. Each federate taking part in the distributed simulation sends the next event time to BBTM. BBTM selects the federate that has an event with the smallest next event time and grants it permission to execute that event. If two or more participating federates send the same event time then the BBTM gives federates permission for execution in First In, First Out sequence (Boer, 2002).

2.6.3.4 CSPE-CMB

The CSPE-CMB middleware (Mustafee, 2004) implements both the Chandy-Misra-Byrant "null" message algorithm (Chandy and Misra, 1979) and the deadlock avoidance and recovery mechanisms (Chandy and Misra, 1981). Unlike HLA-RTI, FAMAS or GRIDS middleware that depend on a central process (e.g., HLA-RTI depends on the central *rtiexec* process, FAMAS is dependent on *backbone time manager*) to grant individual simulations permission to advance their simulation clocks, CSPE-CMB implements a decentralized approach. This requires each federate to run the conservative synchronization algorithm and interact with other federates to find out the next safe time to advance the simulation.

The CSPE-CMB middleware has been used with a CSP emulator (Mustafee, 2004) to compare its performance with HLA-RTI on different interconnected federate topologies (Taylor et al., 2005b). It has also been used to successfully simulate three Simul8 models, each of which represents a part of a fictitious manufacturing assembly line consisting of a source, a variable number of queues and workstations and a sink (figure 16). Development of this middleware has since been discontinued in favour of CSPI PDG standards that encourage CSP interoperability through HLA-RTI.

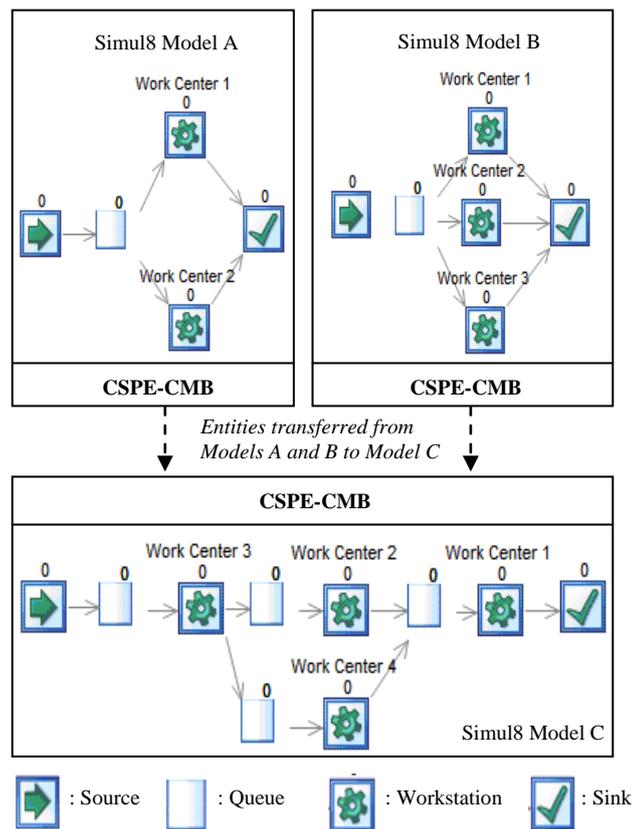


Figure 16: Distributed simulation using Simul8 and CSPE-CMB middleware

2.6.4 Distributed simulation service

From the above discussion it is clear that a higher-level grid service that facilitates the execution of distributed DES over grid resources will require the support of a distributed simulation middleware. Since a grid infrastructure consists of multiple computing resources, it will be possible to execute the individual DES models, which together make up a distributed simulation federation, over different grid resources. The distributed simulation middleware can be started on one of the grid nodes. In this thesis the distributed simulation service will be discussed in the context of enabling HLA-based DES over the grid.

Simulation practitioners may benefit from using a CSP that supports distributed simulation if they are involved in creating large and complex models (Mustafee et al., 2006b) or modelling supply chains (Gan et al., 2000; Sudra et al., 2000). Swain's survey (Swain, 2007) of CSPs,

complemented by the author's review of product information that is published in the vendor websites, is now used to investigate the level of distributed simulation support present in existing DES CSPs. This is presented in table 14 below.

Table 14: CSPs and distributed simulation support

Software	Vendor	Features	Info. Source (Appendix A.7)
Arena	Rockwell Automation	Interprocess Communication is possible. IMPORTANT: <i>No further information is available on the website.</i>	Vendor website
AutoMod	Brooks Software	<i>The Model Communications Module</i> allows information to be transferred between models and control systems, multiple models, and models to other applications.	Vendor website and from author's discussions with simulation experts
Simprocess	CACI Products Company	Simprocess provides support for external entity schedules that allow external applications to "feed" entities to a SIMPROCESS model. This allows the user to develop portions of a SIMPROCESS model and distribute it to separate computers to share the workload. IMPORTANT: <i>Although entities can be transferred between models the simulation clocks across the separate computers are not synchronized.</i>	Vendor website

Of the three CSPs that have been identified as providing some sort of distributed simulation support, only AutoMod has some form of distributed model execution capabilities. In Simprocess there is no mechanism for time synchronization between the running models and therefore it cannot be considered as providing distributed simulation support. Arena allows some form of inter-process communication but it does not necessarily suggest that it supports distributed simulation.

Robinson (2005b) has highlighted "distributed model execution" and "linking separate models" as two potential applications of simulation in a networked environment under the "model execution category" (table 8). Both these applications can be considered as distributed simulation because, distributing the execution of a simulation by (1) splitting a large model into sub-models and linking them or (2) by linking existing models together is frequently referred to in literature as distributed simulation (Hibino et al., 2002; Wang et al., 2006; Taylor et al., 2006a).

2.7 Web-based simulation

According to Page et al. (2000), "Web technology has the potential to significantly alter the ways in which simulation models are developed (collaboratively, by composition), documented (dynamically, using multimedia), analyzed (through open, widespread investigation) and executed (using massive distribution)". Observations relating to the use of web technology in the field of simulation can be found in literature (Pidd et al., 1999; Fishwick, 1997; Kuljis and Paul, 2000). These are:

- Web enables distributed component-based simulation.

- Parallel and distributed model execution is possible over the web.
- Distributed model repositories can be made available for simulation practitioners.
- Simulation education and training can benefit from free and widely accessible modelling environments made available on the web.
- Web-based simulation of autonomous software agents is possible.
- Scientific simulations that require, for example, execution of software applications on multiple machines, data stored on various locations, etc., is facilitated over the web.

The Java programming language is increasingly being used for implementing web-based applications. Some of the advantages of using Java as a platform for creating web-based simulations are (Pidd et al., 1999; Page et al., 2000):

- Java is an object oriented programming language and is therefore suitable for component-based simulation.
- Simulation models in Java can be made widely accessible through Java applets that can be downloaded by client browsers.
- Java is platform independent and, thus, Java applets can be run on any Operating System that has Java Runtime Environment (JRE) installed.
- Java has built-in threads that can be put to good use in modelling simulations.
- Java provides support for graphics.
- Some aspects of Java such as multi-threaded programming are generally considered easier to learn compared to some other programming languages.

Kuljis and Paul (2000) present an overview of several Java based DES environments like DEVJSJAVA, JavaGPSS, Silk, JavaSim, Web-enabled Simulation Environment (WSE), etc. that either support web-based simulation (like WSE) or can be considered as potential candidates to offer such web-based simulation functionality in the future. In this thesis, however, web-based simulation is discussed only in the context of MCS and DES CSPs. Most of the applications of web-based simulation that have been described above (for example, parallel and distributed model execution, model composition using components, massive distribution, joint model development, etc.) have already been discussed in the context of higher-level grid services that can be potentially used with CSPs.

2.7.1 Defining web-based simulation

For the purpose of this research it is important to distinguish between simulations running on a networked environment (network-based simulation) and simulations running over the web. In this thesis all web-based simulations are also considered as network-based simulations since they rely on multiple computing resources that are connected through a network. However, all network-based simulations are not considered as web-based simulations since they may not use the underlying World Wide Web technologies but may implement customized distributed computing solutions. A discussion on Internet, Intranet, World Wide

Web (WWW) and WWW-technologies will make this distinction more apparent. Kurose and Ross (2003) has been extensively referenced in this discussion.

The public **Internet** is a worldwide computer network that connects millions of end systems (computing devices like desktop PCs, UNIX workstations, servers, PDA's, televisions, game consoles, etc) and intermediate switching devices (like routers, hubs and switches) that mainly run the **TCP** (the Transmission Control Protocol) and **IP** (the Internet Protocol) protocols to control the sending and receiving of information between them. The private **Intranet** uses infrastructure (end systems, switching devices and communication links) and protocols similar to that of the Internet but its access is confined within an organization.

The Internet Protocol stack is based on layered protocol architecture, with each protocol belonging to one of the layers and providing a service to a protocol belonging to a layer above it. The Internet Protocol stack consists of five layers – the physical layer, the data link layer, the network layer, the transport layer and the application layer. The *Open Systems Interconnection Basic Reference Model*, or OSI Reference Model for short, has two additional layers – session layer and presentation layer. But for the purpose of this discussion an overview of five layers will be sufficient. The protocol layers can be implemented in software (like protocols in the application and transport layers, example, HTTP, TCP, UDP), hardware (like protocols in the physical and data link layers), or in a combination of both (like protocols in the network layer). The hardware is generally the Network Interface Card (NIC). Each layer communicates with the other by exchanging layer-specific messages. The Internet protocol stack is shown in figure 17 below. Messages generated at the application layer (layer-5 messages) flow down the protocol stack, and at each layer these messages are complemented with further layer specific data to create a corresponding layer-specific message. Thus, layer-5 message becomes a layer-4 message in the transport layer. A brief discussion of the protocol layers follows next.

Application Layer (Layer -5 messages)
Transport Layer (Layer -4 messages)
Network Layer (Layer -3 messages)
Link Layer (Layer -2 messages)
Physical Layer (Layer -1 messages)

Figure 17: Layered architecture of Internet Protocol (IP) stack

The application layer consists of protocols that support network applications, for example, Hyper Text Transfer Protocol (HTTP) that supports web-based applications, Simple Mail Transfer Protocol (SMTP) for email, File Transfer Protocol (FTP) to support file transfer, etc. The transport layer protocols like TCP and User Datagram Protocol (UDP) provides services for transporting layer-5 messages. The network layer is responsible for routing layer-3

messages from one host to another. It consists of the IP protocol that defines the fields in the layer-3 messages as well as how the end systems and routers act on these fields. Furthermore, it specifies the routing protocols that determine the routes these layer-3 messages take between sources and destinations. The link layer routes layer-2 messages through a series of routers between source and destination. Finally, the physical layer is responsible for moving individual bits in a layer-1 message from one node to the other.

The **World Wide Web**, commonly referred to as the **Web**, is an Internet (and Intranet) application that uses HTTP as its application layer protocol. HTTP protocol is implemented using a client program and a server program, each of which executes on different end systems and communicate with each other using structured HTTP request and response messages. HTTP uses the TCP as its underlying transport protocol. The communication between the client and server programs is shown in figure 18 below (adapted from Kurose and Ross, 2003).



Figure 18: Communication between client and server programs

The web application consists of a hierarchy of interconnected web pages that are accessible to the user through navigation elements called hyperlinks. The web pages comprise of objects like HTML (Hyper Text Markup Language) files, graphics images, Java applets, audio clips, video clips, etc. and are accessible through a unique Uniform Resource Locator (URL). The URL (for example, `www.brunel.ac.uk/students/library.htm`) identifies the end system running the server program (the domain name *brunel.ac.uk* is translated into a specific machine address through the Domain Name Server [DNS] service) and the path to a web page (*students/library.htm*). Web pages are requested by the end system running the client program from the end system running the server program through invocation of the URLs. The client programs are the web browsers like Internet Explorer, Mozilla Firefox and Netscape; the server programs are web servers like Internet Information Server (IIS), Apache HTTP Server and Sun Java System Web Server.

The overview of the World Wide Web, arguably the most popular Internet application to date, as presented above is the basis of the definition of web-based simulation that is presented in this thesis. Web-based simulation is defined as simulation in a client-server based distributed computing environment that uses web-based technologies like web browsers, web servers, hyperlinks, URLs, among others. The CSPs that support web-based simulation may therefore make their CSP applications accessible to simulation practitioners through web pages. These

web pages are rendered by the client browsers (running on the PCs of simulation users) and may contain static text, dynamic simulation applets, HTML form elements (like input boxes, list boxes) for inserting experiment parameters, etc.

2.7.2 Web-based simulation and CSPs

Lorenz et al. (1997) have described three possible approaches to web-based simulation using existing simulation packages. They refer to these packages as simulation and animation (S&A) tools. The first approach is the *remote S&A approach* where the user specifies parameter values for a simulation model using web-based HTML forms and submits it to the PC that is hosting the web server. The web server invokes the S&A tool, executes the simulation based on the input values returned by the form, and sends back the results to the user. The drawback of this approach is that the animation of the simulation cannot be viewed by the user and he has no control over the simulation once it has started running on the server. The second approach is defined as the *local S&A approach* where the user downloads Java applets (which include the code for simulation executive) from the server, loads it into the web browser and then runs the simulation on a local machine. This approach supports animation and user interaction with the simulation model. The third approach is referred to as *animation and manipulation using a Java data server*. In this case the simulation runs remotely on the server; however, the user is able to view the animation and exert some control over the running of the simulation through a Java applet (downloaded from the server) that establishes a connection with the Java data server and gets a continuous data feed from the running model. The three approaches are graphically shown in figures 19, 20 and 21 below. The figures have been adapted from Lorenz et al., (1997). It is generally possible to interface the CSPs that expose package functionality (see table 10) with an application running over the web server. For example, Whitman et al. (1998) have implemented the remote S&A approach to web-based simulation using DES CSP Witness (that exposes package functionality) and web-based technologies like VBScript and HTML. As web service is a web-based technology, access to CSPs through use of web services also qualifies as web-based simulation.

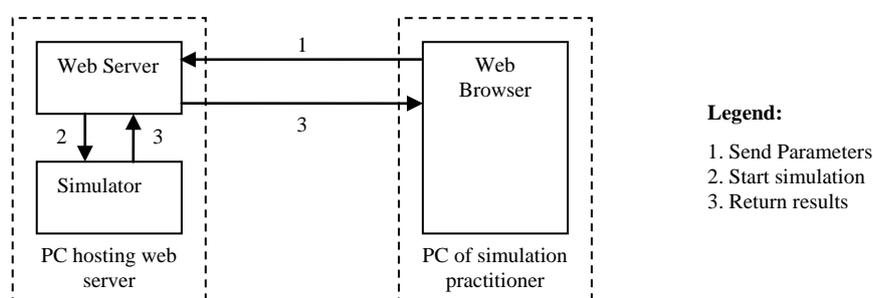


Figure 19: Remote S&A approach to web-based simulation using CSPs

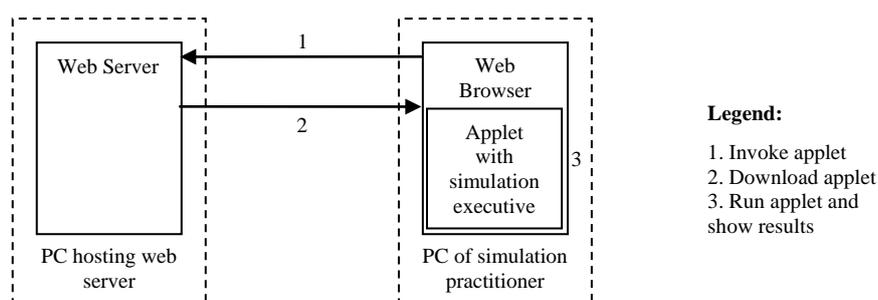


Figure 20: Local S&A approach to web-based simulation using CSPs

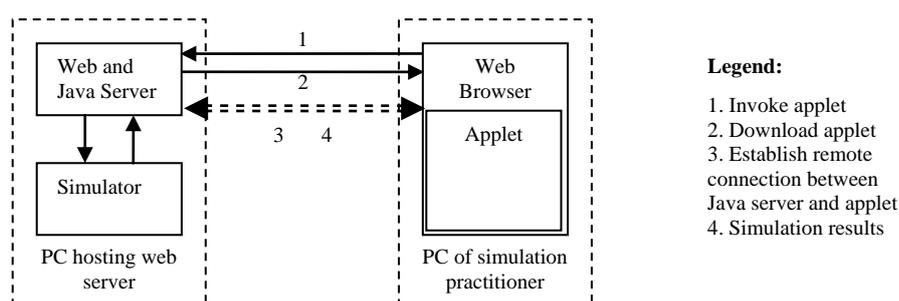


Figure 21: Java data server approach to web-based simulation using CSPs

The CSPs that support web-based simulation through static HTML pages, dynamic Java applets and callable web services are listed in table 15 below.

Table 15: CSPs that provide support for web-based simulation

Software	Vendor	Features	Information Source (Appendix A.8)
Quantitative Methods Software (QMS)	QuantMethods	QMS runs as a <i>client-server</i> application. The problem solution engine is the server and the browser is the client. The browser is used to create, edit, and optionally store problems; and to view and print the output. The server accepts input from the browser, generates solutions to problems, and sends the output to the browser. Since the software is accessed through the browser, there is no need to install QMS on every machine.	Vendor website
MineSim	Systemflow Simulations, Inc.	MineSim is an interactive 3-D web-based simulation of an underground mine. MineSim is written in Java and runs as an applet in the browser. The applet downloads the MineSim application to the local machine but does not install any program.	Vendor website
Vanguard Studio (DecisionPro)	Vanguard Software Corporation	<i>Vanguard Application Server</i> makes all models, built using Vanguard Studio and a <i>Web Development Add-in</i> , available as web-based applications that anyone in the organization can access using only a Web browser interface. Vanguard's <i>Web Services Add-in</i> allows inter-connection between the Vanguard models, that are executed on Vanguard server and other enterprise systems.	Vendor website
AnyLogic	XJ Technologies	AnyLogic models can be placed on a website as applets. It allows clients to run fully functional interactive models directly in their web browsers without installing any kind of runtime or viewer version.	Vendor website

Software	Vendor	Features	Information Source (Appendix A.8)
AgenaRisk Enterprise Edition	AgenaRisk	The AgenaRisk API is a set of java routines that lets the user create, edit and execute AgenaRisk models as part of a client server, web service or desktop enabled application.	Vendor website
Witness	Lanner	<i>Witness Server</i> is an additional module for Witness that runs on a central server and serves multiple users. There is no need for Witness to be installed on every computer. Data for a simulation experiment can be set through a webpage or through customized dialogue boxes. After the data has been set, an experiment can be submitted to the central server and the job monitored. There is an optional web site hosting facility on the server.	Vendor website
Analytica	LuminaDecision Systems, Inc	<i>Analytica Decision Engine (ADE)</i> can deliver Analytica models as a Web application.	Vendor website
Simprocess	CACI Products Company	SIMPROCESS has the capability to provide simulation models as callable services through the use web services. The models are executed on the server.	Vendor website

The table above shows that only eight out of the 45 CSPs that have been surveyed in this research support web-based simulation. Based on the limited adoption of web technology by the CSP vendors on one hand, and the ever gaining popularity of WWW-based applications on the other, it would be interesting to investigate whether a grid-facilitated web-based simulation service could be use together with the CSPs.

2.7.3 Web-based simulation service

The discussions on higher-level grid services in section 2.2.2 have described the grid portal service as a web-based application that provides users with higher-level abstraction to the underlying grid services. Use of grid portals may make it possible for the user to upload simulation models and experiment parameters, monitor simulation progress and to download the results of the simulation using their web browsers. The grid portal interfaces with the grid middleware to provide these services to the users. Unlike custom web-based CSP solutions that are implemented by vendors for particular CSPs (see table 15), grid portals are generally not targeted at specific applications (for example, Simul8) or particular application domains (for example, simulation). As such, the level of CSP-specific functionality that can be provided by grid portals is usually limited when compared to the functionality provided by custom CSP-specific solutions.

Screenshot 2 shows the job submission web page for the NGS portal (Yang et al., 2005). As can be seen from the screenshot, the web page provides input boxes to specify the path for the executable (which can be the CSP), the input file (which can be the simulation model), the output file (which can be used to collect the results of the simulation), links to specify the arguments (which can be the different simulation experiment parameters), etc. The screenshot only shows a part of the job submission web page and the reader is referred to <<https://portal.ngs.ac.uk>> for more details.

NGS JSDL Application Repository /
National Grid Service Job Submission Portal (JSR-168 Compliant) ③

[Start](#) | [Authenticate/ Credentials](#) | [Applications Repository](#) | [Upload/ Download](#) |

④ JSDL POSIX Extensions:

④ WORKINGDIR (Mount Point Path): <Clear> ④

④ Create New (Use Optional Name): <Create WORKINGDIR> ④

(Specify path Relative to the [File System Mount Point](#) OR Full path starting with '/' - both produce valid JSDL executable element) ▾

④ Executable Or Script:

④ File Paths: Std In/Out/Error Files: (Specify paths Relative to [File System Mount Points](#) - Recommended) (Can also specify a Full path starting with '/') ▾

④ Input File (Must Exist):

④ Output File (Created if not exist):

④ Error File (Created if not exist):

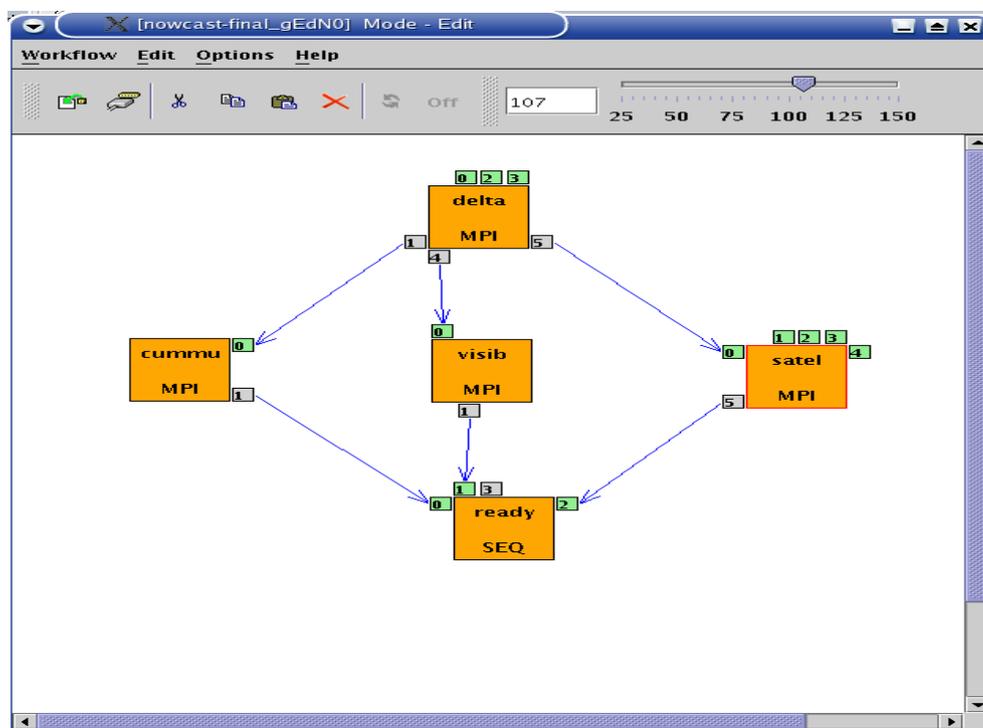
④ JobType + ProcessCount:

④ Wall Time:

④ Arguments + Environment: [Environment](#) [Arguments](#)

Update Active Job Profile

Screenshot 2: Job submission web page for the NGS portal



Screenshot 3: Workflow editor in P-GRADE portal (adapted from Kiss, 2007)

Some grid portals provide a GUI interface to create workflows using Java-enabled web browsers. The P-GRADE portal (Németh et al., 2004) and the NGS portal are examples of grid portals that offer such functionality. Screenshot 3 (previous page) shows the P-GRADE workflow editor. The workflow created using the editor consists of 5 individual jobs (four MPI parallel jobs and one sequential job), wherein the MPI jobs “cummu”, “visin” and “satel” are dependent on data that is output from the MPI job “delta”. Similarly, the sequential job “ready” is dependent on data that is output from “cummu”, “visin” and “satel” MPI jobs.

Web-based simulation service could be potentially provided through the use of web services also. It has been discussed earlier in section 2.2.3.1 that grid computing middleware has traditionally been implemented using custom protocols. However, with the introduction of the web services oriented OGSA framework it is widely believed that middleware based on OGSA standards will become increasingly available in future. A cluster-based grid middleware that implements OGSA standard is Globus GT-4. It allows the creation of user-developed services (based on web services) that can be hosted in GT-4 implemented Java, Python or C containers (figure 6). These containers provide mechanisms for security, service discovery and management, etc., which are usually required for building services in the grid environment. What it means for web-based simulation is that “callable” web services that expose CSP functionality can be deployed through grid middleware. The simulation user can then write applications that call these web services to realize web-based simulation over grids.

This section of the thesis has described web-based simulation and has identified that grid-facilitated web-based simulation service can potentially support CSPs through use of grid portals and through mechanisms to host “callable” web services that expose CSP interfaces. Informed by the discussions on grid middleware (section 2.2.3) and the different forms of grid computing (section 2.2.5) in the earlier sections, the next section investigates the form of grid computing that is suitable for CSP-based simulation in industry.

2.8 Grid middleware and CSPs

Section 2.2.5 has identified four different forms of grid computing. These are cluster-based grid computing, enterprise-wide desktop grid computing (EDGC), public resource computing (PRC) and peer-to-peer computing (P2P). The discussions in section 2.2.3 have highlighted that the middleware for cluster-based grid computing are primarily targeted at UNIX and Linux flavour operating systems (the only notable exception being Condor middleware). Middleware for EDGC, PRC and P2P, on the other hand, are widely support under the Windows platform.

The OR/MS survey of CSPs, complemented by the author’s own investigation of simulation software, has shown that all packages are supported on the Windows platform, 15.56% on both UNIX and Linux operating systems and only 13.33% CSPs are supported on Macintosh.

This shows the prevalence of Windows-based CSPs in industry. Furthermore, it is a widely accepted observation that employees generally use the Windows-based systems at their workplace. It is therefore arguable that for this research to be widely relevant to the practice of CSP-based simulation in industry, it should, first and foremost, focus on Windows-based grid computing solutions. Discussion of cluster-based grid solutions for CSP-based simulation modelling is thus outside the scope of this thesis. P2P computing is also not investigated further because it generally supports only file sharing and as such P2P networks cannot be used to execute programs (like CSPs) on the peer resources. From this point on, the terms “desktop grid computing”, “desktop grids”, “grid computing” and “grids” will be used synonymously to refer to only PRC and EDGC, unless explicitly stated. Two middleware, chosen in this research as representative forms of either the EDGC or the PRC form of grid computing, are now discussed, namely, BOINC and Condor.

BOINC is an open source PRC middleware that allows users to create new BOINC-based projects to cater to their computational needs. Condor is an EDGC middleware that is used for both e-Science research and for enterprise application processing. Both BOINC and Condor are cycle stealing systems (i.e., a system that harnesses the unused CPU cycles of idle PCs to process other jobs in the background) that can run on non-dedicated Windows PCs.

The rationale of choosing BOINC as a representative form of PRC middleware is as follows.

- It is presently the most popular PRC middleware.
- It is presently the only PRC middleware that allows users to create their own projects.
- It is available free of cost.

The rationale of choosing Condor as a representative form of EDGC middleware is as follows:

- It has the largest EDGC deployment base. More than 80,000 Condor hosts around the world make up approximately 160 production-level Condor pools (see <http://www.cs.wisc.edu/condor/map/> for updated Condor statistics).
- It is available free of cost. Other EDGC middleware like Entropia DCGrid, United Devices GridMP and Digipede Network are commercial solutions.

BOINC and Condor are discussed next in sections 2.9 and 2.10 respectively. The purpose of this discussion is to acquire an in-depth understanding of these systems, which would in turn allow proper evaluation of the middleware in respect to its suitability for providing higher-level grid services to the CSPs in later chapters. Unfortunately none of these middleware are OGSA compliant or support hosting of user-developed web services using custom solutions. Therefore some grid-facilitated higher-level services that require web service support cannot be evaluated using these middleware, namely, “web-based simulation through the use of web

services” (web-based simulation service) and “searching and downloading CSP model components” (collaboration service). These are areas for further research.

2.9 Public-Resource Computing (PRC) middleware BOINC

2.9.1 Overview of PRC

Public-resource computing (PRC) refers to the utilization of millions of desktop computers primarily to do scientific research (Anderson, 2004). Berkeley Open Infrastructure for Network Computing (BOINC) (BOINC, 2007b) is the most widely used PRC application that supports scientific projects with diverse objectives such as studying climate change (Stainforth et al., 2002), improving the design of particle accelerators (LHC@home, 2007) and finding cures for human diseases (Taufer, 2006). BOINC was developed by those responsible for the PRC project SETI@home (Anderson et al., 2002), which originally used bespoke software to search for evidence of extraterrestrial intelligence in radio signals. BOINC now provides a generic set of tools and patterns which are used to support a wide range of PRC projects. Presently, BOINC is used by around 20 such projects, which together consume an estimated 350 teraflops of processing power, generated by approximately 1 million computers contributed by some 600,000 volunteers (Anderson, 2006). Non-BOINC based projects use bespoke software to facilitate research with similar objectives, for example, finding a cure to cancer (Parabon computation, 2007), understanding protein folding (Pande, 2007) and computing mersenne prime numbers (Woltman, 2007).

The participants of PRC projects are volunteers who register with one or more such projects and install the required PRC software. This software then contacts the central project servers and downloads work units for processing (in the case of BOINC it also downloads project specific executable code as BOINC is a general purpose PRC client). BOINC implements the master-worker distributed computing architecture and uses the “pull” mechanism for scheduling jobs, where the volunteer computers request (pull) jobs from the PRC project servers (figure 22). The time it takes to complete the execution of a work unit and return back the result depends, among other things, on the machine hardware, the amount of time a PC is left running and user preferences. The volunteers are themselves unable to use the underlying desktop grid infrastructure, of which they themselves are part of, to perform their own computations.

2.9.2 BOINC architecture

The BOINC system [figure 23, adapted from (Anderson, 2006) and (Perez, 2005)] contains several server-side components, which may execute on separate machines if required. Most of the server side components can only be installed over a UNIX or Linux flavour operating system. The database holds all the metadata associated with the project and lifecycle information for each work unit. A client’s command channel operates via the scheduling server, using an XML-based protocol. Results are transferred using HTTP via the data

servers. In addition to work units and results, other files may be transferred between server and client, including application executables and any other interim data the application may require during the operation. The database also has a web-based front-end that is used for displaying project information specific to volunteers, for example, how many computers have been contributed by the user, the number of work units processed, etc. On the client side, the *BOINC core client* manages interaction with the server, while optional components (like screensaver and manager) provide graphical control and display elements for the benefit of the user. The BOINC client API provides the interface between the user-created *application client* and the BOINC core client. The API is a set of C++ functions and the application client is compiled with it. In other words, the BOINC application client will generally have to be written in C++ (BOINC, 2007c). All communication between the BOINC core client and the BOINC project servers take place through HTTP on port 80 (BOINC, 2007d). The BOINC core client can therefore operate behind firewalls and proxies.

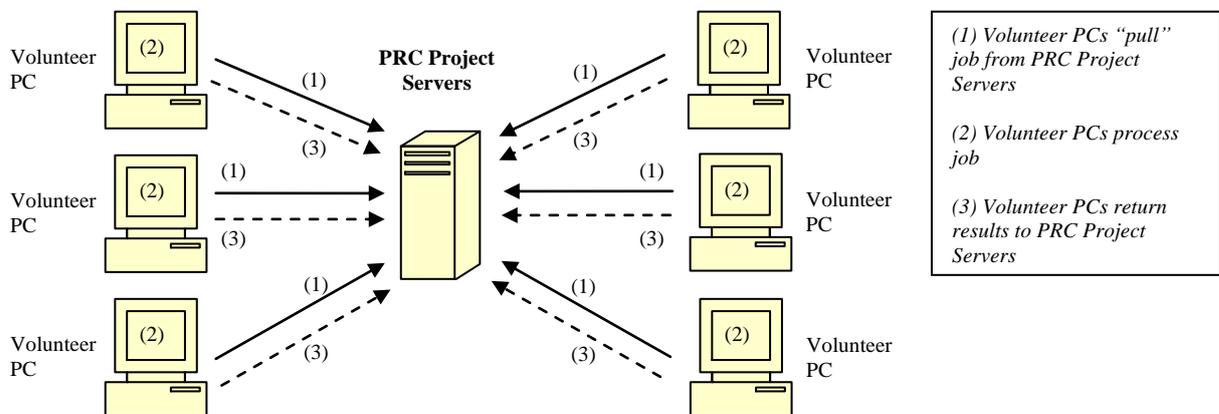


Figure 22: The "pull" model of PRC projects

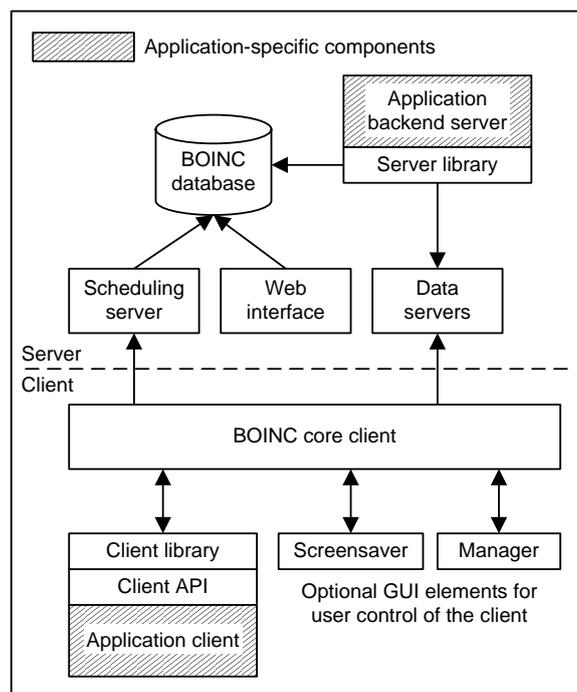


Figure 23: The BOINC system

BOINC has been primarily designed and developed for use as PRC software. As such, some of its design goals arise from the need to (1) attract new participants and to retain existing ones, (2) to guarantee the correctness of results being returned by clients processing work units using heterogonous computing resources, and (3) to ensure that a modest BOINC server setup will be capable of handling tens of thousands of client requests. BOINC implements these design goals by providing support for redundant computing (whereby each work unit is sent to multiple clients for processing in order to identify and reject erroneous results), implementing exponential back off on failure (this allows a BOINC server to gracefully process client requests even after an extended outage), rewarding the participants in the form of a credit system and recognising them through web-based “leader boards”, facilitating community building through the creation of teams, and finally, providing graphics visualization as an inducement to further attract and retain participants (Anderson, 2004).

2.9.3 BOINC in an enterprise setting

Although BOINC was originally designed to support PRC, lately there has been a realization that the same software can be reconfigured to support desktop grid computing (BOINC, 2007a). The widespread availability of desktop PCs in organizations makes the deployment of such an enterprise-wide BOINC infrastructure an even more attractive option. Thus, it may be possible to implement and deploy BOINC-based projects for use exclusively within an enterprise, such that it is geared up to support the execution of the enterprises' applications. The participants of such an enterprise-wide BOINC setup can be the employees of the organization who contribute their work PCs. The participation in such projects may not be voluntary and can be governed by the policy of the organization. The computations being performed by the BOINC clients will be in line with the needs of the enterprise, and unlike PRC where volunteers are encouraged to contribute their resources, only employees and other trusted sources will be allowed to participate in the enterprise-wide BOINC projects. BOINC features that are necessary in the PRC context but may not be required in an enterprise grid (for e.g., user rewards system, anti-cheating measures, mechanisms to deal with client failure or extended network non-connectivity, etc.) can be disabled.

In the PRC setting, project specific application clients are downloaded from the server by the BOINC core client as required. Only BOINC itself needs to be pre-installed on each client computer. This type of BOINC application can be referred to as a *'runtime application client'* (BOINC-RAC) because there are no client-side dependencies for application code. In an enterprise environment such a standalone executable application client may encourage participation outside of the project sponsor's department. A disadvantage is the need to package applications in the downloadable form that BOINC requires, which may require development work.

Within the enterprise employee computers are frequently installed with office productivity applications. When these pre-installed applications are used for client side processing then only a small application client is required to be downloaded by the BOINC core client. This type of BOINC application can be referred to as a *'proxy application client'* (BOINC-PAC) because it processes enterprise data by triggering pre-installed desktop applications. However, this approach may incur additional administration overheads such as ensuring that security permissions and application versions are correct on every participating client machine.

BOINC PRC applications vary widely in their installed footprint, size of work unit, and disk and memory space needed during execution (Christensen et al., 2005). In an enterprise setting, the choice of BOINC-RAC versus BOINC-PAC will depend on these practical factors as well as the administrative policies in place. In a BOINC-based desktop grid environment the inter-departmental participation in a project may vary depending on which of these two approaches is implemented. For BOINC-RAC applications it is relatively easy for different departments to participate in projects because such applications do not impose any client side dependencies. However this inter-departmental camaraderie may not always be possible in the case of BOINC-PAC applications because they require invocation of third-party software which first has to be installed on client PCs.

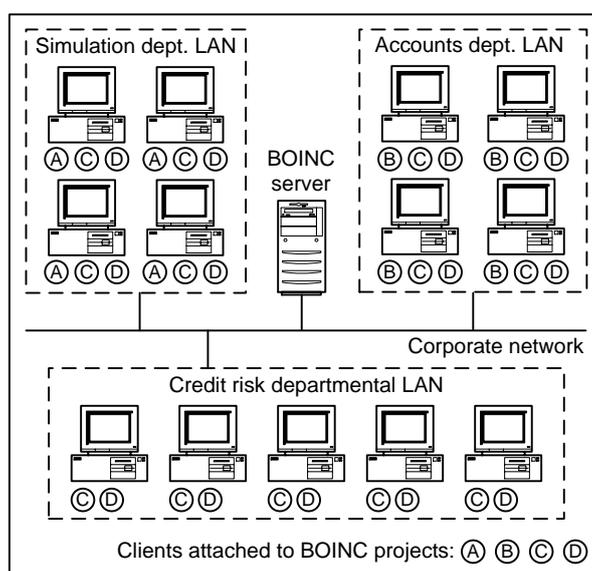


Figure 24: Multiple BOINC projects in an organization

For example, the simulation department may create the BOINC-PAC project "A" using a specialist software package like DES CSP. But the accounts department in the same organization may not be able to participate in such simulation projects because their departmental PCs are only installed with specialist financial software. They can, however, create a BOINC-PAC project "B" to handle their own processing requirements. The credit risk department may create BOINC-PAC project "C" that requires Microsoft Excel. Since Excel is

a widely used application, it can be expected that it is installed on most PCs in the organization. Thus, the simulation and the accounts departments can join in with the finance department to execute the Excel-dependent BOINC application on their respective departmental resources. Similarly, a BOINC-RAC application (project “D”) created by the accounts department can be easily executed by all three departments due to the lack of client-side dependencies. Figure 24 shows these four different BOINC execution scenarios.

This section has presented an overview of PRC and has discussed the architecture of BOINC and how it can be used in an enterprise setting. A discussion of Condor is presented next.

2.10 Enterprise Desktop Grid Computing (EDGC) middleware Condor

The Condor project was born in the University of Wisconsin-Madison in 1988. Condor is an opportunistic job scheduling system that is designed to maximize the utilization of workstations through identification of idle resources and scheduling background jobs on them (Litzkow et al., 1988). A collection of such workstations is referred to as a Condor pool. Condor has mechanisms to checkpoint running jobs (i.e., save the state of a program that is being executed) and migrate them to other workstations, when the previously idle resource are reclaimed by the PC owners (Litzkow et al., 1997). When Condor was first introduced in 1988 it was unique because it was arguably the only production system that allowed every user to contribute as much or as little of their resources, and offered an alternative to the dominant centralized processing model of the day (Thain et al., 2004).

Condor established the term *High Throughput Computing* (HTC) to distinguish a distributed computing environment that could deliver large amounts of processing capacity over long periods of time (i.e., it focuses on providing an increasing number of floating point operations over time), with the centralized *High Performance Computing* (HPC) environment that focuses on delivering an increasing number of floating point operations per second (FLOPS) (Livny and Beck, 1997). HTC is thus a 24 hours a day, 7 days a week, 365 days a year activity with non-dedicated user computers. As desktop PCs become faster, cheaper and more widely available, the aggregate processing power that could be made available using Condor HTC is constantly on the rise.

Although Condor was originally designed to provide HTC through cycle stealing, the same system design can also be used to manage Beowulf clusters, multi-processor machines and wide-area distributed systems; for example, the Condor pool at the University of Wisconsin-Madison manages workstations, several clusters, and several multiprocessors all in one system and a Condor pool in Italy harnesses resources from workstations spread throughout ten cities (Condor, 2007). The focus of this thesis is however on using Condor on a network of commodity PCs.

Over the years the functionality provided by Condor has steadily increased to include features like Condor flocking (two or more Condor pools in different administrative domains that are linked together), multiple Condor universe (each universe supports one specific job execution environment, e.g., Condor MPI universe supports execution of MPI programs), Condor-MW (specifically for master-worker type applications), Condor-G (the job management part of Condor that allows users to submit jobs to clusters running Globus middleware), Condor Directed Acyclic Graph Manager (DAGMan supports workflow), Chirp protocol (lightweight remote I/O protocol that can be used with Condor), NeST (resource manager for Condor network storage), among others (Condor Version 6.9.1 Manual, 2007b; Condor DAGMan, 2007; Condor MW, 2005). In the subsequent sections of this thesis a subset of these features that are considered appropriate for providing grid-facilitated higher-level services to the CSPs are investigated. The next section looks at the architecture of Condor.

2.10.1 Condor architecture

Condor HTC architecture defines resource providers and resource consumers. The resource providers make their resources available to Condor for the processing of jobs that originate from the resource consumers. The jobs to be processed may have dependencies with regards to the operating system on which the job is to be processed, the memory and disk space required, the available software libraries that are needed and so forth. On the other hand, the resource providers may have certain conditions (e.g., only Java jobs can be run) and preferences (e.g., jobs originating from resource consumer “x” is given priority) based on which access to their resource is granted. Condor allows resource consumers and resource providers to advertise these requirements, conditions and preferences by providing a language called *classified advertisements (ClassAds)* that provide a flexible and expressive framework for matching jobs originating from the former with resource offers from the latter (Thain et al., 2004).

The ClassAds are scanned by a Condor *matchmaker agent* (an agent is a Condor software component), running on only one computer in a Condor Pool, to find a match between the requirements advertised by the *resource consumer agents* (representing the resource consumers) and the resources advertised by the *resource provider agents* (representing the resource providers). The same computer can run both resource consumer and resource provider agents. Once a match has been found by the matchmaker agent, it notifies both the resource consumer and the resource provider agents. Upon receiving this notification, the resource consumer agent claims the resource advertised by the resource provider agent through a claiming protocol. The job is executed by the resource provider agent and the results of the computation are returned back to the resource consumer agent. The matchmaking process is illustrated in figure 25. The figure has been adapted from Basney and Livney (1999). The Condor matchmaker agent can be considered as the resource broker in a Condor pool. The existence of the Condor matchmaker agent as a broker introduces an extra layer of communication between the resource consumer and the resource provider

agents. As such, it is arguable that the Condor resource management architecture does not directly map either to the “pull” or the “push” job scheduling mechanism (although, after a match has been found, the resource consumer agent may “push” the job to the resource provider agent [Robinson and DeWitt, 2007]). Condor’s “broker-based” job scheduling mechanism is important for later discussions and therefore it is discussed in more technical detail below.

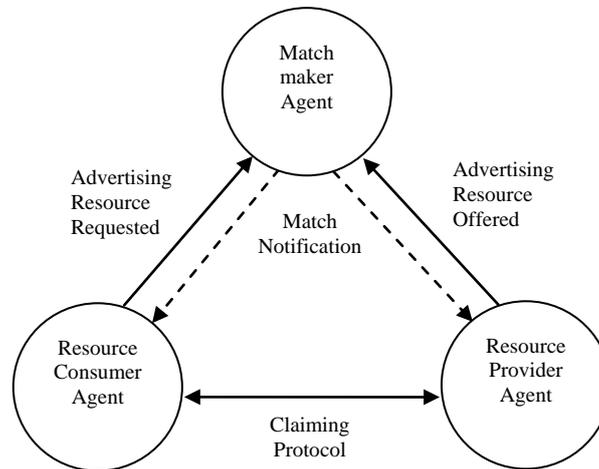


Figure 25: Condor resource management architecture

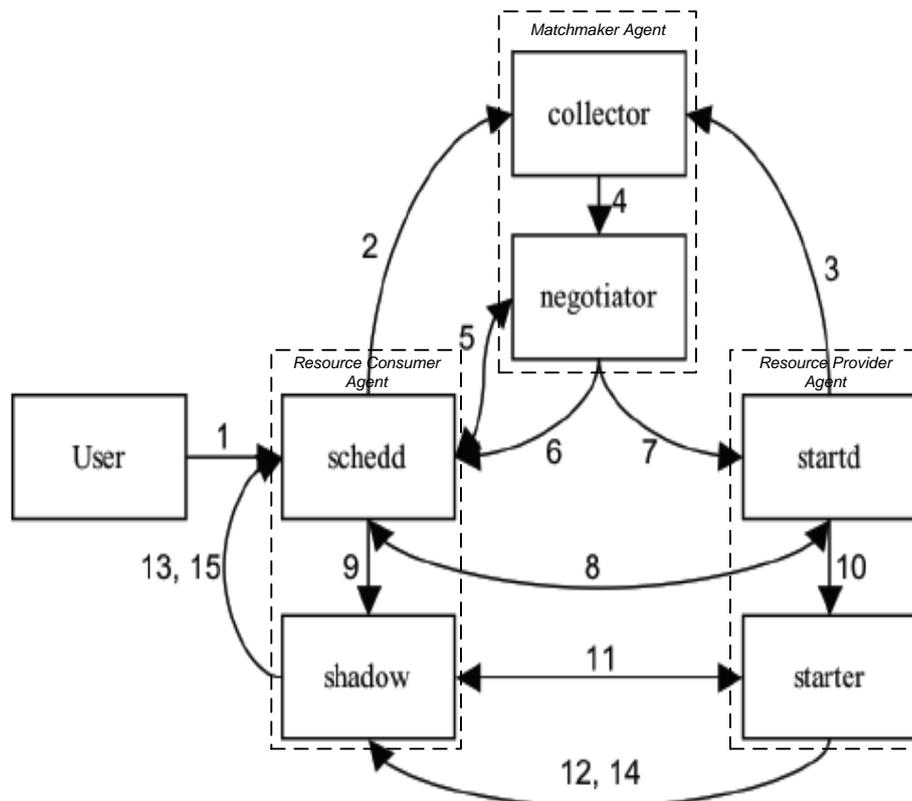


Figure 26: Communication between different Condor processes

The matchmaker agent consists of two separate processes – the *collector* and the *negotiator*. Similarly, the resource consumer and the resource provider agents are made up of two

separate processes each, namely, *schedd* and *shadow* in the case of the former, and *startd* and *starter* in the case of the latter. The interaction between the processes is shown in figure 26 and is described in table 16 below. Both the figure and the table are referenced from Robinson and DeWitt (2007).

Table 16: Interaction between different Condor processes

Step	Description
1	User submits job to <i>schedd</i> , <i>schedd</i> creates job in in-memory queue, logs job to disk
2	<i>Schedd</i> sends job queue summary to <i>collector</i>
3	<i>Startd</i> sends periodic heartbeat to <i>collector</i>
4	<i>Collector</i> forwards job, machine data to <i>negotiator</i> for scheduling algorithm
5	<i>Negotiator</i> contacts <i>schedd</i> for job-specific information, <i>schedd</i> sends job data to <i>negotiator</i>
6	<i>Negotiator</i> informs <i>schedd</i> of job-machine match
7	<i>Negotiator</i> informs <i>startd</i> of job-machine match
8	<i>Schedd</i> , contacts <i>startd</i> to confirm match
9	<i>Schedd</i> spawns <i>shadow</i> to monitor job progress
10	<i>Startd</i> spawns <i>starter</i> to start up, monitor job
11	<i>Shadow</i> , <i>starter</i> establish socket connection to exchange job state information
12	<i>Starter</i> sends <i>shadow</i> periodic job state update messages
13	<i>Shadow</i> forwards job update messages to <i>schedd</i>
14	<i>Starter</i> notifies <i>shadow</i> when job completes, exits
15	<i>Shadow</i> exits, <i>schedd</i> captures exit code, removes job from queue

The Condor agents generally run on multiple machines over the Condor pool (with the exception of matchmaking agent that runs on only one computer), and therefore the interactions between the agents is through Socket communication. Condor uses multiple static ports (ports that are opened on usually known port numbers, for example, matchmaking agent uses port numbers 9614 and 9618), multiple dynamic ports (ports that are opened at randomly chosen port numbers from a particular port range, for example, Condor uses all valid port numbers above 1023 for dynamic port assignment), relies on bi-directional (many-to-many) pattern of communication between machines and uses both TCP and UDP ports

(Beckles et al, 2005). This discussion (Condor's reliance on multiple, bi-directional, static and dynamic ports) will be referenced later in the thesis.

The architecture of Condor allows jobs from different users to be executed simultaneously over one Condor Pool. Furthermore, these jobs can be standalone jobs requiring only one computer to process it, or they can be MPI or PVM-based parallel jobs requiring concurrent access to multiple resources. The sections below examine key Condor concepts that can help provide higher-level grid services to the CSPs.

2.10.2 Condor universe

Condor universe is an execution environment for jobs that are submitted by the users. Depending upon the type of job to be executed and its requirements, the user needs to select from among the following Condor universes (Condor Version 6.9.1 Manual, 2007b):

- Standard universe
- Vanilla universe
- Java universe
- PVM universe
- Parallel universe
- Grid universe
- Scheduler universe
- Local universe

Standard universe provides support for *checkpoint* and *migration* of user jobs. To run jobs that can use the standard universe the program to be executed has to be recompiled with the Condor libraries using the *condor_compile* program. This allows Condor to transparently save the current state of the running job at periodic intervals. If the resource on which the job is currently running becomes busy, the job is migrated to another resource along with the checkpoint file. Thus the program is restarted from the previous checkpoint state. Standard universe also supports *remote system calls* which permit remote resources to access files in the job submission machine. Although the support for checkpoint and migration might be useful for running large CSP-based simulations on non-dedicated resources, standard universe will not be discussed further as it is not currently supported in Windows (Condor Version 6.9.1 Manual, 2007a).

Vanilla universe is for executing programs that cannot be re-linked with Condor libraries. It does not support checkpoint, migration or remote system calls. Therefore, when a resource becomes busy, the currently executing job will either have to be suspended for later execution (until the time the resource becomes idle again), or the job terminated and resumed on a different host. Because vanilla universe does not support remote system calls, the access to files is through the use of a network file system (NFS) or using the Condor file transfer mechanism (FTM).

Java universe supports the execution of java programs using the Java Virtual Machine (JVM) execution environment. The JVM itself is not included with the Condor installation package and it will have to be separately installed. The command `condor_status -java` lists the JVM vendor and the JVM version information for each resource in the Condor pool. For example, as can be seen in screenshot 4 below, resource 217-H is the only PC that is running JVM version 1.5.0. All the other PCs are running JVM version 1.4.2. Unlike vanilla universe where the user jobs usually consist of executable files (.exe) that can be run natively by the Operating System, Java universe jobs comprise of .class and .jar files that are executed through the JVM. However, like vanilla universe, Java universe does not support checkpoint, migration or remote system calls, and employs either NFS or Condor FTM for file transfers. Since both of these universes have much in common, and because Java is platform-neutral, open source, and is widely used in the industry, only Java universe will be discussed again in later sections. Although it may be possible to execute a java job in the standard or vanilla universe, it would be a waste of network resources because it would involve the transfer of the entire JVM binary and the standard Java libraries to each resource (Thain et al, 2004).

```
E:\>condor_status -java
```

Name	JavaVendor	Ver	State	Activity	LoadAv	Mem	ActvtyTime
217-H	Sun Microsy	1.5.0	Unclaimed	Idle	0.000	255 [?????]	
210-A	Sun Microsy	1.4.2	Unclaimed	Idle	2.050	1023	0+00:00:57
211-B	Sun Microsy	1.4.2	Claimed	Busy	3.200	1023	0+00:00:21
212-C	Sun Microsy	1.4.2	Unclaimed	Idle	0.000	1023	0+00:00:18
213-D	Sun Microsy	1.4.2	Unclaimed	Idle	0.020	1023	0+00:00:10
214-E	Sun Microsy	1.4.2	Claimed	Busy	1.260	255 [?????]	
215-F	Sun Microsy	1.4.2	Unclaimed	Idle	0.030	255 [?????]	

Screenshot 4: JVM related information output using Condor command “condor_status”

PVM universe supports the execution of parallel programs written for the Parallel Virtual Machine (PVM) environment (Geist et. al, 1994). PVM provides a set of software tools and message passing libraries that enable parallelism at program level by allowing parallel computation of spawned processes on multiple computers. However, PVM universe is not currently supported on Windows (Condor Version 6.9.1 Manual, 2007a) and will be excluded from further discussion.

Parallel universe provides an execution environment for parallel jobs. It has superseded the Condor MPI universe as it not only provides support for programs written using the MPI standard, but also other parallel programming environments and different MPI implementations like MPICH2, Open MPI, etc. Parallel universe is supported in the Windows platform but requires installation of parallel programming libraries on the different PCs that make up the Condor pool. It appears from this discussion that Condor parallel universe execution environment can provide the grid-facilitated parallel computation service to the CSPs.

Grid universe enables a user to submit jobs to various grid job management systems using the standard Condor job submission interface. Thus, grid universe jobs can be submitted to grid resources running the Globus middleware (referred to as GT-2, GT-3 and GT-4 grid types or simply as Condor-G) or the UNICORE middleware (referred to as Unicore grid type); they can be submitted to clusters running the PBS batch system (PBS grid type) or the LSF batch system (LSF grid type) or jobs can be submitted to another Condor system itself (Condor grid type [Condor-C]). Of these, only the Condor grid type is presently supported in Windows (Condor Version 6.9.1 Manual, 2007a). Condor-C makes it possible for users to transfer jobs between different condor resources that may or may not be a part of the same Condor pool. If the resources are not part of the same pool then Condor-C utilizes the Condor flocking mechanism that allows two or more Condor pools to be linked together. Transfer of jobs between queues is a functionality that does not directly map to any of the six higher-level grid services that have been identified in this chapter. As such, it will not be discussed any further in this research.

Scheduler universe and **Local universe** allow jobs to be executed immediately on the resource on which the job is submitted. Thus, there is no need for matchmaking with remote resources. Another feature of both these universes is that jobs are never pre-empted. Local universe provides better job management features compared to the scheduler universe and should normally be used when executing jobs on the submit machine. These universes will not be discussed any further because they only support program execution on one machine. This research, on the other hand, is based on the assumption that multiple computers are available for grid-enabled CSP-based simulation.

The discussions in this section have shown that, of the 8 Condor universes only Java universe and parallel universe merit further investigation for the purposes of this research. The next section looks at the job submission mechanism for Condor.

2.10.3 Condor job submission mechanism

There are four steps for running jobs under Condor – (1) code preparation, (2) selection of Condor universe, (3) creation of submit description file and, finally, (4) job submission (Condor Version 6.9.1 Manual, 2007b).

Code preparation: A Condor job consists of user executables and associated data. It is run unattended in the background and is unable to interact with the users. However, a limited degree of interaction may be possible through files that contain proper program inputs. The console output generated while running the job are directed to files. In the code preparation stage the program may have to be modified to support its execution along these lines.

Selection of Condor universe: The next step involves selection of an appropriate Condor universe. This selection is based on the requirements of the job. For example, Condor java

universe will have to be selected if the user intends to run a Java job. Depending upon the universe chosen, the program may have to be recompiled with Condor libraries. For example, to use standard universe the user program will have to be recompiled with the *condor_compile* command.

Creation of submit description file: The third step involves the creation of a submit description file (*.sub*). Every Condor job has a corresponding *.sub* file that controls the details of the job submission through different Condor-defined variables. Examples of a few of these variables and their purpose are given below.

- *executable*: Informs Condor which program to run (*executable* = HelloWorld.class).
- *arguments*: The command line arguments for a program (*arguments* = HelloWorld Hi)
- *universe*: The runtime environment to use (*universe* = Java).
- *input*: The filename containing keystrokes that emulate interactive program input (*input* = inputfilename.txt).
- *output*: Console output during program execution will be redirected to this file (*output*=outputfilename.txt).
- *log*: Messages generated by Condor will be written to this file (*log*=logfile.log)
- *queue*: The value assigned to the queue variable will determine the number of replications of a single job to run (*queue* = 10). In the case of a CSP-based simulation experiment, for example, the value of *queue*=10 will mean that the experiment is executed 10 times over the available grid nodes.
- *transfer_input_files*: The files that are to be transferred to the execution directory of a resource (*transfer_input_files* = ..\AsianStockOption.class, ..\jacob.jar)
- *should_transfer_file*: Whether files are to be transferred to a resource (*should_transfer_file* = yes)

An example of a submit description file is shown later in section 5.5.

Job submission: The fourth and final stage involves the submission of a job using the *condor_submit* command. The argument to this command is the name of the submit description file. Once submitted, the progress of the job can be monitored through the *condor_q* command. This command shows the jobs that are either running or idle (i.e., in queue), the job number, the job owner, the time the job was submitted, etc. A job can also be removed from the queue prior to its execution by using the command *condor_rm*. The argument to this command will specify the job that has to be marked for removal. The status of the Condor pool can be determined using *condor_status*. The output from this command will list the machines currently in the Condor pool, their hardware configurations (CPU and memory), their activity status (busy or idle), etc. Screenshots showing the output of these commands are included in section 5.5.

The next section gives an overview of a Condor component – Condor DAGMan – that can be potentially used to provide the CSP-specific workflow service.

2.10.4 Condor DAGMan

Condor Directed Acyclic Graph Manager (DAGMan) is a workflow management system. It is a meta-scheduler for Condor that operates at a higher-level than the Condor scheduler and manages dependencies between jobs (Condor DAGMan, 2007). The job workflow is represented using the DAG data structure which shows jobs as vertices in the graph. The directed lines that connect these vertices are called the graph edges and they provide the direction of the work flow. For example, a “diamond” DAG (figure 27) represents the direction to the flow of work between four jobs wherein Job A has to be executed first followed by simultaneous execution of Job B and Job C, and finally Job D (Frey, 2002). The diamond DAG can be defined by a *.dag* file as follows:

```
# DAG file
Job A a.sub
Job B b.sub
Job C c.sub
Job D d.sub
Parent A Child B C
Parent B C Child D
```

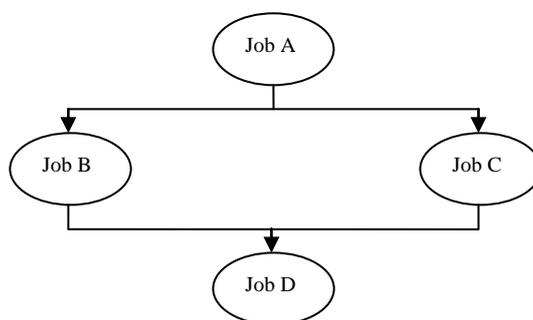


Figure 27: Graphical representation of diamond DAG (Frey, 2002)

In the DAG file each job has a placeholder and lists the accompanying Condor *.sub* job submit file. For example, the job defined in *a.sub* is given a placeholder *Job A*. The *.dag* file represents the direction of work flow between the defined placeholders using *parent* and *child* relationships. Once this *.dag* file is submitted to Condor DAGMan (using command *condor_submit_dag*) it interacts with the Condor Scheduler to submit jobs to the Condor job queue based on the outlined job dependencies.

It appears from the discussion above that Condor DAGMan can provide the grid-facilitated higher-level workflow service to CSPs. Thus, if there are multiple models to be executed, such that the results of one simulation will serve as input for the others, or if there is need to

export simulation result data to another application for further processing, then Condor DAGMan can be potentially used to automate the job execution workflow.

2.10.5 Condor MW

Condor has a *MW (Master Worker) software library* that enables users to create master-worker type applications. This C++ library consists of a set of source files that need to be compiled with a user application before the Condor system can be used for the master-worker type computations. To do this, the user application imports the MW library, subclasses three specific MW classes (*MWTask*, *MWDriver* and *MWWorker*) with application specific code, and compiles the application (Condor MW, 2005). The compiled MW-application code uses Condor's resource management system to find idle machines through matchmaking, to assign computations, to monitor resources, etc. Thus, Condor MW uses the "broker-based" job scheduling mechanism of Condor.

The *MWTask* represents the basic job unit and describes the inputs and outputs that are associated with it. *MWTask* is processed by the *MWWorker* process on allocated resources. The *MWDriver* corresponds to a master process and manages the whole computation. It creates instances of *MWTask*, sends the tasks over to multiple *MWWorker* for processing, retrieves and collates the results of the tasks and finally, decides when the computation is over. Figure 28 below shows how a master worker type computation is performed using Condor MW. The figure has been adapted from Condor MW (2007).

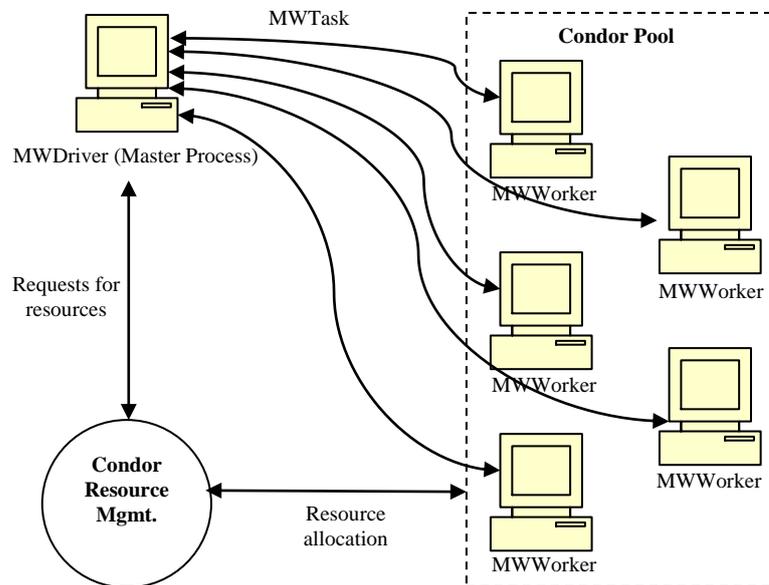


Figure 28: Processing job using Condor MW

Condor MW supports both *task-parallel* and *job-parallel* applications. In a task-parallel application a single process acts as the master and is responsible for directing and coordinating the computations being executed on the workers. A job-parallel application, on the other hand, obtains parallelism through one application (or user) submitting many jobs

(each job is a combination of executable code and associated data) to the Condor scheduler and being responsible for the detection of job completion. In the case of task-parallel applications using Condor MW, the master co-ordinates with the resource consumer agent (see figure 25) to request resources and receives resource allocation and de-allocation notifications (a resource is de-allocated when the job is completed). In the case of a job-parallel application using Condor MW, the application (or user) uses the standard Condor system commands to submit jobs and reads the log files using Condor-provided APIs to determine when a job is complete (Basney et al., 1999). It has to be added, however, that although task-parallel applications and job-parallel applications are both referred to as task farming (in this thesis and in some papers), the widely accepted definition of task farming applies mostly to task-parallel applications alone. This definition can be summarized as follows. The task farming application consists of one master entity and multiple worker entities, wherein the master entity decomposes the problem into small tasks, distributes these tasks among multiple worker processes and gathers the partial results to produce the final result of the computation; and the worker entities receive messages from the master with the next task, process the task and send back the results to the master (Heymann et al., 2000).

The discussions in this section have shown that Condor MW can potentially provide grid-facilitated task farming service to the CSPs. Furthermore, such service can be provided for both CSP-specific task-parallel applications and job-parallel applications. In this thesis Condor's support for job-parallel applications will be examined in the context of Condor Java universe, using standard Condor job submission and monitoring mechanisms (section 5.5).

2.10.6 Section summary

This section of the thesis has examined the EDGC middleware Condor. It has identified Condor parallel universe execution environment, Condor DAGMan and Condor MW as potential Condor-specific mechanisms that could provide grid-facilitated parallel computation service, workflow service and task farming services to the CSPs. Condor Java universe has also been identified as the potential execution environment for Java based applications. The next section presents three different approaches to using CSPs together with grid computing middleware.

2.11 Different approaches to using CSPs with desktop grids

For desktop grids to support CSP-based simulation it should take into account that users are specialists in simulation modelling (and not distributed computing) and any technological solution must be developed with little or no change to the CSP. Three possible approaches for using desktop grids with unmodified CSPs are discussed next. These are referred to as the *CSP-middleware integration approach*, the *CSP-runtime installation approach* and the *CSP-preinstalled approach*.

2.11.1 CSP-middleware integration approach

One possible way of using desktop grid middleware together with CSPs is to “bundle” the latter along with the former. When a desktop grid middleware is installed on a PC, the CSP is also installed on it. In an enterprise-wide desktop grid the jobs from other users (guest processes) may run alongside the programs being executed by the resource owner (host processes). However, the guest processes are usually run in a “sandbox” that is implemented by the middleware. This provides a logically separate and secure execution environment for both the host and guest processes. In Entropia DCGrid for example, the sandbox mechanism is called the *Entropia Virtual Machine (EVM)* and it wraps interpreters like cmd.exe, Perl and Java Virtual Machine (JVM) to prevent unauthorized access to a computer (Calder, 2005). Thus, it might be possible to include a CSP installation inside the EVM and offer it as part of an Entropia installation. The problem with this approach is that it will require changes to the enterprise desktop grid middleware as a CSP will have to be integrated with it. Furthermore, an enterprise desktop grid is a general purpose distributed computing environment that allows the execution of various user applications (not limited to simulation alone). Although the integration of interpreters like JVM can be justified because of the wide prevalence of Java applications, it is arguably more difficult to explain the inclusion of a CSP (but which CSP? there are at least 45 of them), unless a customized desktop grid middleware distribution is created for meeting simulation requirements of a specific organization. This approach is not considered appropriate for this research.

2.11.2 CSP-runtime installation approach

The second approach involves the installation of a CSP package at runtime, i.e. just before the simulation experiment is conducted. BOINC-RAC, discussed in section 2.9.3, is an example of this approach. In this case the CSP itself is transferred to the desktop grid nodes, along with the data files associated with the simulation and the trigger code. This approach is not feasible for a number of reasons. (1) the size of CSPs frequently exceed 100s of MBs and it may not be feasible to transfer such large amounts of data to multiple clients over the network, (2) the CSP will first need to be installed on the desktop grid node before the simulation can start, (3) such an installation is normally an interactive process and requires human intervention, (4) an installation normally requires administrative privileges on the client computers, (5) transferring CSPs may lead to a violation of the software licence agreement that may be in place between the CSP vendor and the organization (if the number of desktop grid nodes executing simulations exceed the number of licences purchased). This approach is therefore not considered appropriate for this research.

2.11.3 CSP-preinstalled approach

The third solution is to install the CSP in the desktop grid resource, just like any other application is installed on a PC. BOINC-PAC, discussed in section 2.9.3, is an example of this approach. The drawback with this approach is that the sandbox security mechanism implemented by most enterprise desktop grids may have to be forfeited. However, as

simulations are created by trusted employees running trusted software within the bounds of a fire-walled network, security in this open access scheme could be argued as being irrelevant (i.e. if it were an issue then it is an issue with the wider security system and not the desktop grid). The CSP-preinstalled approach is considered appropriate for using CSPs with desktop grids and will be perused further in this research.

The procedure to execute CSP-based simulation experiments over desktop grids following the CSP-preinstalled approach is as follows:

- The simulation user programs an executable “trigger” code in C++, Java, Visual Basic (VB), etc. that accesses the CSP functionality through exposed interfaces. CSPs that expose package functionality have been listed earlier in table 10. The trigger code should generally invoke the CSP, load the model file, transfer experiment parameters into the model, execute the model, etc.
- The simulation user makes available the data files associated with the simulation (simulation model files, experiment parameter files, etc.) and the executable file containing the trigger code to the desktop grid nodes where the experiment will be executed. Two possible ways of doing this is through a shared grid access to a network drive, or by transferring the required files using the desktop grid middleware. The experiment parameters can also be sent from the user node through Socket communication.
- The desktop grid middleware invokes the executable trigger code on a remote desktop node. The simulation starts and results are saved in a file. The user accesses the simulation results from the shared network drive, or the files are transferred back to the user. Alternatively, the results can also be sent across to the user over the desktop grid through Sockets.

2.12 Chapter summary

The purpose of this chapter was to investigate what grid computing has to offer to CSP-based simulation in industry. Towards this aim, a literature review on grid computing was conducted in section 2.2. Two important outcomes of this review were, (1) identification of different higher-level grid services that could be provided through use of grid middleware (e.g., parallel computation service, task farming service, computation steering service, etc.) and (2) identification of different forms of grid computing (e.g., cluster-based grid computing, EDGC, PRC and P2P).

This chapter then presented an overview of simulation in industry (section 2.3) and the tools (CSPs) that are used to build and run these simulations (section 2.4). It defined CSPs to include packages that support both discrete event simulation (DES CSPs) and Monte Carlo

simulation (MCS CSPs), as both these forms of simulation are extremely popular in industry. The widespread availability of Windows-based CSPs was also highlighted.

Section 2.5 then focussed on four higher-level grid services (identified earlier in section 2.2) that could be potentially used together with CSPs. The four services that were discussed were parallel computation service, task farming service, workflow service and collaboration service. The DES and MCS CSPs were assessed in relation to the four services in order to investigate the degree to which the CSPs support such functionality through custom implementations. In most cases this support was been found to be extremely limited.

Sections 2.6 and 2.7 then discussed two specific forms of simulation, namely, distributed simulation and web-based simulation, which could potentially benefit from use of grid computing. Two new grid-facilitated higher level services that were specific to distributed simulation and web-based simulation were identified. These services were named distributed simulation service and web-based simulation service respectively.

Section 2.8 then investigated the form of grid computing that was suitable for use with CSPs. Informed by the discussion on grid middleware in section 2.2, it was found that cluster-based grid computing was generally unsuitable for CSP-based simulation because it was mainly targeted at UNIX and Linux systems and the CSPs were predominantly Windows-based. It identified other forms of grid computing, notably PRC and EDGC, that is supported on Windows-based PCs to be more appropriate for CSP-based simulation in industry. Sections 2.9 and 2.10 then discussed two representative middleware for PRC and EDGC forms of grid computing, namely BOINC and Condor. Finally, section 2.11 presented three different approaches to using CSPs with desktop grid middleware and identified one of them (CSP-preinstalled approach) to be the most appropriate.

Based on the six higher-level grid services that were identified for use with CSPs in this chapter (parallel simulation service, task farming service, workflow service, collaboration service, distributed simulation service and web-based simulation service), the next chapter proposes a grid computing framework with the purpose of undertaking an organized study on how grid computing could further the practise of CSP-based simulation in industry.

3 PROPOSING THE CSP-GC FRAMEWORK

3.1 Introduction

Chapter 2 has provided the context to the research hypothesis that CSP-based simulation in industry can benefit from grid computing. It has identified six higher-level grid services and has found Windows-based grid computing middleware, specifically middleware for PRC and EDGC, to be suitable for use with the CSPs. The overview chapter also presented three possible approaches to using the CSPs with grid computing middleware. Among the three approaches, one approach, viz., the CSP-preinstalled approach, was considered appropriate for this research. Before continuing further the reader is reminded that, unless explicitly stated, the terms “desktop grid computing”, “desktop grids”, “grid computing” and “grids” are being used synonymously to refer to both PRC and EDGC.

This chapter proposes the COTS Simulation Package – Grid Computing (CSP-GC) framework for evaluation of the hypothesis (section 3.2). The framework is based on the higher-level grid services that have been identified for potential use with CSPs in chapter 2. Each higher-level grid service is referred to as a *grid-facilitated CSP-specific service* in the CSP-GC framework because the purpose of the framework is to investigate how grid computing can provide support (through *grid-facilitated services*) to existing CSPs. Section 3.3 discusses the implementation aspects of the six CSP-specific services in the grid context (section 3.3).

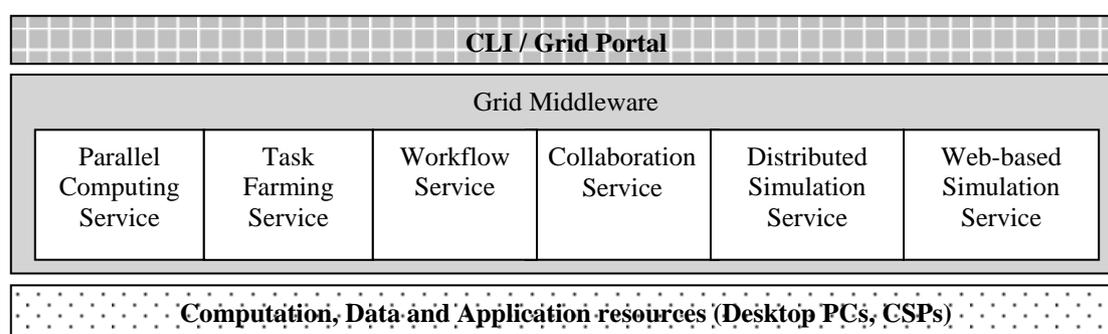
This chapter then examines the PRC middleware BOINC and the EDGC middleware Condor in relation to each of the six CSP-specific services in order to establish their suitability for use with the CSPs (section 3.4). This is followed by a general discussion on the suitability of BOINC and Condor for grid-enabling CSP-based simulations (section 3.5). The chapter concludes by recognising the need for a Windows-based grid computing middleware for use in industry that uses the “push” based job scheduling mechanism, supports Java-based applications and is suitable for deployment in an organization that has network security restrictions in place (section 3.7).

3.2 The CSP-GC Framework

This section proposes the CSP-GC framework to investigate how grid computing can advance the practice of simulation in industry. The CSP-GC framework provides a logical structure for the evaluation of the hypothesis presented in this thesis by organizing the possible uses of grid computing for CSP-based simulation into six distinct *grid-facilitated CSP-specific services*. Each CSP-specific service is a potential application of grids technology for CSP-based simulation and is derived from one of the six higher-level grid services that have been identified in the previous chapter. The six CSP-specific services that are presented in this framework are parallel computing service, task farming service, workflow

service, collaboration service, distributed simulation service and web-based simulation service. The CSP-GC framework is shown in figure 29 below. The service descriptions of the six CSP-GC framework defined services are presented in table 17.

The CSP-GC framework shows that the CSP-specific grid services utilize the basic grid services like computation service, data service, application service, etc., and the core grid mechanisms like authentication and authorization, resource discovery, resource allocation, etc., that are usually provided by the grid middleware. The reader is referred to section 2.2.1 for a discussion on the basic grid services and the core grid mechanisms. The grid middleware, in turn, makes use of enterprise resources like desktop PCs, corporate Intranet and DES and MCS CSPs, to provide the underlying hardware, network and software infrastructure required to support a desktop grid. The grid middleware can be accessed using middleware-specific Command Line Interface (CLI) commands or, optionally, through a grid portal.



Legend

	Access to grid middleware		Enterprise computing resources
	Basic grid services (computation service, data service, etc.) and core grid mechanisms (resource discovery, job submission, job scheduling, job monitoring, etc.) provided by grid middleware		CSP-specific services that can <i>potentially</i> be provided through the use of grid computing

Figure 29: The CSP-GC framework

Table 17: CSP-GC framework defined services and their descriptions

CSP-GC framework defined services	Service description
Parallel computing service	Parallel computing service can potentially speed up the execution of a single CSP-based DES or MCS using multiple processors. The grid middleware should generally provide support for running parallel MPI/PVM applications. Further description of this service can be found in section 3.3.1.
Task farming service	Task farming service can potentially reduce the time taken to execute batch simulation experiments by distributing the execution of multiple CSP-based DES and MCS experiments over different grid nodes. This service supports concurrent execution of multiple instances of the same simulation model (SMMD task farming) or different simulation models (MMMD task farming). Further description of this service, including SMMD and MMMD variants of task farming, can be found in section 3.3.2.
Workflow service	Workflow service can potentially enable phased execution of different CSP-based DES/MCS models and other external applications based

CSP-GC framework defined services	Service description
	on the underlying data dependencies. Further description of this service can be found in section 3.3.3.
Collaboration service	Collaboration service can potentially facilitate collaboration among simulation practitioners by providing mechanisms which allow (1) reuse of DES/MCS model components among different users, (2) sharing of DES/MCS models for joint development and (3) virtual meetings. Further description of this service can be found in section 3.3.4.
Distributed simulation service	Distributed simulation service has the potential to execute DES CSP-based distributed simulation using the HLA-RTI middleware for distributed simulation. Further description of this service can be found in section 3.3.5.
Web-based simulation service	Through the use of grid portals, web-based simulation service can potentially provide simulation users with web-based access to DES and MCS CSPs for conducting simulation experiments. Furthermore, this service can potentially provide mechanisms to host “callable” web services that expose CSP interfaces. Further description of this service can be found in section 3.3.6.

The next section of the thesis examines the grid-facilitated CSP-specific services that are outlined by the CSP-GC framework in greater detail.

3.3 Grid-facilitated CSP-specific services

The CSP-GC framework has identified six CSP-specific services that can be potentially used together with the CSPs. Table 17 has presented service descriptions pertaining to each of the services. This section further examines these services in relation of its implementation requirements.

3.3.1 CSP-specific parallel computing service

Parallel computing is the concurrent use of multiple processors to solve a computational problem in the fastest possible time. Parallel computing service in the grid environment has the potential to speed up the execution of a single DES or MCS using multiple processors. The multiple processors taking part in such a computation may include shared-memory and distributed memory multiprocessor computers, network of workstations, etc. The form of grid computing that has been found suitable for grid-enabling CSP-based simulations is desktop grids. The computing infrastructures of such grids are generally made up of a network of workstations that do not have access to shared memory. It has been discussed earlier in section 2.5.1 that parallel programs in a distributed memory environment (like desktop grids) can be run using message passing mechanisms like the MPI and PVM. This generally requires that the grid middleware has support for MPI implementation like MPICH2 (Argonne National Laboratory, 2006) and / or PVM environment. Thus, for the desktop grid middleware to support the CSP-specific parallel computation service, it should ideally support execution of MPI/PVM-based parallel programs.

3.3.2 CSP-specific task farming service

Task farming service for CSPs has the potential to speed up DES or MCS experimentation using multiple distributed processors. In the context of this research, task farming is defined as the execution of multiple individual simulations on PCs that are connected through the

network. It is based on the master-worker distributed computing architecture. Unlike parallel computation service, the objective here is not to speed up the execution of one instance of a simulation but to utilize many computers to complete a set of simulation experiments faster.

In the context of a MCS the distinction between the parallel computation service and the task farming service is not very obvious. This is because a MCS run may require execution of the same model many thousands of times over, but with different random numbers. In such cases the number of Monte Carlo iterations can be distributed over a set of processors through the task farming service. This results in speeding up the execution of one MCS – the same objective as that of parallel computation service for MCS CSPs. Nevertheless there exists one key difference between them. In the case of a parallel MCS, the MCS CSP may spawn multiple child processes and use MPI / PVM messages to communicate with them. In the case of the task farming approach there are individual MCS CSPs running on each processor (but executing the same MCS code) and there exists one master process that has the task of distributing the Monte Carlo iterations to the individual CSPs and collating the results. The communication between the master process and the individual CSPs is through the underlying grid infrastructure. Thus, the task farming approach is based on the principles of master-worker (also known as master-slave) and two separate programs are involved, namely, the master program and the worker program (the CSP). The parallel computation approach consists of only one program (the CSP) that concurrently executes several processes that are spawned from it (Eltis and Komolkin, 2004).

The task farming service for CSPs can potentially support simultaneous execution of multiple sets of simulation experiments, wherein each set consists of one MCS or DES model with associated experiment parameters. For a MCS the experiment parameters can be the different values for simulation variables, the number of iterations that are to be performed, the random number seed to be used, etc. Similarly, for a DES the experiment parameters can consist of values for different model-defined variables like processing time for workstations, number of entities in the queue, model warm-up time, the simulation end time, etc.

3.3.2.1 Task farming scenarios

Two terminologies relating to task farming service for CSPs are now introduced – *Single Model Multiple Data (SMMD)* and *Multiple Model Multiple Data (MMMD)*. These terminologies are inspired from Michael Flynn's 1966 classification of very high speed computer architectures and parallel programming models.

Michael Flynn's 1966 classification: Michael Flynn has classified the computer architectures into *Single Instruction Stream-Single Data Stream (SISD)*, *Single Instruction Stream-Multiple Data Stream (SIMD)*, *Multiple Instruction Stream-Single Data Stream (MISD)* and *Multiple Instruction Stream-Multiple Data Stream (MIMD)* (Flynn, 1966). A computer with SISD architecture is a serial computer that executes one instruction on a single data stream

at any particular point in the program's execution (i.e., the *von Neumann computer* comprising of a single CPU that runs a series of instructions through a sequence of read and write operations on the memory). When different instruction sets are executed on multiple processors but access only one data stream then they can be termed as MISD machines. A SIMD machine has multiple processors that execute the same instruction in synchronization but on different data streams. Finally, a MIMD machine has multiple processors that execute different instruction sets on different data streams. This classification was done along two independent dimensions of Instruction and Data, where each dimension could have a state that was either Single or Multiple, and could be represented in the form of a matrix (Barney, 2006). This matrix is presented in table 18 below.

Table 18: Michael Flynn's classification of computer architectures

	Single Data	Multiple Data
Single Instruction	SISD	SIMD
Multiple Instruction	MISD	MIMD

Parallel programming models: Two parallel programming models, namely, *Single Program Multiple Data (SPMD)* and *Multiple Program Multiple Data (MPMD)* are frequently used for programming the MIMD multiple processor machines (Aubanel, 2000). A MIMD machine executing a SPMD parallel program will run a single program over multiple processors, but each processor will have access to multiple data. On the other hand, a MPMD program being executed on a MIMD machine will execute different program code on each processor and will access multiple data streams.

In the context of task farming for CSPs, CPUs of multiple PCs are harnessed together using grid middleware and used for cooperatively executing a set of CSP-based simulation experiments faster. The collection of these PCs can arguably be referred to as a MIMD machine because each PC can execute different instructions on different data sets. For CSP-based simulations the multiple instructions (MI) can be the different MCS or DES models that can be potentially executed in parallel on different computers, and the multiple data (MD) can refer to different experiment parameters for these models (this is subsequently referred to as *Multiple Model Multiple Data [MMMD] task farming*). Furthermore, like the SPMD parallel computing model, one single MCS or DES model having different experiment parameters can also be executed on a grid-based MIMD system (this is subsequently referred to as *Single Model Multiple Data (SMMD) task farming*). The use of SPMD and MPMD terminologies have not been considered appropriate to describe the two task farming scenarios (SMMD and MMMD) because of inherent differences in parallel computing and the master-worker based distributed computing.

It is worth considering the other variants of CSP-based task farming that may exist. In the case of MMMD task farming, the different models may belong to the same CSP or to different CSPs. These are referred to as *single CSP MMMD* and *multiple CSP MMMD* task farming

respectively. This thesis investigates only the single CSP MMMD. Thus, the concurrent execution of different simulation models, each having a separate set of experiment parameters, created using a single MCS or DES CSP will be examined. However, it is arguable that the master-worker architecture that supports single CSP MMMD task farming can equally support its multiple CSP variant.

Finally, it is possible to represent two other task farming scenarios by drawing a matrix similar to the one used for the classification of computer architecture (shown in table 18). This matrix is presented in table 19 below and shows a total of four task farming scenarios.

Table 19: Possible task farming scenarios with CSPs and desktop grids

	Single Data	Multiple Data
Single Model	Single Model Single Data (SMSD)	Single Model Multiple Data (SMMD)
Multiple Model	Multiple Model Single Data (MMSD)	Multiple Model Multiple Data (MMMD)

This thesis identifies SMSD task farming to be the serial execution of a single simulation model with one set of experiment parameters on one computer. This will not be examined any further because it is contrary to the objective of task farming which uses multiple computers. MMSD task farming is identified as the execution of multiple models that use the same set of experiment parameters over a grid. Again this will not be examined in this thesis because it is unlikely that two different models will have the same set of variables and use identical sets of experiment parameters.

3.3.3 CSP-specific workflow service

Grid-facilitated workflow service has the potential to logically link the execution of different CSPs and software applications that are available on the various grid resources. In the context of CSP-based simulation, workflows can be used, for example, to potentially enable phased execution of different CSP models that represent different parts of the supply chain. For grid computing to support workflow service, it should ideally be possible for the grid middleware to provide mechanisms to execute multiple programs in a phased manner over different grid nodes and to transfer the data generated by the programs amongst the nodes.

3.3.4 CSP-specific collaboration service

The term “collaboration” can be defined as the cooperation among different individuals to attain common goals. It can therefore be argued that all the six CSP-GC framework defined services involve some form of collaboration between the modellers because the desktop grid infrastructure being used for delivery of grid services is composed of the computing resources that are used by the modellers at their workplace. Thus, by making their resources available over the desktop grid, each user is contributing towards the overall goal of using grid computing technologies to support simulation at their workplace.

However, in this thesis, the CSP-specific collaboration service is derived from the grid facilitated higher-level collaboration service. Discussions in sections 2.2.2 and 2.5.4 have identified three potential uses of this service in the context of CSP-based simulation modelling, namely, (1) collaboration service can facilitate reuse of model components between different users (through search and download of model components), (2) it can facilitate sharing of CSP models (for joint development purposes), and (3) it can facilitate interaction between those involved in simulation studies (through virtual meeting support). These are subsequently referred to as three different forms of CSP-specific collaboration service. These three forms of collaboration service have also been recognised as potential application areas of simulation in a networked environment by Robinson (2005b). The different forms of collaboration service are discussed next in relation to the grid middleware support required to implement them.

Model reuse: Simulation model reuse will generally involve the “search and download” of model components for model building (Robinson, 2005b). Through user-developed web services, an OGSA-compliant grid middleware (like GT-4) can potentially provide the “search and download” support for existing CSP-model components that may be distributed over different grid resources. P2P grid computing middleware, generally used for “search and download” of multimedia files, can also potentially offer such services (the reader is reminded that the P2P form of grid computing is not discussed in this thesis because it generally does not allow the execution of user programs, like CSPs, over peer computers). For searching CSP models, an ontology-based semantic approach that utilizes web service discovery and deployment architecture has been proposed by Bell et al. (2006). This involves the creation of external descriptions for the CSP models using well-defined simulation ontology. This approach could possibly be used to search for models in the grid environment.

Sharing single model: It is arguable as to what extent grid computing can effectively support sharing of the same CSP models for joint development purposes. It may be possible to download copies of a model using user-defined web services, but synchronization of multiple copies of the same model will generally require package level support. CSP AnyLogic, for example, allows use of version control software to facilitate joint model development (see table 12). This research does not concern itself with CSP functionality that is implemented through custom solutions, and therefore this form of collaboration service falls outside the scope of this thesis.

Virtual meeting support: For grid computing to support virtual meetings it will generally be required that such middleware provide integrated support for audio, video, messaging, virtual whiteboards, etc.

In summary, the two forms of grid-facilitated collaboration service for CSPs that will be investigated further are (1) collaboration service that provides support for search and download of model components, and (2) collaboration service that provides support for virtual meetings.

3.3.5 CSP-specific distributed simulation service

Distributed simulation service only applies to DES CSP. A desktop grid middleware that provides distributed simulation support to DES CSPs should generally include mechanisms to enable synchronization of simulation time among different simulation models and to transfer messages between them. The message exchange by models running on multiple desktop grid hosts (henceforth referred to as peer-to-peer message passing) can be implemented in a centralized or a de-centralized manner. In centralized peer-to-peer message passing, one central component is responsible for receiving and sending messages from and to different hosts. When each host is responsible for communication with other hosts it is referred to as de-centralized peer-to-peer message passing. Grid computing middleware, such as BOINC and Condor, are not considered appropriate for enabling distributed simulation over a desktop grid because such solutions do not incorporate mechanisms for time synchronization and communication between individually running models (Lüthi and Großmann, 2001). The reasons for this are discussed below.

Time synchronization is outside the purview of grid middleware because these are general purpose programs that are designed to support a wide range of user applications, and the vast majority of applications do not require time synchronization mechanisms. Centralized and de-centralized peer-to-peer message passing is also outside the scope of most grid middleware because the focus is on executing serial applications over multiple computers. An exception to this is Condor PVM universe and parallel universe (discussed in section 2.10.2), which support parallel execution through message-passing mechanisms. However, none of these universes have inbuilt time synchronization mechanisms. A distributed simulation middleware may therefore have to be used along with a grid middleware to potentially enable distributed simulation of DES CSPs over the grid. The literature survey has shown that IEEE 1516 HLA standard is increasingly being used for distributed simulation in industry (section 2.6.3.1). As such, this research will discuss the grid-facilitated distributed simulation service with reference to HLA-RTI middleware for distributed simulation.

This thesis proposes two different approaches that could enable a grid middleware to employ the time synchronization and centralized peer-to-peer message passing services provided by HLA-RTI middleware to realize CSP-based distributed simulation. The first approach, referred to as the *middleware integration approach*, requires that a grid middleware communicate with the HLA-RTI middleware (HLA *rtiexec* process) using HLA-defined interfaces to manage the distributed simulation. The second approach, referred to as the *application integration approach*, proposes that the distributed simulation application (different CSP models and

associated code) be written such that they manage the simulation execution amongst themselves. Irrespective of the approach followed, the simulation applications themselves will interact with the HLA services for federation management, declaration management, time management, etc., by using the HLA-defined interfaces.

3.3.5.1 *Middleware integration approach to CSP-based distributed simulation*

The middleware integration approach will generally require modification to the grid software because it will now have to communicate with HLA-RTI to manage the CSP-based distributed simulation. Such communication may be possible through a *manager federate* that is invoked by the middleware and over which it exerts local control (figure 30). One advantage of this approach is that jobs can be migrated from busy nodes to idle nodes, thereby potentially speeding up the distributed simulation execution. Migration is possible because of two reasons. One, the grid middleware, together with the manager federate, is effectively the manager of the distributed simulation. Two, the grid middleware is aware of the status of the individual grid nodes and has mechanisms to schedule and monitor jobs.

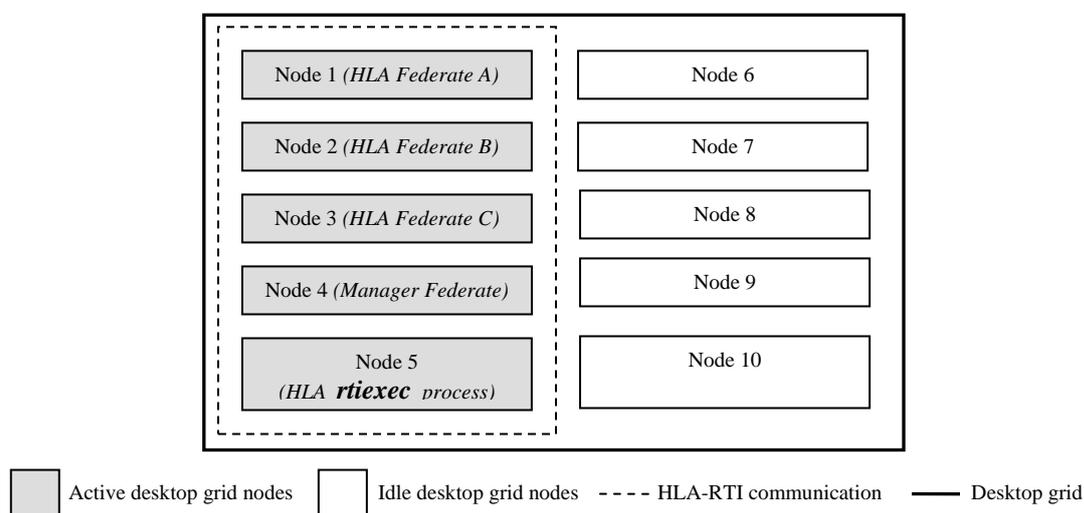


Figure 30: Middleware integration approach to providing distributed simulation service

3.3.5.2 *Application integration approach to CSP-based distributed simulation*

The application integration approach does not require any modification to the grid middleware itself. Here the distributed simulation application (consisting of the CSP models and associated code) has to manage the execution of the federation. The grid middleware is only responsible for allocating idle computing nodes over which the distributed models can be run. Thus job migration between nodes is not possible because the middleware no longer acts as the manager for the federation. The HLA *rtixec* process can be started as a different process on one of the nodes of the desktop grid or on another computer altogether (figure 31). The simulation federates can then communicate with the HLA *rtixec* process to advance time and to exchange messages between them. In this approach the grid middleware is unaware of *rtixec*-mediated peer-to-peer communication taking place between models that are being

executed over the grid. As no change to the desktop grid middleware is necessary, any grid middleware can be potentially used to implement the application integration approach.

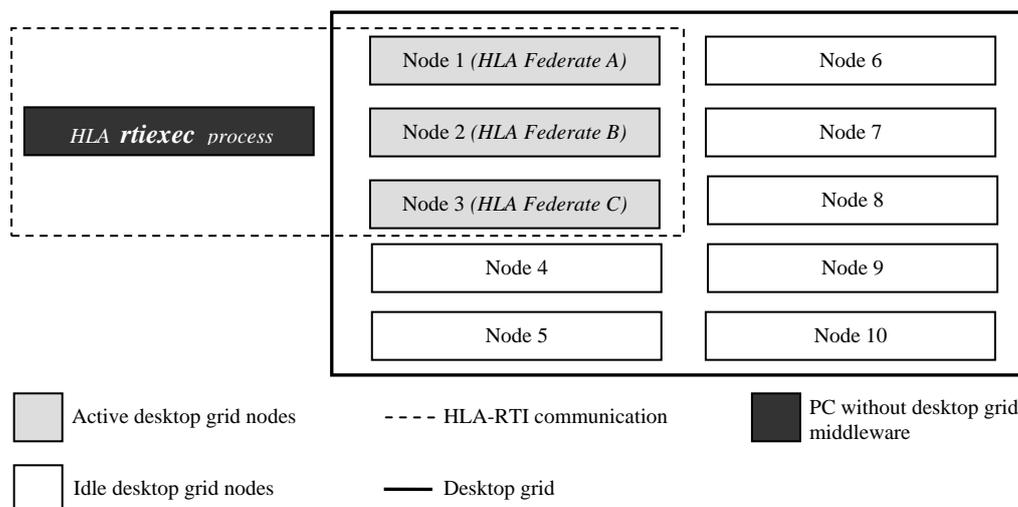


Figure 31: Application integration approach to providing distributed simulation service

3.3.6 CSP-specific Web-based simulation service

For the purpose of this research, web-based simulation is defined as simulation in a client-server environment that uses web-based technologies like web browsers, web servers, web services and Java applets, among others. In the context of CSPs it means that the simulation packages are accessible through web pages or through “callable” web services. It is usually possible to create a web-based front-end to a CSP application that exposes package functionality. An example of this has been shown with regards to DES CSP Witness in the earlier chapter (section 2.7.2). A simulation user who is able to access a package through a web browser will arguably not have a need to use grid-facilitated web-based simulation service. However, this service is only one among six potential CSP-specific services. If grid technology is adopted to support the other five services, then it is likely that web-based simulation service will also be used because it standardized the access to CSPs in a distributed environment.

Discussions in section 2.7.3 have identified two possible ways through which web-based simulation service could potentially support the CSPs, namely, (1) through use of grid portals and (2) through use of “callable” user-developed web services that expose CSP interfaces and which are hosted in web services containers provided grid middleware. These are subsequently referred to as two different forms of CSP-specific web based simulation service.

This section has discussed the six CSP-GC framework defined CSP-specific services in detail. Sections 2.9 and 2.10 have earlier presented a detailed discussion on BOINC and Condor with the objective of examining the underlying grid middleware mechanisms that can be potentially used to grid-enable the DES and the MCS CSPs. The next section examines BOINC and Condor in relation to each of the six CSP-specific services.

3.4 Investigation of CSP-specific services using BOINC and Condor

3.4.1 Investigation of parallel computation service

It has been discussed earlier in section 3.3.1 that the grid middleware that can potentially provide parallel computation service to CSPs will generally need a mechanism for inter-process communication between the simulation processes being executed in parallel over multiple PCs. This is usually only possible if the grid middleware has support for parallel computing environments such as PVM, MPICH2, Open MPI, etc.

BOINC middleware is designed for jobs that do not require any form of inter-process communication between executing processes. Here, lot of instances of the same computation are executed, but with different input parameters. As such, parallel computing environments like those discussed earlier are not supported by it. It can therefore be argued that BOINC will not be able to provide parallel computation service to the CSPs.

Condor provides two runtime environments – PVM universe and parallel universe – for running parallel programs that use message passing mechanisms for inter-process communications. Of these only parallel universe is supported on Windows environment. Thus, Condor parallel universe may be potentially able to provide parallel computation service to Windows-based CSPs.

3.4.2 Investigation of task farming service

BOINC and Condor MW support task-parallel applications, i.e., one master process directing the execution of several worker processes. In addition, Condor MW and Condor Java universe also support job-parallel applications, i.e., one process (or user) submits multiple jobs to a job scheduler (section 2.10.5 highlights some other differences between task-parallel and job-parallel applications). It is therefore considered likely that these middleware will also be able to execute the CSP-based SMMD and MMD task farming.

Task-parallel application execution over a desktop grid is generally based on the master-worker distributed computing architecture. BOINC implements the master-worker distributed computing architecture and uses the “pull” mechanism for scheduling jobs (subsequently referred to as “pull” based model of the master-worker architecture). Condor MW, on the other hand, implements this architecture but utilizes Condor’s “broker-based” job scheduling mechanism (subsequently referred to as “broker-based” model of the master-worker architecture).

The reader is reminded that Condor MW provides a C++ software library that has to be compiled with a user application before it can be executed over the Condor desktop grid. The compilation requirements of a MW application over Windows platform are as follows (Condor MW, 2005).

- The compiler needed is any C++ compiler that is compatible with the compiler that built the *libcondorapi.a* library. This library is a part of Condor. Presently MW has only been tested with G++ compiler version 3.3.4 over Windows XP.
- Compilation of user application with MW library is via *Cygwin*. Cygwin provides a Linux-like environment for Windows using a DLL (*cygwin1.dll*) that acts as a Linux API emulator layer and provides Linux API functionality (Cygwin, 2007).

Using Condor MW for implementing task-parallel applications in a Windows environment has some drawbacks.

- Condor MW has a *Resource Management and Communication* (RMComm) component that is responsible for communication between master and workers. There are different RMComm implementations like CondorPvm, Files, Sockets, etc. In a Windows environment the user application generally has to be compiled with MW library through the Cygwin environment. Compilation of the MW application using Cygwin suggests that the RMComm will use POSIX system calls during execution. POSIX or Portable Operating System Interface for uniX is an IEEE 1003 standard that describes standard interfaces to the Unix operating system and its different variants (Walli, 1995). Thus, every Windows machine over which a MW application will be run may require access to Cygwin. This can be done in several ways, for example, (1) using Condor's file transfer mechanism *cygwin1.dll* can be temporarily transferred to the machines running the MW application; (2) If the Windows machines have access to a Network File System then *cygwin.dll* may be placed in the network share; (3) Cygwin can be installed on all the Windows machines in the Condor pool.
- The user application should be a C++ application as it requires recompilation with the C++ MW library. Thus, a Java-based application will not generally be able to use the MW library to implement Java-based task parallel solutions over Condor.

In summary, it can be argued that BOINC, Condor MW and Condor Java universe can potentially support CSP-specific task farming services.

3.4.3 Investigation of workflow service

It has been discussed earlier in section 3.3.3 that for a grid middleware to support workflow service it should ideally provide mechanisms to execute multiple programs in a phased manner over different grid nodes and transfer data between them. Investigation of BOINC middleware has shown that BOINC projects usually consist of only one executable (sometimes with multiple versions). For running multiple programs, therefore, different BOINC projects may have to be created. In a workflow there is usually a dependency between executing programs, for example, the data output from one program can be the input to a subsequent program. It is very unlikely that such dependencies can be maintained when using BOINC across projects because of the following reason:

- Every BOINC project has its own application and relational database. The database stores descriptions of applications, workunits, results, user information, etc. Furthermore, every project also has its own scheduling servers and data servers (Anderson, 2004).

Implementation of application workflows using BOINC does not seem feasible because it may necessitate communication between project specific scheduling and data servers. BOINC does not presently support such inter-project communication and it is therefore not considered suitable for providing workflow support to CSPs.

The discussion of Condor DAGMan meta-scheduler in section 2.10.4 has shown that it has been designed to manage dependencies between jobs. The workflow itself has to be defined in a *.dag* file. DAGMan reads this file and submits jobs to Condor in a phased manner based on the underlying job dependencies. Condor DAGMan can therefore be potentially used along with Condor's Java Universe execution environment to provide CSP-specific workflow service.

3.4.4 Investigation of collaboration service

Support for (1) search and download of CSP-based model components and (2) support for virtual meetings are the two forms of collaboration service that have been identified for further investigation in section 3.3.4. The reader is reminded that the search and download of model components may be possible through the use of web services that are hosted by the grid middleware, and virtual meetings would generally require integrated middleware support for audio, video, etc. The discussions on BOINC and Condor in sections 2.9 and 2.10 have shown that none of these facilities are supported by the middleware, and consequently it can be argued that neither of the middleware can provide CSP-based collaboration service.

3.4.5 Investigation of distributed simulation service

Section 3.3.5 has presented two approaches that could be used for HLA-based distributed simulation using DES CSPs in the grid environment. The first approach is the middleware integration approach which proposes that the manager component, responsible for controlling the distributed simulation, be integrated with the grid middleware. BOINC provides source code access and it may be possible to use the "middleware integration approach". But clearly, source code modification and recompilation of a general purpose desktop grid middleware is not a trivial task. Furthermore, a simulation practitioner in industry cannot be expected to have the distributed systems expertise required to implement such a middleware-integrated solution. This approach is therefore not considered appropriate for providing distributed simulation service and will not be discussed any further.

The second approach is referred to as the "application integration approach" that does not propose any modification in the grid middleware itself. Here the responsibility of managing a distributed simulation federation rests with the application (consisting of the CSP models and

associated code). As no changes to the grid middleware are necessary, it can be argued that both BOINC and Condor can be potentially used to provide distributed simulation service to DES CSPs. The following two sections discuss how BOINC and Condor could be used for providing this service.

3.4.5.1 Investigation of distributed simulation service using BOINC

A BOINC project usually consists of a single executable file (henceforth referred to as BOINC proxy application client or BOINC-PAC for short) that is transferred to the BOINC middleware (also referred to as the core client) running on a client computer when it first attaches itself to a project. The BOINC-PAC dependencies such as project initialization files, library files, DLLs, etc. are also passed along with the executable. The core client periodically downloads BOINC workunits from the project servers. These workunits generally provide input parameters or data to the BOINC-PAC application for processing. In the context of CSP-based simulation, the BOINC-PAC could be an executable file that invokes a CSP and loads a simulation model that has been downloaded by the BOINC core client. The work units can provide different simulation parameters (e.g., processing time for work stations, queue length) that are to be loaded into the model before running it. The results of the simulation can then be written into text files for transfer back to the BOINC project servers. Interaction between BOINC-PAC and the CSPs could be through the package interfaces that are exposed by the latter.

A distributed simulation requires the synchronized execution of two or more DES models. For BOINC to be able to support distributed simulation, the BOINC-PAC downloaded from the project server should generally be able to execute different models on different computers. For example, if a distributed simulation consists of model-A and model-B, then both models are downloaded to client computers (say, computer X and computer Y) as presently there appears to be no mechanism to transfer selective files to different core clients. For BOINC-based distributed simulation to begin, the BOINC-PAC in computers X and Y have to be told to execute either model-A or model-B. For obvious reasons both X and Y cannot execute the same model. The investigation of BOINC middleware has shown that workunits can pass different variable values to BOINC-PAC. Thus, if two BOINC workunits are created (for model-A and model-B) then the core clients running on computers X and Y will generally download one workunit each and execute either model-A or model-B. The variable values passed along with the workunit will determine which model is executed over which computer. This will, in turn, start the CSP-based distributed simulation over BOINC middleware. For a more in-depth discussion on creation of workunits with different parameters the reader is referred to section 5.4.

This discussion has shown that BOINC can potentially provide distributed simulation service to DES CSPs. However, this will also require the use of HLA-RTI distributed simulation middleware.

3.4.5.2 Investigation of distributed simulation service using Condor Java universe

Condor Java universe allows the execution of Java programs using Condor's standard job submission mechanisms. A CSP-based distributed simulation comprising of two models (say, model-A and model-B) can be submitted as two separate jobs (job-A and job-B). Each job has a corresponding job description file that lists Condor-defined variables like *executable* (this can be the name of the program which interfaces with the CSP and the HLA-RTI), *arguments* (the argument can be the name of the simulation model that will be used by the program), *transfer_input_file* (this can be the simulation model files, simulation experiment parameter files, etc. that have to be transferred over the network), and so on. The job description files for these jobs can be written such that they provide values that would facilitate the execution of different models on different nodes of the desktop grid. For example, the value provided to variable *transfer_input_file* in the job description files for job-A and job-B could be model-A and model-B respectively. Thus, it seems possible that Condor Java universe, along with HLA-RTI, will be able to provide distributed simulation service to DES CSPs.

3.4.6 Investigation of web-based simulation service

The two possible ways of supporting the CSP-specific web-based simulation service in the grid environment have been identified in section 3.3.6. These are through the use of web services and grid portals. A grid middleware that implements the OGSA standards will generally enable users to create web services that can be deployed over the middleware. In the context of CSP-based simulation it may thus be possible to create web services that expose CSP functionality, which in turn can be used by the simulation user to access the CSP. BOINC and Condor middleware do not conform to the OGSA framework, nor do they provide a custom web service hosting solution. Similarly, these middleware do not include a web-based front-end to submit jobs, which could possibly have been used to upload CSP-based simulation models and parameters for remote execution over grid nodes. BOINC and Condor are therefore considered unsuitable for providing web-based simulation service to the CSPs.

3.5 Suitability of BOINC and Condor for CSP-specific services

The previous section has examined BOINC and Condor middleware in relation to the six services proposed by the CSP-GC framework. Table 20 below summarizes the middleware that have been identified as having the potential of offering such CSP-specific services.

Table 20: BOINC and Condor support for CSP-specific services

CSP-specific service	Grid Middleware	Comments
Parallel computation service	<ul style="list-style-type: none"> ▪ Condor parallel universe 	MES and DES CSPs may need to support MPI / PVM
Task farming service (both MMMD and SMMD variants)	<ul style="list-style-type: none"> ▪ BOINC ▪ Condor Java universe ▪ Condor MW 	Condor MW cannot use the Condor java universe execution environment
Workflow service	<ul style="list-style-type: none"> ▪ Condor DAGMan with Condor Java universe 	None

Collaboration service (search & download of CSP models and virtual meetings)	None	None
Distributed simulation service	<ul style="list-style-type: none"> ▪ BOINC with HLA-RTI ▪ Condor Java Universe with HLA-RTI 	HLA-RTI distributed simulation middleware will also have to be used
Web-based simulation service (web services and grid portals)	None	None

Condor parallel universe execution environment supports the execution of parallel jobs. To exploit this environment, the MCS and DES CSPs will generally have to be implemented such that they support parallel processing through message passing mechanisms like MPI / PVM for inter-processor communication. However, none of the 45 MCS and DES CSPs surveyed in this research support such a feature and consequently they may not benefit from parallel simulation service that can possibly be offered by Condor. Task farming service and distributed simulation service can be potentially supported by both the middleware; however the latter service will also require using HLA-RTI middleware. Condor DAGMan can be potentially used along with Condor Java universe to support the workflow service. However, none of the middleware presently supports collaboration service and web-based simulation service. In summary it can be said that BOINC and Condor may be able to offer four of the six CSP-specific services.

BOINC and Condor have been identified as representative middleware for PRC and EDGC forms of grid computing (section 2.8). The rationale for this includes their wide deployment base and the fact that they are available free of cost. Since two specific middleware have been used to evaluate the potential of offering grid-facilitated CSP-specific services, it is difficult to generalize the results of this investigation. However, since both BOINC and Condor are based on the general principle of PRC and EDGC forms of grid computing, it is arguable that some of the conclusions pertaining to the extent of BOINC and Condor's support for CSP-specific services may well apply to other middleware implementations of PRC and EDGC. For example, the middleware that includes a workflow mechanism (like Condor DAGMan) should generally be able to support the CSP-specific workflow service, middleware which supports execution of Java-based programs should generally be able to provide task farming service using the CSP-grid integration technology presented in this thesis (CSP-grid integration technology is discussed in section 4.4), and so on.

3.6 Suitability of BOINC and Condor for deployment in industry

This section discusses the suitability of BOINC and Condor for supporting CSP-based simulation in industry. This discussion is not about the CSP-specific services (which have already been discussed in the earlier paragraph), but about implementation and deployment aspects of the middleware. It is informed by literature, by author's interactions with simulation experts and IT staff, and the author's own experience with implementing different grid-based solutions.

This discussion is structured under five specific categories. Four of these categories directly map to the implementation aspects of the middleware (over which a user usually has no control) and are considered important when deciding upon the suitability of the middleware for deployment in industry. These four categories refer to the *operating system* for which the middleware has been implemented, the number of ports that are opened by the middleware for *communication*, the *job scheduling mechanism* that is implemented and whether the middleware provides task-parallel or job-parallel *task farming* support. The fifth category, namely, *application support*, is specific to the application that is being written to be executed over the grid and over which the user has some control. The programming language being used to implement the application is the important consideration here. Table 21 below shows BOINC and Condor specific information pertaining to each of the five categories. The sections in this thesis that refer to the middleware specific information, which is presented in the table, have also been indicated. Each of the five categories are discussed next.

Table 21: BOINC, Condor and middleware deployment considerations

	BOINC	Condor
Operating system	UNIX / Linux to host BOINC server (section 2.9.2). The clients can be Windows based.	Supported on Windows (section 2.2.3.2). Some components are only supported in Unix / Linux, but for CSP-specific services Windows installation is adequate.
Communication	Uses port 80 (section 2.9.2)	Uses multiple, bi-directional, static and dynamic ports (section 2.10.1)
Job scheduling mechanism	“pull” based model of the master-worker architecture (section 2.9.1)	Implements “broker-based” job scheduling mechanism (2.10.1). (<i>Condor MW implements the “broker-based” model of the master-worker architecture [section 2.10.5]</i>)
Task farming support	Supports task-parallel applications (section 3.4.2)	Supports job-parallel applications (section 3.4.2). Condor MW supports both job-parallel and task-parallel applications.
Application support	Supports applications written using C++. User applications have to be compiled with the BOINC client C++ APIs (section 2.9.2)	Different Condor universes support user applications written in C, C++ and Java (section 2.10.2). For creating job-parallel and task-parallel applications, the user applications have to be compiled with the C++ Condor MW library (section 2.10.5).

3.6.1 Category - operating system

The table shows that BOINC requires at least one UNIX or Linux flavour operating system to support BOINC server side components. Although grid middleware targeted at UNIX and Linux operating systems were not considered appropriate for this research (section 2.8), BOINC was an exception because it allowed Windows-based BOINC clients to process the jobs. Nevertheless, the requirement of at least one UNIX/Linux PC for BOINC-based desktop computing may not fit with an enterprise’s existing infrastructure or expertise. Furthermore, the creation and management of projects on the BOINC server require a high degree of intervention from the user, which runs counter to the principle of transparent job processing which desktop grids should generally aspire to provide. Unlike BOINC, Condor does not rely

on the presence of any Unix/Linux PC within the Condor pool. Furthermore, job submission, job monitoring and result retrieval are relatively straightforward processes in Condor (see section 5.5). Thus, in the operating systems category, Condor may be more appropriate for deployment in organizations that have Windows infrastructure in place.

3.6.2 Category – communication

BOINC uses the HTTP port (port 80) for all communication. Condor middleware, on the other hand, uses multiple, bi-directional, static and dynamic ports. Deploying Condor middleware in industry will therefore require the network administrator to open a large number of ports – something which network administrators are generally most reluctant to do (Beckles et al, 2005). Thus, in the communications category, BOINC will generally be preferred for deployment in an organisation.

3.6.3 Category - job scheduling mechanism

It has been discussed earlier in section 2.2.3.3 that “pull” and “push” are two different middleware approaches for scheduling tasks on resources (Hantz and Guyennet, 2005). Its application is not limited to PRC and EDGC forms of grid computing. Cluster-based grid computing middleware also implement these approaches to process jobs that are submitted by the users. For example, EDC middleware implements the “push” approach (section 2.2.3.3) and gLite-3 middleware supports both approaches (section 2.2.3.5). If the middleware implements the “push” mechanism then it periodically polls the grid nodes to find out the load levels and decide on whether new jobs are to be assigned to the node; on the other hand, a middleware that implements the “pull” mechanism empowers the grid nodes to decide the best time to start a job and thereafter request a new job (Berlich et al., 2005). Furthermore, in the centralized “push” approach the state information of all the nodes is maintained at a central resource, whereas in the decentralized “pull” approach the system state information is maintained by each node (Garonne et al., 2005). A third approach can be a “broker-based” approach to job scheduling. In this case a software process (for example, the matchmaking agent in Condor) is responsible for matching jobs with available resources, before the job can be “pushed” from the job submission machine to the job execution machine. The broker-based approach has earlier been discussed in section 2.10.1. The implementation of the “pull” mechanism results in stateless grid (the system does not need to know the status of the underlying grid resources) which is a lot more fault tolerant and simpler to implement, but this comes at the expense of a slightly worse performance compared to a “push” implementation (Saiz et al, 2003). “pull” mechanism is generally suited for situations where the supply of jobs greatly exceeds the available computing resources and the jobs are not generally time critical (Garonne et al., 2004). This is typical of a PRC project.

This discussion now considers the efficiency of “pull”, “push” and “broker-based” scheduling mechanisms in the enterprise environment. Garonne et al. (2005) have conducted performance studies related to the efficiency of “pull” and “push” approaches in the context of

scheduling tasks on multiple local schedulers that are shared among many users. The results have shown that, in terms of performance for High Throughput Computing (HTC), the centralized “push” approach is better than the decentralized “pull” approach under ideal conditions (e.g., no network or hardware failures, no disk space shortage, no service failure, etc.). Similarly, a “broker-based” scheduling approach will generally be less efficient than the “push” based approach because the former introduces one more layer of communication between the nodes requesting resources and the nodes providing those resources.

It can further be argued that “pull” approach will generally be less efficient compared to a “push” approach in cases where the length of the job queue may be continually varying. In a “push” scheduling mechanism, as soon as a job becomes available it will be pushed to an available grid node. However, in the case of a “pull” scheduling mechanism, the grid nodes will request jobs at predefined intervals of time. If the request fails (because of server failure or because job queue is empty or for some other reason) then the grid node will generally wait for a predefined interval of time before making another request. For example, BOINC implements exponential client back off in case of server failure (Anderson, 2004). In cases where the BOINC server is up and running, but the BOINC clients are unable to “pull” jobs because the job queue is empty, the clients have to wait for a timeout period (usually 60 minutes) before requesting new job from the server (Chandra et al., 2005).

A general purpose grid middleware used in an enterprise environment will generally have a fluctuating queue size since many employees will be using the grid for processing their applications. In such cases a middleware that implements the “push” architecture will generally be able to utilize more CPU cycles for processing. In a large organization, which may have a grid infrastructure that comprises of 100’s of PCs, the additional processing capacity gained by using a push-based middleware compared to using a pull-based or broker-based middleware can be quite substantial. However, the evaluation of BOINC and Condor has shown that they do not implement the “push” approach. Thus, there may exist a need for a middleware for use in industry which implements the “push” based mechanism for job scheduling.

3.6.4 Category - task farming support

The terms task-parallel and job-parallel are discussed next in the context of task farming. Section 3.10.5 has previously discussed that in a task-parallel task farming application one master process is responsible for directing and coordinating the execution of multiple worker process and assimilation of the results; whereas in a job-parallel task farming application one application (or user) submits many jobs using standard middleware-specific job submission mechanisms and is responsible for the detection of job completion (it receives no job completion message from the middleware unlike task-parallel applications). It is arguable that BOINC only supports task-parallel applications because it consists of server side daemon processes like the *on-demand work generator* (generates BOINC workunits in response to a

scheduler request), *validator* (examines the results returned from the grid nodes) and *assimilator* (parses the results and inserts it into a database) that can together be considered as a master process that is in total control of multiple BOINC clients. The reader is referred to Anderson (2004) for more information on the BOINC daemon processes. Condor Java universe supports job-parallel applications, and Condor MW supports both job-parallel and task-parallel applications.

For conducting CSP-based simulation experiments, task-parallel applications will generally be better suited since one master process will be in control of the overall experimentation process. Thus, the simulation practitioner will usually be able to load the experiment parameters into the task-parallel application, which will in turn interact with the underlying grid middleware to schedule the experiments over different grid nodes, receive simulation results asynchronously from nodes, and finally collate the results and present them to the simulation user. Since both Condor and BOINC support task-parallel applications, both the middleware are considered appropriate for use in the task farming category.

3.6.5 Category - application support

Table 21 shows that BOINC supports user applications that are written in C++. This is because the user application will have to be compiled with BOINC-defined C++ APIs. Similarly, Condor MW supports user applications that are written in C++. The different Condor universes, however, support C, C++ and Java-based applications. For example, Condor standard universe and vanilla universe execution environments support C and C++ applications; Java is supported by Condor Java universe, etc.

Java is widely used in industry. It is generally accepted that the two important reasons contributing to its popularity and widespread use are, Java applications can be run on any operating system that has Java Runtime Environment (JRE) installed and Java is open source and available for free. Thus, in the application support category, it is arguable that Condor with Java universe execution environment will be the middleware of choice.

This research is investigating how grid computing technologies can be used to support CSP-based simulation practise in industry. Because of Java's extensive use in industry, the *CSP-grid integration technology* that is presented in this thesis is mainly based on Java. Although the technology presented also uses dynamic link libraries (DLLs) created in Visual Basic, it is more for the purpose of facilitating faster program development rather than a strict technical requirement. In other words, it is generally possible to implement the DLL code in the Java program itself. The CSP-grid integration technology is presented in section 4.4. Condor Java execution environment allows Java applications to be run using Condor, and as such the CSP-grid integration technology proposed in this thesis is compatible with Condor's Java environment.

3.6.6 Section summary

This section has investigated BOINC and Condor with respect to its suitability for deployment in industry for supporting CSP-based simulation. The suitability was assessed under five different categories, namely operating system, communication, job scheduling mechanism, task farming support and application support. It was found that none of the middleware had ideal implementation, with respect to supporting CSP-based simulation in industry, under all the five categories. For example, under the operating system category Condor was found suitable for deployment; under communication category BOINC, which uses the standard HTTP port for all its communication, was considered suitable since it does not require opening up extra ports; in the job scheduling mechanism category none of the middleware were considered ideal since they did not implement the “push” model of job scheduling; in the task farming support category, however, both BOINC and Condor were considered appropriate since both the middleware supported task-parallel task farming applications; finally, in the application support category, the use of Condor with Java execution environment was considered appropriate. Table 22 below summarizes this information.

Table 22: Ideal middleware implementation for CSP-based simulation

	Ideal middleware implementation for CSP-based simulation in industry	Middleware that implements the feature
Operating system	Middleware is supported on Windows operating system	Condor
Communication	Middleware opens only one communication port	BOINC
Job scheduling mechanism	Middleware implements “push” job scheduling mechanism	None
Task farming support	Middleware supports task-parallel task farming applications	BOINC and Condor
Application support	Middleware supports Java-based user applications	Condor with Java execution environment

The table above shows that neither Condor nor BOINC has an ideal middleware implementation for running CSP-based simulation in industry. The ideal middleware would be the one which is supported on Windows, which uses only one communication channel, implements the “push” job scheduling mechanism, supports task-parallel task farming applications and would support Java-based user applications. Thus, there may exist a need for a middleware that is an ideal implementation for supporting CSP-based simulation in industry.

3.7 Chapter summary

In this chapter a grid computing framework for CSP-based simulation is proposed (section 3.2). This framework is called the COTS Simulation Package – Grid Computing (CSP-GC) framework. The objective of this framework is to provide a logical structure for the evaluation of the hypothesis presented in this thesis. The CSP-GC framework consists of six CSP-specific services that can potentially be provided to simulation users in industry through the use of grid technology. This chapter has then discussed the implementation aspects of these

services from a technological perspective, i.e., how can grid computing middleware support the CSP-specific services (section 3.3)?

Next, this chapter has examined PRC middleware BOINC and EDGC middleware Condor in relation to the six CSP-specific services (section 3.4) and has concluded that both BOINC and Condor can potentially support some of these services (section 3.5). The chapter has concluded by identifying the implementation requirements of the ideal grid middleware for CSP-based grid computing in industry (section 3.6). These requirements have been one of the motivations for the development of the WinGrid middleware during the course of this research.

WinGrid is an EDGC middleware which have been developed specifically for CSP-based simulation in industry. As such, it implements all the ideal middleware requirements that have been identified in section 3.6. The next chapter presents an overview of WinGrid (and a web services extension of it called WinGrid-WS) and examines the level of support this middleware can provide for CSP-specific services.

4 DEVELOPMENT OF DESKTOP GRIDS FOR WINDOWS

4.1 Introduction

The previous chapter has proposed the CSP-GC framework. The framework has defined six grid-facilitated CSP-specific services that could potentially help the simulation practitioner in industry. The evaluation of PRC middleware BOINC and EDGC middleware Condor have shown that some of these services could possibly be supported by them. However, it has also been noted that neither of the middleware are ideal implementations for supporting CSP-based simulation in industry.

This chapter introduces a new grid computing middleware called WinGrid (Mustafee and Taylor, 2006a; Mustafee et al., 2006b). WinGrid is an EDGC middleware that is targeted at the Windows operating system (thus, the chapter name “WinGrid-The Desktop Grid for Windows”). The primary motivation for implementing WinGrid was to provide an ideal middleware implementation for supporting CSP-based simulations in industry. As such, WinGrid incorporates the five ideal middleware characteristics that were identified in the last chapter and were considered important for grid-based simulations in industry. Thus, WinGrid is supported on Windows, it uses only one communication channel, it implements the “push” job scheduling mechanism, it supports task-parallel task farming applications and would support Java-based user applications.

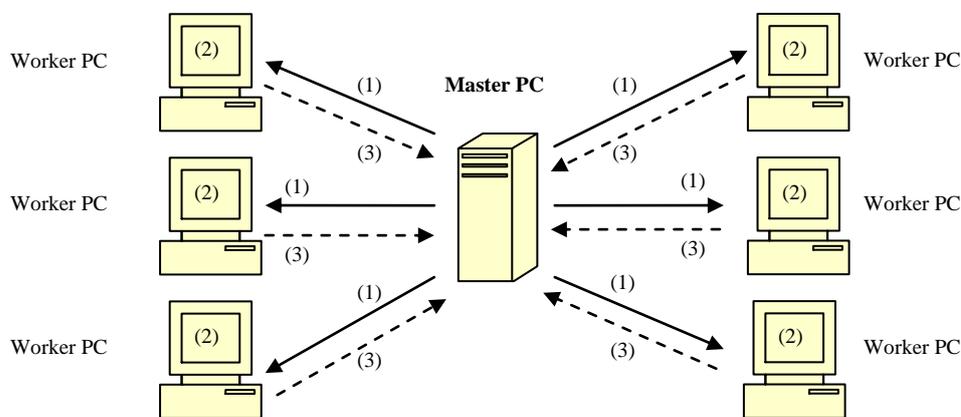
The author’s extensive involvement with the **GridAlliance** project (http://www.westfocus.org.uk/ICT/p54g12_Home.aspx) was another motivation for the development of WinGrid, as it was thought that source code control over the middleware would facilitate quick implementation of grid applications for industry. Grid Alliance was a **WestFocus** funded (<http://www.westfocus.org.uk>) one year project (2005-2006) between the University of Westminster and Brunel University that aimed at providing grid solutions to industry.

Subsequent to presenting an overview of the WinGrid architecture (section 4.2), this chapter discusses the web-service extension to WinGrid called WinGrid-WS (section 4.3), presents the CSP-grid integration technology that is used for WinGrid-CSP integration (section 4.4), evaluates both WinGrid and WinGrid-WS in relation to the CSP-specific services (section 4.5) and then concludes with a general discussion on their suitability for supporting the six CSP-GC framework defined services (section 4.6).

4.2 WinGrid architecture

WinGrid is a Java-based middleware that is based on the master-worker distributed computing architecture. It supports execution of task-parallel applications where a single master process is responsible for directing and coordinating the computations being executed

on the workers. This generally involves the master process dividing a problem into small parallel tasks, sending them over to the worker nodes for processing (using the WinGrid infrastructure) and assimilating the results that are returned by the workers (figure 32). WinGrid, being a Java-based grid middleware itself, supports the execution of task-parallel applications written in Java. WinGrid implements the “push” job scheduling mechanism. It does this by starting a server process on each worker. The server process enables the workers to continuously listen for incoming tasks from the master computer. The server process is started on only one port number at each worker node. Thus, WinGrid uses only one port number for all its communication. Before the architecture of WinGrid is described any further, the reader should note that the discussions in this paragraph and the previous section have highlighted that WinGrid implements all the five ideal middleware characteristics that were identified in the last chapter and were considered important for grid-based simulations in industry.



(1) Master PC “pushes” job to WorkerPCs, (2) WorkerPCs process job, (3) WorkerPCs return results to MasterPC

Figure 32: The “push” model implemented by WinGrid

WinGrid consists of the following four components:

- The *Manager Application (MA)*
- The *WinGrid Job Dispatcher (WJD)*
- The *Worker Application (WA)*
- The *WinGrid Thin Client (WTC)*.

The MA is a task-parallel application that runs on the master computer. The master computer is the desktop grid node from which jobs are submitted. The WJD, which is the WinGrid job scheduler, also runs on the master computer. The WAs and WTCs run on each worker computer that is part of the WinGrid infrastructure. The MA interacts with the WJD to transfer work to the WTCs. The WTCs, in turn, interact with their WAs to process the jobs and return the results back to the WJD. These results are then transferred by the WJD to the MA.

The WAs are unmodified software applications that are connected to the WTCs via open interfaces that are exposed by the applications. The WTC is also responsible for advertising and monitoring local grid resources, accepting new jobs from the master process and returning back the results. It provides a GUI interface through which the desktop user can set preferences like whether to accept guest jobs, which applications to share, etc. As shown in figure 33 below, the user submits a job through the MA (1), which in turn interacts with the WJD process (2) in the manager computer to send work (3) to the WinGrid workers and their WTCs (4). The WTC passes this work to their WA for processing (5) and returns the result to the WJD (6). The results of all the individual jobs are communicated back to the MA which then collates the results and presents it to the user.

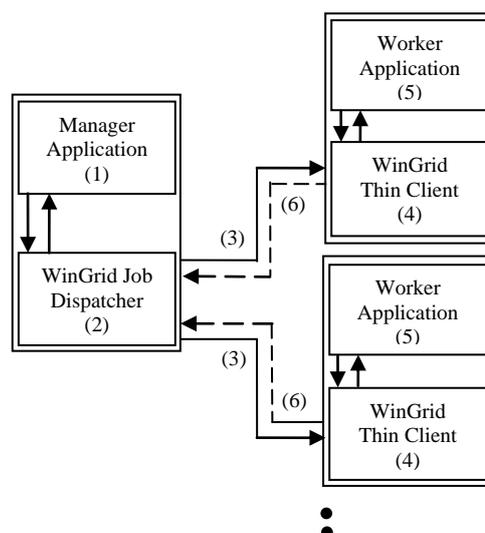


Figure 33: WinGrid architecture

In the context of using WinGrid with the CSPs to support task-parallel applications, the MA can be a user application that lists different experiment parameters and the WA can be unmodified CSPs that are installed over different WinGrid nodes. In order to simulate multiple instances of the model over different WTCs, WinGrid will usually have to create different instances of the CSP-model file and transfer them over to the different WTCs. The files can be transferred either through Sockets, or alternatively, it could be made available over a shared network drive. WinGrid presently uses a network share drive that is accessible by all the WTCs and the WJD. The experiment parameters that are present in the MA can be transferred to the WAs through the Socket channel that is established between the WJD and the WTCs. Thus, it is possible for different WTCs to start their WAs using different experiment parameters. The simulation results that are output by the WAs can similarly be returned back to the WJD. These results can then be collated together and displayed to the user through the MA.

As has been discussed earlier, WinGrid uses only one port number for all communication. However, unlike BOINC, the port it uses is not the standard HTTP port (i.e., port 80) – a port

that is usually reserved for web-based applications (including web services). Instead WinGrid uses port 5000. Port 5000 is opened on each WTC. This is unlike BOINC which, irrespective of the number of BOINC clients, usually operates through only one open port on the server side. The need to start one server process per WTC can sometimes be seen as a security-threat by organizations. The alternative to this can be to implement a single-server based “pull” job scheduling mechanism, whereby the workers pull jobs from the master, as it requires starting only one server process for the master. In this case the server listens continually for incoming job requests from the workers. This “pull” job scheduling mechanism, implemented through the Java web services extension to WinGrid, is discussed next. The reader is reminded that BOINC middleware is another grid middleware which implements the pull model.

4.3 WinGrid-WS architecture

The architecture of WinGrid-WS (Mustafee et al., 2006a) extends the original WinGrid architecture through the addition of the *WinGrid Shared Repository* (WSR). WSR is server software that needs to be installed on only one desktop grid node. In the WinGrid-WS architecture the WJD transfers user jobs to the WSR (1). To pull jobs from the shared job repository the WTCs send requests to the WSR on a regular basis (2). When a WTC has finished with a job it transfers the results back to the WSR (3). To retrieve the results (4), the WJD similarly sends out requests to the WSR on a regular basis. The interactions between the WJD and MA, and between WTC and WA are similar to that described earlier in section 4.2. Figure 34 below shows the architecture of WinGrid-WS.

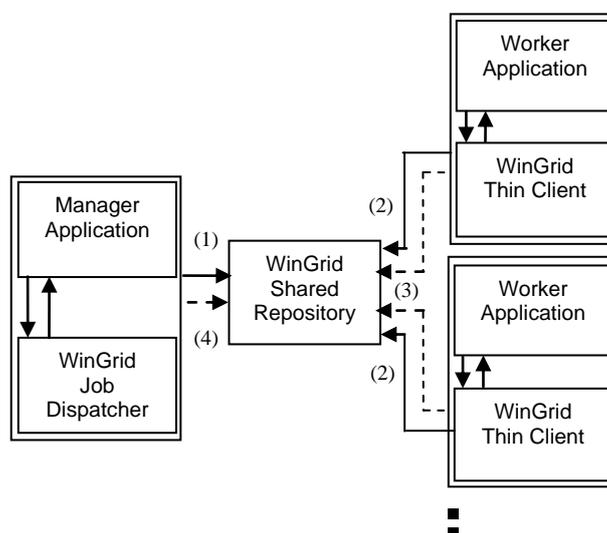


Figure 34: Architecture of WinGrid-WS

The next section presents the design and technology that has been used for the purpose of integrating WinGrid with CSPs. Although the discussion is specific to WinGrid and CSPs, the same design principle and technology can be potentially used for the integration of any Java-based grid middleware (or a Java-based execution environment like Condor Java universe) with applications that expose their package functionality through COM, OLE or other related

technologies. For obvious reasons the application being grid-enabled should generally have non-trivial processing requirements to benefit from the use of grid computing. The technology used for integrating WinGrid with CSPs is also referred to as *CSP-grid integration technology*.

4.4 CSP-grid integration technology

The CSPs are referred to as WAs in the WinGrid architecture. The WTCs running on different grid nodes need to communicate with the WAs to pass on simulation parameters, control simulation execution, retrieve the results, etc. Interaction between the WTCs and the CSPs is made possible through a Visual Basic Dynamic Link Library (DLL). The WTC, which is written in Java, invokes operations on the DLL through the use of Java Native Interface (JNI) (Sun Microsystems Limited, 2000). WinGrid uses the JACOB Java-COM Bridge (Alder, 2004), which in turn is based on JNI, for WTC-DLL communication. JNI is required because Java code cannot directly access native code (i.e., code written in a programming language other than Java). The DLL can be manually registered (using the Windows *regsvr32* command) or automatically registered (through WTC code) on the different WinGrid nodes. The DLL uses the vendor defined open interfaces to access the CSPs. The simulation packages that expose functionality have been listed earlier in table 5 in chapter 2.

The DLL has well-defined methods that are invoked by both WTCs and CSPs. The functions that are invoked by the WTCs are referred to as *WinGrid-defined Invocation Methods (WIM)*, whereas the methods called by the CSPs are referred to as *CSP-defined Callback Methods (CCbM)*. The DLL, in turn, invokes the open interfaces of the CSPs by calling *CSP-defined Invocation Methods (CIM)* to accomplish a variety of tasks, for example, to load a simulation package into computer memory, to load a simulation file into the CSP, etc. The DLL also returns back the results to the WTC by invoking *WinGrid-defined Callback Methods (WCbM)* through JNI. These callbacks enable asynchronous WTC processing (i.e., after invoking WIM the WTC does not have to wait for the method to return, and it can process some other code before it receives a callback from the DLL through WCbM).

The CSP packages usually provide some callback methods (CCbM) that have to be implemented by the application invoking the CIMs. For example, DES CSP Simul8 has a CCbM *MySimul8_S8SimulationEndRun* which is invoked by Simul8 to signal the end of a simulation run. DES CSP Witness, on the other hand, defines *Modelstatus* variable whose value has to be checked by an application from time to time to find out when a simulation run has ended. The DLL may, therefore, have to implement one of these mechanisms to receive information from the CSP.

In this research the DLL is also referred to as an *adaptor*. For interfacing WTC with a particular CSP a new adaptor will generally have to be written (WA adaptor). This adaptor will provide application-specific implementation to the WIM that will be invoked by the WTC. For

example, the implementation of the WIM method *wtcapp_runWA(timeX)*, which may be invoked by the WTC to run a simulation model to *timeX*, will be different based on the underlying CSP. Thus, a *Simul8 adapter* that interfaces the DES CSP Simul8 with WTC will use the Simul8-defined CIM *RunSim(timeX)* to run a model to *timeX*. Similarly, *Witness adapter* will use the Witness-defined CIM *Run(timeX)* to achieve a similar objective. The interaction between WTC-WAadapter-CSP is shown in figure 35 below.

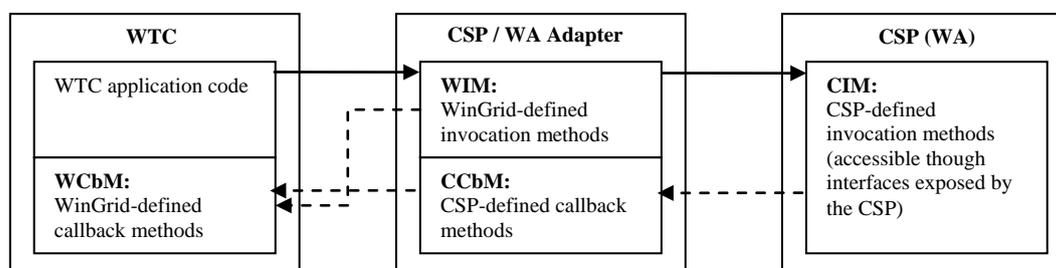


Figure 35: Interaction between WTC-WAadapter-CSP

Table 23 below lists some of the WIM and WCbM methods that are used for communication between WTC and the WA adapter and describes their purpose. However, it has to be added that these method signatures (method name and argument list) have not yet been standardized in WinGrid and some of them are written bespoke according to the requirements of the application.

Table 23: Interfaces used for communication between WTC and WA adapter

Interfaces	WIM / WCbM	Description of the interfaces <i>Note: The WIM and WCbM methods described in this table are implemented in WA adapter and WTC respectively.</i>
<i>wtc_init ()</i>	WIM	WA adapter initialization method. This method can be used to initialize variables defined by the WA adapter, register DLLs, etc.
<i>wtc_dest ()</i>	WIM	This method can be invoked by the WTC to deregister DLLs, de-initialize variables defined by the WA adapter, etc.
<i>wtcapp_init ()</i>	WIM	This method can be used to perform application specific initialization pertaining to the WA. For example, loading CSP-specific libraries.
<i>wtcapp_dest ()</i>	WIM	This method can be used to perform application specific de-initialization pertaining to the WA. For example, it can be used to unload CSP-specific libraries.
<i>wtcapp_openWA (filename, phase, ...)</i>	WIM	Opens the WA. The filename is the name of a CSP file. Phase is used only for workflows.
<i>wtcapp_closeWA ()</i>	WIM	Closes the WA.
<i>wtcapp_decipherApplication SpecificMessage (message, ...)</i>	WIM	The WJD transfers experiment parameters in the form of messages to the different WTCs. These messages are encoded using method <i>wjdapp_encodeApplicationSpecificMessage()</i> [See table 24]. This method deciphers the messages that are received from the WJD.
<i>wtcapp_setExperimentParams ()</i>		Invokes appropriate CIM calls to insert experiment parameters into the WA.
<i>wtcapp_runWA (timeX)</i>	WIM	This method is used to run a simulation till <i>timeX</i> .
<i>wtcapp_getResults ()</i>	WIM	Extracts the results from the WA.
<i>wtc_simulationComplete ()</i>	WCbM	This callback method is invoked from the WA adapter. It informs the WTC that simulation is

Interfaces	WIM / WCbM	Description of the interfaces <i>Note: The WIM and WCbM methods described in this table are implemented in WA adapter and WTC respectively.</i>
		complete.
<i>wtc_jobComplete (results)</i>	WCbM	Returns the result to the WTC. This callback method is invoked from <i>wtcapp_getResults()</i> .

The WTC-WAadapter-CSP integration design makes it possible for the WTC to communicate with WA on the worker nodes. A similar design is implemented for communication between the WJD and the MA on the master node. For example, the MA can be an Excel spreadsheet that contains the parameters of an experiment. An *Excel adapter* (MA adapter) can thus be created for communication between the WJD and the Excel spreadsheet (figure 36). The WJD will communicate with the MA adapter through well-defined WIM and WCbM methods. The MA adapter will use the *Excel-defined Invocation Methods (EIM)* and *Excel-defined Callback Methods (ECbM)* to communicate with the MA. Table 24 lists some of the WIM and WCbM methods that are used for communication between WJD and the MA adapter on the WinGrid master computer.

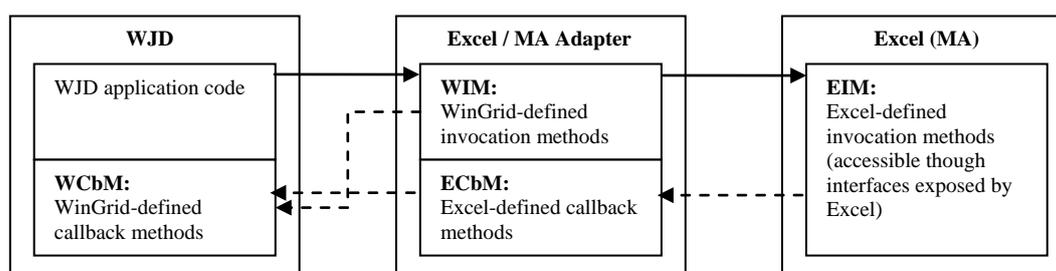


Figure 36: Interaction between WJD-MAadapter-Excel

Table 24: Interfaces used for communication between WJD and MA adapter

Interfaces	WIM / WCbM	Description of the interfaces <i>Note: The WIM and WCbM methods described in this table are implemented in MA adapter and WJD respectively.</i>
<i>wjd_init ()</i>	WIM	MA adapter initialization method. This method can be used to initialize variables defined by the MA adapter.
<i>wjd_dest ()</i>	WIM	This method can be invoked by the WJD to de-initialize variables defined by the MA adapter.
<i>wjdapp_init ()</i>	WIM	This method can be used to perform application specific initialization pertaining to a MA.
<i>wjdapp_dest ()</i>	WIM	This method can be used to perform application specific de-initialization pertaining to a MA.
<i>wjd_openPropertiesFile ()</i>	WIM	Opens the WTC properties file. This file contains the IP addresses, port numbers and machine names of the different WinGrid nodes.
<i>wjd_closePropertiesFile ()</i>	WIM	Closes the WTC properties file.
<i>wjd_getIPsandPorts ()</i>	WIM	Extracts IP addresses, port numbers and machine names from the WTC properties file.
<i>wjd_returnIPandPort (value)</i>	WCbM	Returns the IP addresses, port numbers and machine names to the WJD. This callback method is invoked from <i>wjd_getIPsandPorts ()</i> .
<i>wjdapp_openMA (filename)</i>	WIM	Opens the MA. The filename is the name of the application to open. For example, if the MA is an Excel-based application then the file to open is an Excel file.
<i>wjdapp_closeMA ()</i>	WIM	Closes the MA.
<i>wjdapp_encodeApplication SpecificMessage ()</i>	WIM	Gets experiment information from the MA. The implementation of this method will extract the required values

Interfaces	WIM / WCbM	Description of the interfaces <i>Note: The WIM and WCbM methods described in this table are implemented in MA adapter and WJD respectively.</i>
		from the MA and will construct a string with different fields, separated with field demarcations like comma and colon. This method is called only once by the WJD. Parameters pertaining to all the experiments are returned to the WJD through invocation of <i>wjdapp_returnApplicationSpecificMessage</i> (see below). WJD will then create sub-jobs for each experiment and will allocate them to the available WTCs. The jobs received by the WTCs are deciphered using method <i>wtcapp_decipherApplicationSpecificMessage(message, ..)</i> [See table 23].
<i>wjdapp_returnApplicationSpecificMessage (message)</i>	WCbM	Returns the experiment parameters to the WJD. This WJD-implemented callback method is invoked from <i>wjdapp_encodeApplicationSpecificMessage ()</i> .
<i>wjdapp_gatherResults (var1, var2, ..)</i>	WIM	This method is invoked by the WJD to collate individual results returned by the WTCs. Its implementation is specific to the MA.
<i>wjdapp_resultcollection_openFile (filename, var , ..)</i>	WIM	This method is invoked when the individual WTC results are presented in a different file (not the MA). This method will open the file whose <i>filename</i> is passed as an argument.
<i>wjdapp_resultcollection_closeFile()</i>	WIM	This method closes the result collection file that was earlier open by <i>wjdapp_resultcollection_openFile (filename, var , ..)</i> .
<i>wjdapp_donemessage</i>	WCbM	This callback method informs the user that application processing is complete. It is invoked after <i>wjdapp_gatherResults (var1, var2 ..)</i> has completed.

Tables 23 and 24 have shown the interfaces that can be used for communication between the WTC and the WA adapter on the worker nodes and the WJD and the MA adapter on the master node. These WIM and WCbM methods are defined by WinGrid and are implemented by the WA adapter, MA adapter, WTC or the WJD. For the WA / MA adapter to actually invoke the WA / MA, it should have access to open interfaces made available by the external WA and MA applications. In the context of this research, the MAs are Excel-based applications and the WAs are the CSPs.

Excel applications can be accessed through *Microsoft Excel 11.0 Object Library*. This library can be imported when writing the Visual Basic MA adapter for communication with the MA. The library provides methods that can be used by the adapter to access Excel-defined workbooks and worksheets, cells, formulas, formatting functions, etc. Using these methods the adapter will generally be able to extract experiment parameters from the Excel-based MA and import simulation results into it. Some of the methods that can be called by the MA adapter are presented in table 25 below. The reader is referred to Microsoft Support (2007) for an example that shows a Visual Basic application accessing Excel through the Microsoft Excel 11.0 Object Library.

Table 25: Interfaces used for communication between MA and MA adapter

Interfaces	EIM / ECbM	Description of the interfaces <i>Note: EIM and ECbM methods are implemented in MA(Excel) and MA adapter respectively.</i>
<i>Excel.Application varXIApp = New Excel.Application ()</i>	EIM	Gets a reference to the <i>Excel.Application</i> object. The variable which holds a reference to this object is

Interfaces	EIM / ECbM	Description of the interfaces <i>Note: EIM and ECbM methods are implemented in MA(Excel) and MA adapter respectively.</i>
		<i>varXIApp.</i>
<i>Excel.workbook varXIWbk = varXIApp.Workbooks.Open (filename)</i>	EIM	Opens Excel workbook (MA) with the name <i>filename</i> . The reference to this workbook is held in the object <i>varXIWbk</i> .
<i>Excel.worksheet varXIWsheet = varXIWbk.Worksheets (sheetname)</i>	EIM	A workbook can contain multiple worksheets. This method selects the worksheet with the name <i>sheetname</i> . A reference to this worksheet is held in variable <i>varXIWsheet</i> .
<i>varXIApp.Visible = TRUE</i>	EIM	This is an Excel-defined property. The value <i>TRUE</i> indicates that the Excel application will be visible to the user.
<i>varXIWsheet.Cells (rownumber, colnumber)</i>	EIM	This method is used to either extract values from cells or to assign values to cells within a worksheet. A reference to the worksheet is held in variable <i>varXIWsheet</i> . The row number and column number of the cell are passed as arguments <i>rownumber</i> and <i>colnumber</i> .
<i>varXIApp.Application.Run (macroname)</i>	EIM	This method is used to execute a user-defined macro with name <i>macroname</i> .
<i>varXIApp.Worksheets.Add ()</i>	EIM	Adds a new worksheet to the workbook.
<i>varXIWbk.Close (Boolean FALSE)</i>	EIM	Closes the workbook without saving any changes (<i>FALSE</i>). A Boolean value <i>TRUE</i> will save and close the workbook.
<i>varXIApp.Quit ()</i>	EIM	Quits the Excel application.
<i>varXIApp _WorkbookOpen (Wb As Excel.Workbook)</i>	ECbM	This callback method is invoked by Excel when a workbook is successfully opened.
<i>varXIApp _WorkbookNewSheet (Wb As Excel.Workbook, Sh As Object)</i>	ECbM	This callback method is invoked by Excel when a new worksheet is successfully added to a workbook.

In the case of the WA, the simulation practitioner will have to refer to the documentation provided by the vendor to investigate whether the CSP could be accessed by an external application, and if so, the functionality that can be accessed. For implementing CSP task farming using WinGrid, the exposed interfaces should generally support operations to load the CSP software into computer memory, open and save model files, import variable values (like queue size, processing time for a workstation, etc.) into the model, execute a model for a pre-defined simulation time, extract results from the model (example, number of entities that have been processed, number of entities waiting in a queue, etc.), execute CSP-defined program code (example, Visual Logic in case of Simul8) and exit the CSP application. To support these operations, a CSP-specific library (if available) can be imported when writing the Visual Basic WA adapter. Table 26 below lists some of the CIM and CCbM methods that are exposed by CSP Simul8 Professional and which can be used to support task-parallel task farming applications. For a more exhaustive list of the COM methods the reader is referred to Simul8 Corporation (2002). CSP Simul8 Professional is used as an example to highlight the key functionality that may be required to be exposed by the CSPs. As such, the descriptions of the interfaces are more important than the signatures of the interfaces themselves.

Table 26: Interfaces used for communication between WA and WA adapter

Interfaces	CIM / CCbM	Description of the interfaces <i>Note: CIM and CCbM methods are implemented in WA (Simul8 Professional) and WA adapter respectively.</i>
<i>SIMUL8.S8Simulation varS8Obj = GetObject ("", "SIMUL8.S8Simulation")</i>	CIM	Gets a reference to the <i>SIMUL8.S8Simulation</i> object. The variable which holds a reference to this object is <i>varS8Obj</i> .

varS8Obj.Open (<i>filename</i>)	CIM	Loads Simul8 into computer memory and opens the file specified by the argument <i>filename</i> .
varS8Obj.Save (<i>filename</i>)	CIM	Saves the Simul8 model that is presently open with the name specified by the argument <i>filename</i> .
varS8Obj.RunSim (<i>timeX</i>)	CIM	Runs simulation till <i>timeX</i> .
varS8Obj.ExecVL (<i>command</i>)	CIM	Executes Simul8 Visual Logic code. The code to be executed is specified in the argument <i>command</i> . For example, <i>varS8Obj.ExecVL ("SET Machine1.Operation Time = 10")</i>
varS8Obj.SimulationTime	CIM	This is a read-only property. It returns the present simulation time from Simul8.
varS8Obj.StopSim ()	CIM	Stops the presently running simulation.
varS8Obj.Close ()	CIM	Closes the Simul8 model that is presently open.
varS8Obj.Quit ()	CIM	Exits Simul8.
<i>SIMUL8.S8SimObject</i> varS8SimObject = varS8Obj.SimObject (<i>objectname</i>)	CIM	Gets a reference to a Simul8 simulation object (for example, workcentre, storage, entry, exit, conveyor, tank, resource, etc.) that is present in the model. The variable which holds a reference to this object is <i>varS8SimObject</i> .
varS8SimObject.Completed	CIM	This is a read-only property. If <i>varS8SimObject</i> is a Simul8 workcentre object, then this value refers to the number of entities that have been processed by the workcentre.
varS8SimObject.CountContents	CIM	This is a read-only property. If <i>varS8SimObject</i> is a Simul8 exit (sink) object, then this value refers to the number of entities that have been processed by the model.
varS8Obj_S8SimulationEndRun ()	CCbM	This callback method is invoked by Simul8 when a simulation run is complete.
varS8Obj_S8SimulationReset ()	CCbM	This callback method is invoked by Simul8 when a simulation has been successfully reset.
varS8Obj_S8SimulationEndTrial ()	CCbM	This callback method is invoked by Simul8 when a trial has ended.

The interaction between the different WinGrid components (namely, MA, MA adapter, WJD, WTC, WA adapter and WA) are shown using a UML sequence diagram in figure 37. The WA in this example is CSP Simul8 Professional and the MA is an Excel-based application. The reader should use the sequence diagram only as a reference, as the sequence of the method invocations and indeed the methods themselves may vary depending on the application to be grid-enabled. The interfaces used by WinGrid have not yet been standardized and some of them are written bespoke based on the application requirement.

This section has described the adapter technology that has been used in this research for the integration of WinGrid with CSPs. Since this technology is based on Java, any middleware that supports the execution of Java programs can also potentially use this technology. For example, it may be possible to use the Condor middleware with the Java universe execution environment to implement CSP-specific task farming. Furthermore, the WAs are not limited to CSPs alone and any application that (1) exposes its functionality through well-defined interfaces, (2) requires non-trivial amounts of CPU cycles to process user jobs, and (3) supports partitioning of a large job into multiple parallel sub-jobs, can be considered as a potential WA that might gain from task farming using WinGrid. The next section examines WinGrid and WinGrid-WS with regards to the CSP-specific services proposed by CSP-GC framework.

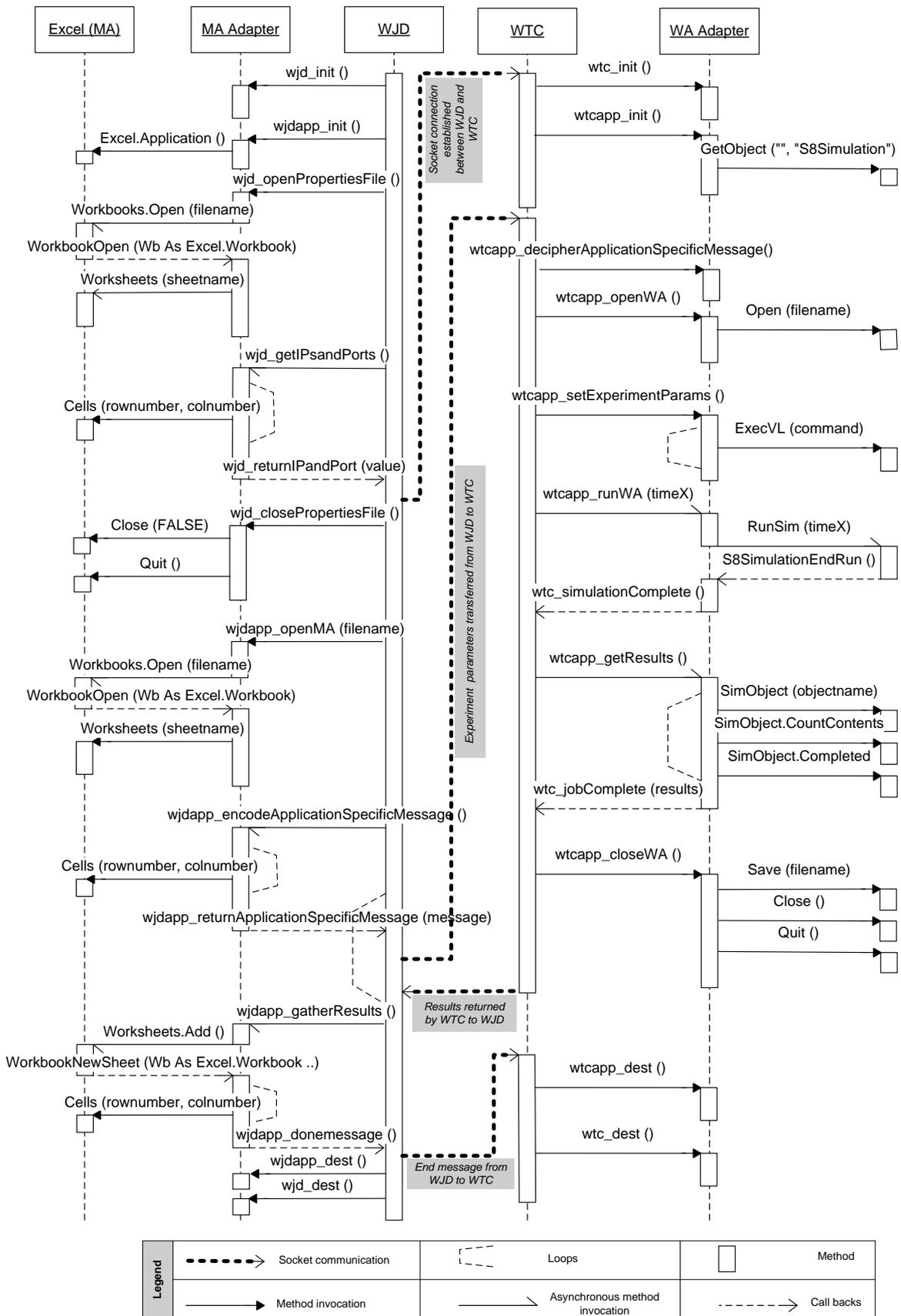


Figure 37: UML sequence diagram showing the interaction between WinGrid components

4.5 Investigation of CSP-specific services using WinGrid and WinGrid-WS

The CSP-GC framework, presented in chapter 3, has identified six CSP-specific services that can be potentially used to support CSP-based simulations in industry. This section examines WinGrid in relation to the CSP-specific services to assess the degree to which this purpose built middleware, for CSP-based simulation in industry, can support these services. WinGrid-WS is only discussed in the context of task farming and web-based simulation service because it was specifically implemented for providing these two services.

4.5.1 Investigation of parallel computation service

Section 3.5 has highlighted that for a CSP to effectively benefit from the parallel simulation service, it is generally required that these packages are implemented to support parallel processing through message passing mechanisms like MPI / PVM. However, none of the 45 MCS and DES CSPs that have been surveyed in this research presently have a parallel implementation (section 2.5.1). Consequently, incorporating parallel computing service with WinGrid was not considered necessary at this time because WinGrid is specifically implemented for CSP-based simulation, and the CSPs do not presently have parallel MPI/PVM implementations.

4.5.2 Investigation of task farming service

The discussion on WinGrid (section 4.2) and WinGrid-WS (section 4.3) has shown that these middleware can potentially support task-parallel task farming service through their respective “push” and “pull” implementations. Furthermore, WinGrid may be able to support both SMMD and MMMD variants of task farming as it is possible to run different programs concurrently over multiple grid hosts (similar to Condor). WinGrid-WS was designed for only SMMD task farming.

4.5.3 Investigation of workflow service

WinGrid implements workflows through the WJD – the job scheduler for WinGrid. The WJD is aware of the dependencies between different jobs that are submitted to it. This is a bespoke WJD solution for executing workflows and has its disadvantages. Hard coding workflow logic into the WJD implies that WinGrid will not be able to support other workflow routines until the source code itself is modified. The improved solution to this is to create a WinGrid Workflow component on top of the WJD (like DAGMan in Condor). The application workflow logic, which can be represented using XML-defined tags, can then be input into the WinGrid Workflow component and which will thereafter be responsible for submitting jobs to WJD based on the underlying workflow logic.

4.5.4 Investigation of collaboration service

Grid facilitated collaboration service can be provided in two possible forms. One, through providing mechanisms that facilitate the search and download of CSP-based model components over the grid, and, two, through virtual meetings. Section 3.3.4 has identified the

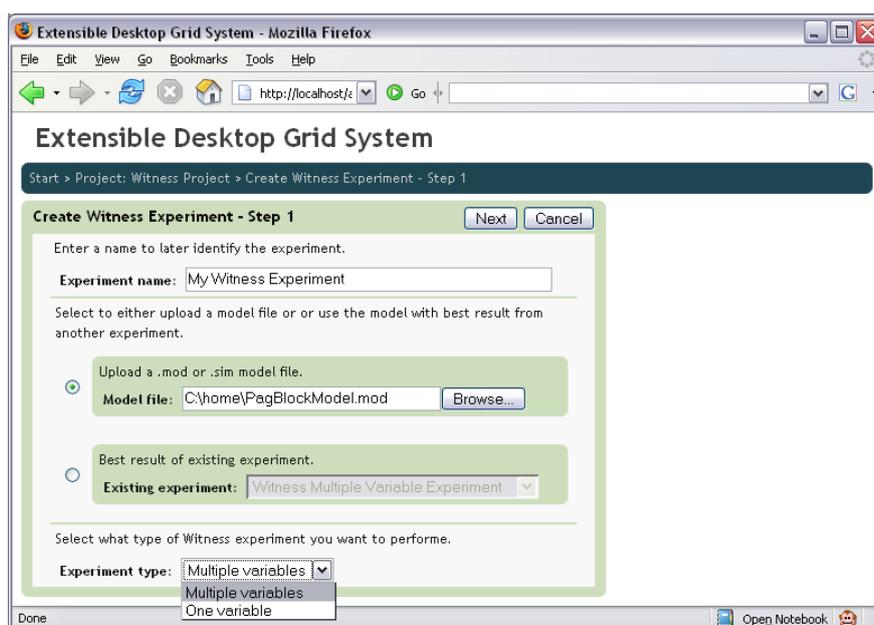
use of web services and integrated video conferencing mechanisms for enabling this service through grid middleware. WinGrid does not support the hosting of user-defined web services. Moreover, communication in WinGrid is implemented using Java sockets and not web services. Neither does it include integrated support for virtual meetings. WinGrid is therefore considered inappropriate for supporting CSP-based collaboration service over the grid.

4.5.5 Investigation of distributed simulation service

Investigation of distributed simulation service with BOINC and Condor (section 3.4.5) has shown that the “application integration approach”, wherein the user application (consisting of CSP models and associated code) is responsible for managing a distributed simulation federation, is more appropriate for distributed simulation in the grid environment. WinGrid supports this approach. WinGrid has mechanisms to execute multiple CSP models concurrently over different grid nodes, and it can therefore potentially provide distributed simulation service through use of HLA-RTI middleware.

4.5.6 Investigation of web-based simulation service

As discussed in section 3.3.6, web-based simulation service can be potentially supported over the grid environment through the use of web services and grid portals. As has been said earlier, WinGrid does not support the hosting of user-defined web services. Neither does WinGrid have a web-based front-end. WinGrid-WS, on the other hand, uses web services for communication between grid nodes. But it does not presently support deployment of user-developed web services. However, WinGrid-WS implements a grid portal (screenshot 5) that could be accessed by simulation users through their web browsers. The portal makes it possible to upload simulation models and experiment parameters for batch execution, monitor the progress of simulations and download the results.



Screenshot 5: Web front-end to WinGrid-WS (Alstad, 2006)

In summary, it can be said that WinGrid does not provide support for web-based simulation service. WinGrid-WS only partially supports this service through a grid portal.

4.6 Suitability of WinGrid and WinGrid-WS for CSP-specific services

The previous section has examined the level of support that WinGrid can provide for all the six CSP-specific services. It has also examined WinGrid-WS in relation to two specific services – task farming service and web-based simulation service. This investigation has shown that WinGrid can potentially support distributed simulation, both variants of task farming service and work flow service; WinGrid-WS can support the SMMD variant of task farming service and web-based simulation service (through grid portal). Table 27 below summarizes this information.

Table 27: WinGrid and WinGrid-WS support for CSP-specific services

CSP-specific service	Grid Middleware	Comments
Parallel computation service	▪ None	MES and DES CSPs may need to support MPI / PVM
Task farming service (both MMD and SMMD variants)	▪ WinGrid ▪ WinGrid-WS	WinGrid-WS presently supports only SMMD task farming
Workflow service	▪ WinGrid	None
Collaboration service (search & download of CSP models and virtual meetings)	None	None
Distributed simulation service	▪ WinGrid with HLA-RTI	HLA-RTI distributed simulation middleware will also have to be used
Web-based simulation service (web services and grid portals)	▪ WinGrid-WS	WinGrid-WS presently supports only grid portals

WinGrid and WinGrid-WS specific information provided in the table above is now combined with the middleware suitability information presented in the context of BOINC and Condor in section 3.5, to present the reader with a single suitability reference table (table 28) that lists all the CSP-specific services and the middleware that could be potentially used to support them.

Table 28: Middleware support for CSP-specific services

CSP-specific service	Grid Middleware	Comments
Parallel computation service	▪ Condor parallel universe	MES and DES CSPs may need to support MPI / PVM
Task farming service (both MMD and SMMD variants)	▪ BOINC ▪ Condor Java universe ▪ Condor MW ▪ WinGrid ▪ WinGrid-WS	Condor MW cannot use the Condor java universe execution environment. WinGrid-WS presently supports only SMMD task farming
Workflow service	▪ Condor DAGMan with Condor Java universe ▪ WinGrid	None
Collaboration service (search & download of CSP models and virtual meetings)	None	None
Distributed simulation service	▪ BOINC with HLA-RTI ▪ Condor Java Universe with HLA-RTI ▪ WinGrid with HLA-RTI	HLA-RTI distributed simulation middleware will also have to be used
Web-based simulation service (web services and grid portals)	▪ WinGrid-WS	WinGrid-WS presently supports only grid portals

The information presented in the table shows varying levels of grid support for CSP-specific services. On the one hand, task farming service can be potentially supported by all middleware; on the other hand, collaboration service is not supported at all. The parallel simulation service is unique in the sense that although Condor could potentially support it, the non-parallel implementation of CSPs limits its use. Furthermore, the table identifies that WinGrid-WS is the only middleware that can partially support web-based simulation service through the use of grid portals.

WinGrid can be criticized along the lines that, although it was implemented specifically for CSP-based simulation in industry, it does not support all the CSP-specific services. However, the reader is reminded that the ideal middleware implementation requirements, identified in section 3.6.6 of this thesis, were mainly based on core middleware architecture designs which were considered important for grid middleware deployment in industry. It is however acknowledged by the author that a middleware specifically implemented to support CSP-based simulation should ideally support all the six services defined by the CSP-GC framework. WinGrid was developed on an incremental basis. The services that were considered most important were implemented first. Thus, support for task farming service was included in WinGrid, followed by the inclusion of workflow service and distributed simulation service. The support for web-based simulation service was included in WinGrid-WS. Support for parallel simulation service was not considered a top priority at this time because the CSPs do not presently provide MPI/PVM implementations. Similarly, support for virtual meeting, a form of collaboration service, was presently not considered because groupware applications like Microsoft NetMeeting include support for virtual meetings and are usually available for use in the Windows environment.

4.7 Chapter summary

This chapter has presented the architecture of WinGrid (section 4.2). WinGrid is an EDGC middleware that has been implemented during the course of this research primarily to provide an ideal middleware implementation for supporting CSP-based simulations in industry. This chapter has also presented an overview of WinGrid-WS middleware (section 4.3). WinGrid-WS supports task-parallel task farming with a “pull” based job scheduling mechanism. The CSP-grid integration technology for communication between WinGrid with CSPs, through the use of adapters, is discussed in section 4.4. It is further noted that this adapter-based approach can be potentially used with any Java-based middleware (or a middleware that supports Java execution environment).

This chapter has then examined the extent to which WinGrid and WinGrid-WS can support the CSP-specific services (section 4.5). The discussion on WinGrid-WS was however limited to task farming service and web-based simulation service because WinGrid-WS was specifically implemented to support only these two services. The examination of the

middleware has shown that some of the services can be supported by WinGrid and WinGrid-WS (section 4.6).

The hypothesis presented in this thesis is that CSP-based simulation practice in industry will benefit from the adoption of grid computing technologies. To provide a logical structure to evaluate this hypothesis the CSP-GC framework has been proposed. The framework has identified six CSP-specific services that are derived from the higher level grid services. Through a detailed investigation of four grid computing middleware – BOINC, Condor, WinGrid and WinGrid-WS – it has been established that some of these services can be potentially supported. Thus, until this point the hypothesis has not been disproved. To prove this hypothesis, however, it has to be shown through implementation that grid middleware can be used together with the CSPs to provide the middleware specific solutions that have been recognised to support the CSP-specific services (section 3.5 and section 4.6). This will be investigated through case study experimentation in the next chapter.

5 CASE STUDIES

5.1 Introduction

The previous chapter has presented an overview of WinGrid and the web services extension to WinGrid called WinGrid-WS. WinGrid was developed during the course of this research as it was considered important to investigate an EDGC middleware that implemented the ideal middleware characteristics, identified in section 3.6.6, for executing CSP-based simulation in industry. The chapter then examined WinGrid in relation to the six CSP-specific services. It was argued that WinGrid was potentially able to support three of these services, namely, task farming service, workflow service and distributed simulation service. WinGrid-WS, on the other hand, supported task farming service and web-based simulation service through use of grid portal. Thus, the hypothesis presented in this thesis has not been disproved because all four grid middleware that have been assessed, namely BOINC (chapter 2), Condor (chapter 2), WinGrid and WinGrid-WS (chapter 4), can potentially support some of the CSP-GC framework identified services.

This chapter investigates whether the theoretical and technical evaluation of the middleware, presented in the earlier chapters, in support for CSP-specific services is realizable in practice, i.e., can it be implemented? Section 5.2 presents the criteria for the evaluation of the hypothesis. These criteria are tested using a total of five real-world and hypothetical case studies. These case studies are outlined in section 5.3 together with their evaluation criteria. The first and second case studies examine BOINC and Condor in relation to SMMD and MMMD task farming service in sections 5.4 and 5.5 respectively. This is followed by an investigation of WinGrid in the context of SMMD task farming service in the third case study (section 5.6). The fourth case study evaluates the workflow service using WinGrid (section 5.7). This is followed by an investigation of WinGrid in relation to distributed simulation service in the fifth and final case study (section 5.8).

5.2 Criteria for hypothesis evaluation

To prove the hypothesis that the adoption of grid computing will help the simulation practitioner in industry, it has to be shown that the middleware can operate together with the CSPs towards the realization of the CSP-specific services.

For the evaluation of distributed simulation service and task farming service, another yardstick could be the time taken to execute simulation experiments over a grid as compared to standalone execution. However, such an evaluation may only be possible if the grid infrastructure is comprised of dedicated resources (like cluster-based grid computing). The two forms of grid computing that have been found suitable for implementing grid solutions in industry are PRC and EDGC. The middleware for both of them are designed for non-dedicated resources like desktop PCs that are used by the employees at their workplace.

PRC and EDGC are primarily meant for High Throughput Computing where the objective is to provide sustained access to idle CPU cycles made available by the user PCs. But it is also true that PCs that are not being used can be considered as dedicated resources, and a simulation performance comparison with dedicated standalone PCs, in this case, could therefore be justifiable.

In the case of task farming service it can intuitively be said that multiple dedicated grid nodes will execute a set of simulation experiments faster, compared to one dedicated desktop PC, and the more the number of grid nodes the faster the execution speed. It can also be argued that faster grid execution over non-dedicated resources is very much possible as research has shown that desktop PCs can be under utilized by as much as 75% of the time (Mutka, 1992). However, this may ultimately depend upon the number of resources that make up the grid infrastructure and specific usage pattern of the PC owners.

In the case of distributed simulation service it may be difficult to arrive at a similar conclusion because the simulations running on different PCs will need synchronization between them, i.e., they are more like peer-to-peer simulation where the executions of the models are interlinked. In this case, an efficient execution of a set of distributed models may only be possible if the grid nodes are dedicated. This is because in the case of non-dedicated resources an interruption in the running of even one model may eventually halt the entire distributed simulation federation.

In summary, the hypothesis presented in this thesis is primarily evaluated based on whether the identified grid middleware solutions for supporting CSP-specific services are implementable in practice, and thereby whether the services are realizable. In the case of task farming service and distributed simulation service the additional requirement is whether grid computing can offer better performance compared to standalone execution, and thereby making it a viable technology for use by the simulation practitioners. In the case of task farming service, the second requirement has to be evaluated using both dedicated and non-dedicated resources. In the case of distributed simulation service this requirement needs to be evaluated using only dedicated resources. Parallel simulation service and collaboration service is not being considered for hypothesis evaluation because it has been identified in section 4.6 that none of the middleware that have been examined in this research would support these services.

As case studies are being used to examine the CSP-GC framework defined services, the hypothesis evaluation criteria that have been discussed in this section will be used for the evaluation of the case studies. The hypothesis evaluation criteria are applicable to both DES and MCS CSPs; the only exception being distributed simulation service which is applicable

only in the case of DES CSPs. Table 29 below summarizes the case study evaluation criteria for the four CSP-specific services that have been identified as “grid implementable solutions”.

Table 29: Criteria for hypothesis evaluation

CSP-specific service	Case study evaluation criteria
Task farming service (both <i>SMMD</i> and <i>MMMD</i> variants)	<ul style="list-style-type: none"> ▪ Solution is implementable and the service is realizable ▪ Execution is faster over dedicated grid resources compared to a standalone execution ▪ Execution is faster over non-dedicated grid resources compared to a standalone execution
Workflow service	<ul style="list-style-type: none"> ▪ Solution is implementable and the service is realizable
Distributed simulation service	<ul style="list-style-type: none"> ▪ Solution is implementable and the service is realizable ▪ Execution is faster over dedicated grid resources compared to a standalone execution
Web-based simulation service (<i>grid portals</i>)	<ul style="list-style-type: none"> ▪ Solution is implementable and the service is realizable

5.3 CSP-GC framework investigation scenarios

In this section the terms “investigation scenarios” and “case studies” are used synonymously. The *CSP-GC framework investigation scenarios* are important because they provide a well-defined structure for experimental evaluation of the CSP-GC framework, and in turn form the basis for evaluation of the hypothesis.

The case studies are either based on real-world problems that were encountered by the author during the course of this research or are hypothetical investigation scenarios. In the case of the former category, irrespective of their different requirements, the case studies are similar on three accounts; one, all the requirements can be mapped to one or more grid-facilitated CSP-specific services; two, all of them involve the integration of a MCS or a DES CSP with desktop grid middleware; three, the simulations have been created by simulation users in industry. The hypothetical case studies also involve the integration of a MCS or DES CSP with a grid middleware, map into one or more grid-facilitated CSP services and the simulations that are used are created by OR/MS researchers, CSP vendors or simulation users in industry. However, the hypothetical case studies have been specifically targeted at middleware that have not been used for real-world investigation scenarios.

As has been listed in table 28 (chapter 4), parallel simulation service cannot be used by the CSPs, although the Condor parallel universe execution environment can potentially support it, because the underlying packages do not presently have parallel implementations. It has also been discussed that none of the middleware that have been investigated in this research can presently support collaboration service. Similarly, it has been discussed that web-based simulation service can presently be supported only through the use of grid portals. An abridged version of Table 28 is presented in the next page (table 30) that shows grid middleware support for the remaining four CSP-specific services. The sections in this thesis which discussed these services in relation to the middleware, and which in turn formed the basis of the information provided in table 30, are also indicated.

Table 30: CSP-specific services that can be potentially implemented

CSP-specific service	Grid Middleware	Comments
Task farming service (both MMD and SMMD variants)	<ul style="list-style-type: none"> ▪ BOINC (section 3.4.2) ▪ Condor Java universe (section 3.4.2) ▪ Condor MW (section 3.4.2) ▪ WinGrid (section 4.5.2) ▪ WinGrid-WS (section 4.5.2) 	Condor MW cannot use the Condor java universe execution environment. WinGrid-WS presently supports only SMMD task farming
Workflow service	<ul style="list-style-type: none"> ▪ Condor DAGMan with Condor Java universe (section 3.4.3) ▪ WinGrid (section 4.5.3) 	None
Distributed simulation service	<ul style="list-style-type: none"> ▪ BOINC with HLA-RTI (section 3.4.5.1) ▪ Condor Java Universe with HLA-RTI (section 3.4.5.2) ▪ WinGrid with HLA-RTI (section 4.5.5) 	HLA-RTI distributed simulation middleware will also have to be used
Web-based simulation service (grid portals)	<ul style="list-style-type: none"> ▪ WinGrid-WS (section 4.5.6) 	None

As can be seen in the table above, the potential grid middleware support for CSP-specific services vary from service to service. The CSP-GC framework investigation scenarios are implemented using a subset of these middleware. The middleware that is selected for CSP-grid integration is based on the following considerations:

- For real-world case studies, access to computing resources that are necessary for installing a desktop grid middleware. For example, for installing BOINC middleware access to at least one UNIX / Linux PC is mandatory.
- For real-world case studies, the security restrictions in place within the organization have played a key role in the selection of a middleware. For example, there may be a restriction on the number of communication channels that can be opened by the middleware, the file transfer mechanisms, etc.
- For real-world case studies, the flexibility offered by a grid middleware to implement the problem solution.
- The author's involvement in the *GridAlliance* project has played a part in the selection of middleware for the case studies. One of the primary motivations for developing WinGrid was to facilitate quick implementation of *GridAlliance* demonstration applications, as it was thought that source code control over the middleware would be an advantage.
- The middleware support for CSP-grid integration technology (section 4.4) that have been used in this research.
- Finally, all four grid middleware have been used in at least one case study to realize a grid-facilitated CSP service.

The case studies that are used for evaluation of CSP-GC framework are highlighted with a gray background in table 31. Each case study is identified by a name that is presented in capitalized bold letters. The DES or MCS CSPs which have been used together with the grid middleware are also indicated. The hypothetical case studies are marked as *[hyp]*. The table

also lists the evaluation criteria for each case study, which is in turn based on the CSP-specific service the case study is grouped under. However, in the case of task farming service, the evaluation of the case studies are based on a subset of the task farming evaluation criteria (total of three criteria, highlighted in table 29). The reason for this is discussed next.

The three case study evaluation criteria for task farming service are – (1) the solution is implementable and the service is realizable, (2) grid execution over dedicated nodes is faster compared to a standalone execution and (3) grid execution over non-dedicated nodes is faster compared to a standalone execution. There are a total of four case studies associated with SMMD and MMMD variants of task farming that use three different grid middleware (see table 31). The hypothetical Condor case study evaluates MMMD task farming, whereas the other three case studies, viz., the hypothetical BOINC case study, the Ford case study and the Investment bank case study, implement SMMD task farming. The evaluation criterion for the Condor and BOINC case studies is that it should be possible to implement the CSP-grid integration solution. Here, execution speed is not considered because, firstly, both the case studies are hypothetical, and, secondly, the other two real-world task farming case studies present an experimental evaluation of execution speed. The Ford case study uses dedicated WinGrid nodes for this evaluation. The Investment bank case study, on the other hand, uses non-dedicated WinGrid nodes.

Table 31: Case studies

CSP-specific service	Grid middleware	Case study	MCS / DES CSP used	Case study evaluation criteria / Comments
Task farming service (both MMMD and SMMD variants)	▪ BOINC	BOINC CASE STUDY [hyp] (SMMD task farming)	Microsoft Excel (MCS CSP)	▪ Solution is implementable and the service is realizable
	▪ Condor Java universe	CONDOR CASE STUDY [hyp] (MMMD task farming)	Microsoft Excel (MCS CSP)	▪ Solution is implementable and the service is realizable
	▪ Condor MW	No case study-based investigation	None	Could not be applied to real-world "ford case study" and "investment bank case study" because of security concerns related to Condor middleware, viz., opening multiple ports.
	▪ WinGrid	FORD CASE STUDY (SMMD task farming)	Witness (DES CSP)	▪ Solution is implementable and the service is realizable ▪ Execution is faster over dedicated grid resources compared to a standalone execution
		INVESTMENT BANK CASE STUDY (SMMD task farming)	Analytics (MCS CSP)	▪ Solution is implementable and the service is realizable ▪ Execution is faster over non-dedicated grid resources compared to a standalone execution
	▪ WinGrid-WS	No case study-based investigation	None	Could not be applied to real-world "investment bank case study" because WinGrid-WS does not support workflows.

CSP-specific service	Grid middleware	Case study	MCS / DES CSP used	Case study evaluation criteria / Comments
				WinGrid-WS was, however, deployed at Ford to support SMMD task farming (discussed later).
Workflow service	<ul style="list-style-type: none"> ▪ Condor DAGMan 	No case study-based investigation	None	Could not be applied to real-world “investment bank case study” because of security concerns related to Condor middleware, viz., opening multiple ports.
	<ul style="list-style-type: none"> ▪ WinGrid 	INVESTMENT BANK CASE STUDY	Analytics and Excel (MCS CSPs)	<ul style="list-style-type: none"> ▪ Solution is implementable and the service is realizable
Distributed simulation service	<ul style="list-style-type: none"> ▪ BOINC with HLA-RTI 	No case study-based investigation	None	WinGrid was chosen over BOINC because of the author’s involvement with GridAlliance project.
	<ul style="list-style-type: none"> ▪ Condor Java Universe with HLA-RTI 	No case study-based investigation	None	WinGrid was chosen over Condor because of the author’s involvement with GridAlliance project.
	<ul style="list-style-type: none"> ▪ WinGrid with HLA-RTI 	UK NATIONAL BLOOD SERVICE (NBS) CASE STUDY [hyp]	Simul8 Professional (DES CSP)	<ul style="list-style-type: none"> ▪ Solution is implementable and the service is realizable ▪ Execution is faster over dedicated grid resources compared to a standalone execution
Web-based simulation service (grid portals)	<ul style="list-style-type: none"> ▪ WinGrid-WS 	No case study-based investigation	None	WinGrid-WS was deployed at Ford to support web-based access to SMMD task farming (discussed later).

The table above shows that there are a total of five different case studies. The investment bank case study is grouped under two CSP-specific service categories, namely, task farming service and workflow service. Since there are three more case studies that evaluate the potential of grid middleware to offer task farming service, this case study will mainly concentrate on workflow service. This case study will also evaluate WinGrid’s support for task farming service over non-dedicated grid nodes.

Web-based simulation service (through use of grid portal) is the only CSP-specific service that has not been included in a case study investigation, although it has been identified that at least one middleware (in this case WinGrid-WS) can potentially support it. This is because grid portals usually provide a higher level service to the other CSP-specific services. Thus, a simulation user can potentially use a grid portal to upload simulation models, experiment parameters files, etc., to conveniently access the other CSP-specific services. This means that the other CSP-specific services like task farming service, distributed simulation service, etc., should ideally be implementable before grid portals can be used to support these services. Thus, it was considered important to first investigate the other services in the context of different case studies. However, although no WinGrid-WS specific case study has been presented in this thesis, WinGrid-WS has been implemented to support SMMD task

farming and web-based simulation using grid portals and was deployed in Ford (Dunton Technical Centre, Basildon, Essex). This work is reported in Alstad (2006) and Mustafee et al. (2006a). It has to be added here that WinGrid-WS was conceived because WinGrid, although having earlier demonstrated the viability of SMMD task farming in Ford (this case study is presented later), could not be deployed due to security concerns associated with its multiple server socket-based “push” implementation of job scheduling. However, as things stand today (May 2007), WinGrid is again being considered for production deployment at Ford.

The sections that follow will investigate the following five case studies.

- BOINC case study with PRC middleware BOINC and MCS CSP Excel to evaluate middleware support for SMMD task farming service (section 5.4).
- Condor case study with EDGC middleware Condor and MCS CSP Excel to evaluate middleware support for MMMD task farming service (section 5.5).
- Ford case study with EDGC middleware WinGrid and DES CSP Witness to evaluate middleware support for SMMD task farming service (section 5.6).
- Investment bank case study with EDGC middleware WinGrid and MCS CSP Analytics and Excel to evaluate middleware support for workflow service and SMMD task farming service (section 5.7).
- The UK National Blood Service (NBS) case study with EDGC middleware WinGrid and DES CSP Simul8 Professional to evaluate middleware support for distributed simulation service. The distributed simulation middleware that will be used is HLA-RTI (section 5.8).

5.4 BOINC case study for evaluation of SMMD task farming service

This case study investigates whether BOINC can provide SMMD task farming service to the CSPs. A financial model created using MCS CSP Excel is used to experimentally evaluate whether a BOINC-CSP solution is implementable in practice. Table 32 below shows the case study evaluation criteria. As has been stated earlier in section 5.4, the execution speed evaluation criteria is not considered for this case study because it is a hypothetical case study, and furthermore, two other real-world case studies (sections 5.6 and 5.7) evaluate the speed of execution of a batch of simulation experiments over both dedicated and non-dedicated grid nodes. Some performance results for the BOINC case study can however be found in the **Appendix C**, but it will not be included in subsequent discussions.

Table 32: BOINC case study

CSP-specific service	Grid Middleware	MCS / DES CSP used	Case study evaluation criteria
SMMD task farming service	BOINC	Microsoft Excel (MCS CSP)	(1) Solution is implementable and the service is realizable

This section is structured as follows. Section 5.4.1 highlights the importance of using a PRC middleware BOINC in an enterprise setting. Section 5.4.2 then describes the Excel-based MCS application used for the investigation of this case study. This is followed by a technical

discussion on how the application was grid-enabled (section 5.4.3). This section concludes with the evaluation of BOINC with regards to its potential for providing the SMMD task farming service (section 5.4.4).

5.4.1 Overview

BOINC middleware is primarily used for scientific computing using millions of volunteer PCs. However, as discussed in section 2.9.3, it should also be possible to use the PRC middleware within an organization for the processing of enterprise applications. Using a hypothetical case study, this research now investigates how BOINC can be used in a desktop grid environment to provide the SMMD task farming service to the CSPs. This work has been done together with J. Zhang, J. Saville and S.J.E Taylor and has been accepted for publication. Arguably, this is the first attempt to use a PRC middleware in an enterprise environment. There are no existing examples of enterprise application processing using BOINC in literature.

This research is important because BOINC was architecturally designed for large-scale, redundant computing using PCs that are intermittently connected to the Internet. In an enterprise the PCs are usually centrally managed (thereby eliminating the need for redundant computing) and enjoy dedicated connectivity to the corporate LAN. Furthermore the processing requirements for enterprise applications may not be large scale (compared to scientific application processing), but there may be multiple concurrently executing enterprise applications. These applications may be frequently added, updated and deleted. These differing characteristics of PRC and enterprise processing make this investigation interesting because an attempt is being made to use a middleware designed for the former to be used by the latter.

5.4.2 Range Accrual Swap (RAS) application

The application that is used to implement SMMD task farming using BOINC is a Microsoft Excel-based spreadsheet application used for financial modelling by a leading European financial institution. Microsoft Excel provides support for running MCS and is therefore considered as a MCS CSP. The financial model calculates the risk of a *Range Accrual Swap* at various points in time until the maturity of the transactions. Range Accrual Swap is a type of financial derivative instrument in which certain fixed cash flows are exchanged for an uncertain stream of cash flows based on the movement of interest rates in the future. The model requires the estimation of future uncertain cash flows through simulation of the interest rate curve using a standard stochastic process. The implied cash flows, based on the evolved interest rates, are used to determine the value of the instrument at present date and in the future. The possible values of the instrument in the future enable this financial institution to determine the risk for its client on account of these transactions. A screenshot of the RAS application is shown in screenshot 6.

Range Accrual Swap PFE model		Simulate Fixed Rate	Results	Simulated df curve								
Inputs/ Termsheet				Time step (yrs)	0	1	2	3	4	5	6	7
5	Number of Monte-Carlo Simulation Iterations	10	Iteration #	Y(0)	-0.692	-2.551	-0.483	0.907	1.194	-0.245	-1.188	
7	Currency	USD		0	0.000	1.0000	1.0000	1.0000	1.0000	1.0000	1.0000	1.0000
8	Notional Amount	10,000,000		1	0.003	0.9999	1.0000	1.0000	0.9999	0.9999	0.9999	0.9999
9	Start Date	31-Jul-06		2	0.005	0.9999	0.9999	1.0000	0.9998	0.9997	0.9997	0.9998
10	Tenor (yrs)	13		3	0.008	0.9998	0.9998	0.9999	0.9998	0.9996	0.9996	0.9997
11	Reset/ Payment freq	3 months		4	0.011	0.9998	0.9998	0.9999	0.9997	0.9995	0.9994	0.9995
12	Fixed Rate	6.7%		5	0.014	0.9997	0.9997	0.9999	0.9996	0.9994	0.9993	0.9995
13	Pay/Receive Fixed	Pay		6	0.016	0.9997	0.9997	0.9999	0.9995	0.9992	0.9991	0.9993
14	Underlying Libor	6 months		7	0.019	0.9996	0.9996	0.9999	0.9994	0.9991	0.9990	0.9992
15	N = total number of calendar days in the interest period			8	0.022	0.9996	0.9996	0.9998	0.9993	0.9990	0.9988	0.9991
16	n = the number of calendar days in the interest period that the 12 month			9	0.025	0.9995	0.9995	0.9998	0.9992	0.9988	0.9987	0.9990
17	USD LIBOR rate is within the following ranges at Fixing (Inclusive)			10	0.027	0.9995	0.9994	0.9998	0.9992	0.9987	0.9985	0.9989
18	Day Count	Act / 365		11	0.030	0.9994	0.9994	0.9998	0.9991	0.9986	0.9984	0.9988
19	Year	Range	Coupon	12	0.033	0.9993	0.9993	0.9997	0.9990	0.9984	0.9983	0.9986
20	1	0% - 7.00%	7.0%	13	0.036	0.9993	0.9993	0.9997	0.9989	0.9983	0.9981	0.9985
21	2	0% - 7.00%	7.0%	14	0.038	0.9992	0.9992	0.9997	0.9988	0.9982	0.9980	0.9984
22	3	0% - 7.00%	7.0%	15	0.041	0.9992	0.9992	0.9997	0.9987	0.9980	0.9978	0.9983
23	4	0% - 7.00%	7.0%	16	0.044	0.9991	0.9991	0.9997	0.9986	0.9979	0.9977	0.9982
24	5	0% - 7.00%	7.0%	17	0.047	0.9991	0.9990	0.9996	0.9986	0.9978	0.9975	0.9981
25	6	0% - 7.00%	7.0%	18	0.049	0.9990	0.9990	0.9996	0.9985	0.9977	0.9974	0.9980
26	7	0% - 7.00%	7.0%	19	0.052	0.9990	0.9989	0.9996	0.9984	0.9975	0.9972	0.9978
27	8	0% - 7.00%	7.0%	20	0.055	0.9989	0.9989	0.9996	0.9983	0.9974	0.9971	0.9977
28	9	0% - 7.00%	7.0%	21	0.058	0.9989	0.9988	0.9995	0.9982	0.9973	0.9970	0.9976
29	10	0% - 7.00%	7.0%	22	0.060	0.9988	0.9988	0.9995	0.9981	0.9971	0.9968	0.9975
30	11	0% - 7.00%	7.0%	23	0.063	0.9988	0.9987	0.9995	0.9980	0.9970	0.9967	0.9974
31	12	0% - 7.00%	7.0%	24	0.066	0.9987	0.9987	0.9995	0.9980	0.9969	0.9965	0.9973
32	13	0% - 7.00%	7.0%	25	0.068	0.9986	0.9986	0.9995	0.9979	0.9967	0.9964	0.9972
33	14			26	0.071	0.9986	0.9985	0.9994	0.9978	0.9966	0.9962	0.9970
34	15			27	0.074	0.9985	0.9985	0.9994	0.9977	0.9965	0.9961	0.9969
35				28	0.077	0.9985	0.9984	0.9994	0.9976	0.9963	0.9959	0.9968
36	Term Structure		HW parameters	29	0.079	0.9984	0.9984	0.9994	0.9975	0.9962	0.9958	0.9967
37	Tenor	df	alpha	30	0.082	0.9984	0.9983	0.9993	0.9974	0.9961	0.9956	0.9966
38	0.00	1.000	0.0650	31	0.085	0.9983	0.9983	0.9993	0.9974	0.9960	0.9955	0.9965
39	0.08	0.998	sigma	32	0.088	0.9983	0.9982	0.9993	0.9973	0.9958	0.9954	0.9964
40	0.25	0.995	1.00%	33	0.090	0.9982	0.9981	0.9993	0.9972	0.9957	0.9952	0.9963

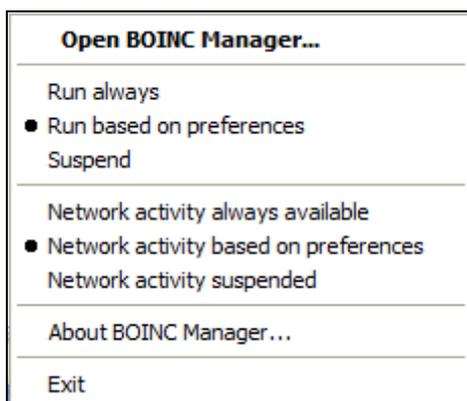
Screenshot 6: Range Accrual Swap (RAS) application

The successful and accurate calculation of risk using the RAS application requires a large number of MCS and takes a significant amount of time. Each simulation run (iteration) is independent of previous runs and is characterized by the generation of random values for various defined variables and by solving equations containing these variables. The conventional approach of using only one instance of Microsoft Excel is not feasible in situations where the business desires a quick turnaround (answer). One solution to this is to distribute the processing of the MCS model over a grid and utilize the spare processing power of the grid nodes and the Excel software installed on them. Thus, if the RAS model requires 100,000 iterations and the grid infrastructure consists of 10 dedicated grid nodes, then it should be possible to assign each node to run 10,000 (100,000/10) iterations instead of using only one computer to run all of the 100,000 iterations. In order to arrive at the final values of the multiple result sets returned by the different grid nodes (10 in this case) can be calculated. This grid-facilitated execution of the RAS model has the potential of speeding up the simulation of the financial models manifold, depending on the number of grid nodes available and whether they are dedicated or non-dedicated resources.

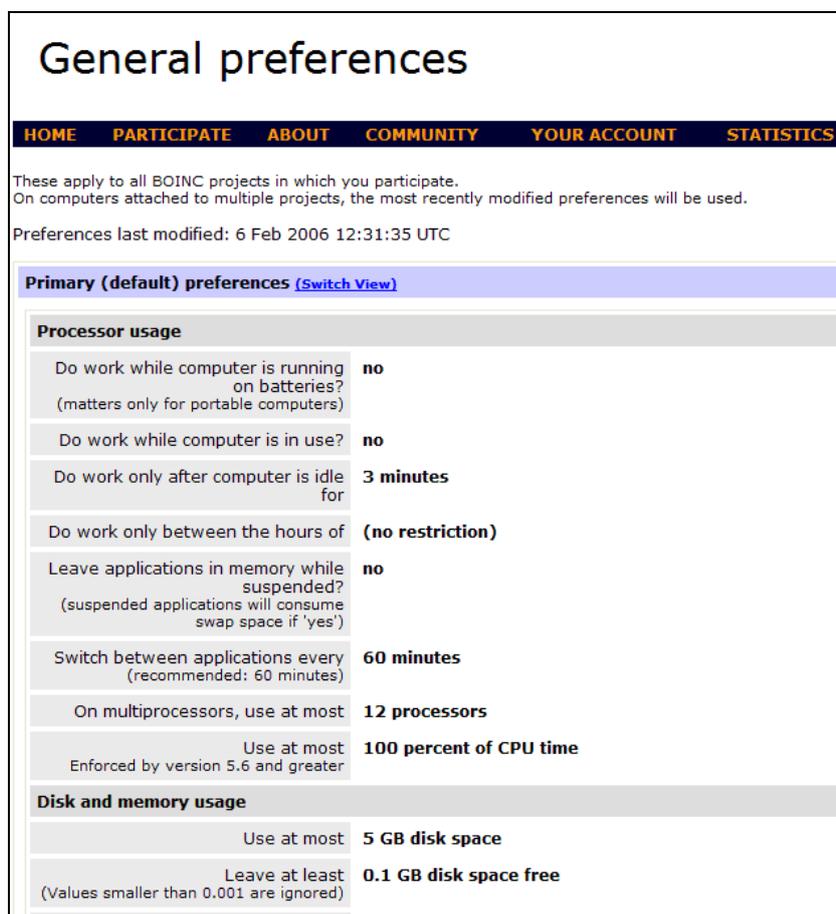
5.4.3 Grid-enabling RAS application

A BOINC-based project requires application specific implementation on both the client side and the server side. The client side implementation usually consists of writing a C++ application client that uses BOINC client library and APIs to integrate with the BOINC core client. This is illustrated in figure 23 in section 2.9.2. The core client is downloaded from the BOINC website, installed on individual PCs and is attached to a BOINC project. Once successfully attached the core client downloads the project specific application client and

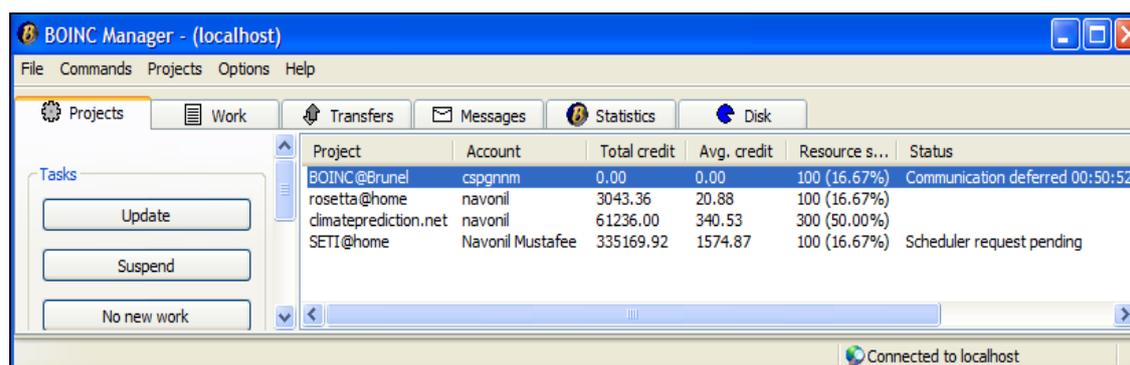
work units for processing. The core client, which is in effect the manager of a compute resource, makes available CPU cycles to the attached project based on the user's preferences. These preferences can be set using either the menu provided by the core client (screenshot 7) or through a web interface (screenshot 8). The latter offers the user more flexibility in specifying CPU, memory, disk and network usage. The core client can support multiple BOINC-based projects, but at any one time only one project can be executed. This is illustrated in screenshot 9 (next page) where four different BOINC projects, viz, BOINC@Brunel, Rosetta@Home, ClimatePrediction.net and SETI@home, are attached but only one project (SETI@home) is communicating with the BOINC server side scheduler.



Screenshot 7: Setting user preference using menu provided by BOINC core client



Screenshot 8: Setting user preference using web interface



Screenshot 9: BOINC core client attached to multiple projects

Section 2.9.3 has distinguished between two different types of BOINC application clients: *runtime application client* (BOINC-RAC) and *proxy application client* (BOINC-PAC). BOINC-PAC is used for this case study which assumes that Microsoft Excel is installed on the BOINC client computers.

The BOINC-PAC for the RAS case study is implemented in Visual C++. The BOINC-CSP integration design is similar to that presented for WinGrid-CSP integration (section 4.4), the difference however is that the former uses Visual C++ and the latter Java for invoking CSP-specific operations defined by the Visual Basic DLL adapter. Also, since Visual C++ code can directly invoke VB DLLs there is no need for JNI in the case of the BOINC-PAC.

BOINC-PAC uses the BOINC client library and APIs to interface with the BOINC core client. It interacts with the *Excel adapter* to execute operations on the RAS Excel-based spreadsheet. The Excel adapter, in turn, uses the COM interface of Excel to perform basic operations like opening and closing the simulation file, setting the number of iterations, executing the simulation and writing the results of the simulation to a text file (*out.txt*). The text file is subsequently uploaded to the BOINC server. The interaction of the different program components is shown in figure 38. Once the BOINC-PAC is downloaded by the core client onto a PC it triggers the execution of the RAS MCS by utilizing the Excel software installed on the local resource. Unlike most BOINC-based PRC projects where the entire executable required to process data is downloaded to a PC, this approach only downloads the proxy application code (executable C++ file and the Excel adapter) and uses enterprise software (in this case Excel) to process the jobs on the grid nodes. Arguably, this not only maximizes an enterprise's return on investment (ROI) on computing resources but also for the software that has been purchased.

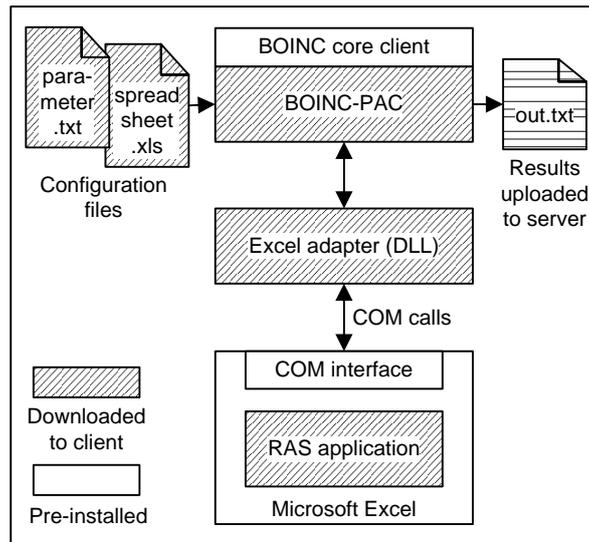


Figure 38: Execution of RAS application using BOINC

The number of Monte Carlo iterations to be performed by the RAS application is not hard-coded and is read by BOINC-PAC from a parameter file (parameter.txt in figure 38 above). This file contains a list of values separated by a space. Each value represents the number of iterations that need to be performed on an experiment being executed on a BOINC node. The position of the value in the parameter file indicates the experiment number. Thus, if the parameter file contains the following five values, viz., 10000 5000 7000 1000 3000, then the first simulation experiment will do 10000 iterations, the next experiment will perform 5000 iterations and so on. In this case study a constant value of 300 has been used for all the 200 experiments that have been conducted. The parameter file therefore contains the value 300, repeated 200 times, each separated by a space. It is arguable that the iteration value (300) could have been hard coded for this case study. However, a parameter file has still been used because it provides a mechanism that allows the passing of different arguments to a simulation. This would allow BOINC to be used for SMMD task farming using DES CSPs, wherein different arguments for the DES models need to be passed to the clients for simulating different experiments.

The discussion that follows mainly concerns the BOINC server side implementation for the RAS application. When the BOINC core client first attaches itself with the RAS project it downloads the BOINC-PAC from the BOINC server. This application consists of a VC++ executable and a client initialization file called *init_data.xml*. Subsequently, the core client downloads the project workunits. In BOINC one unit of computation is represented as a workunit. These workunits are created using the BOINC *create_work* command and then placed in the download directory of the BOINC server. The arguments supplied to the *create_work* command include, among others, (1) the workunit template filename, (2) the result template filename and (3) the *command_line* parameter. The template files are XML files that describe the workunit (*work_unit_template.xml*) and its corresponding results (*result_template.xml*). The workunits are created by running a program that invokes the

create_work command in a loop to generate the required number of workunits. The arguments to the *create_work* command are described next.

- The “workunit template file” lists the input files that are packed together as a workunit. In the RAS BOINC project the input files are the RAS Excel-based spreadsheet, the Excel adapter, and the parameter file. The workunit template file also mentions the quorum (XML tag *<min_quorum>*) and the maximum total results (XML tag *<max_total_results>*). However, since BOINC is being used in an enterprise grid environment that assumes some form of centralized control over the computing resources, the value for both *<min_quorum>* and *<max_total_results>* are set to one. In other words, it is expected that all the results that are returned are valid and therefore the same workunit will not be sent to more than one BOINC node.
- The “result template file” lists the files that will be uploaded to the BOINC server after the results have been computed by the BOINC-PAC. In the RAS application, the file that is uploaded from each BOINC client is called *out.txt*. As has been said earlier, this file contains the results of the RAS simulation.

It has to be added here that a better implementation was to include the RAS application, Excel adapter and the parameter file with the BOINC-PAC itself, because including these files as part of a workunit suggests that they will be downloaded whenever the BOINC core clients request a new workunit. This is not required for the RAS application because all the files are identical. However, it has been observed that the core client does not download files that are already present in the local computer’s BOINC project directory. Thus, only the *command_line* parameter (see below) is transferred to the BOINC client at each invocation of the workunit request.

- The optional *command_line* argument in the *create_work* command is used to pass a position value to the BOINC-PAC application. This position value represents an experiment number and BOINC-PAC reads the parameter file *parameter.txt* to extract the value at this position. This value, as has been discussed earlier, represents the number of iterations that have to be performed on a simulation experiment being run on the client. The use of the *command_line* argument is specific to the BOINC-PAC application being developed.

To experimentally prove that BOINC can provide SMMD task farming service to CSPs, 200 MCS experiments (each with 300 iterations) were conducted. Thus, the parameter file consisted of 200 consecutive values, each value being 300 and separated by a space. A Java program was used to iteratively create these 200 work units by invoking *create_work* with *command_line* argument (the argument is an integer value which is 1 for the first workunit, 2

for the second workunit and so on). These workunits were downloaded by different BOINC nodes and the RAS application executed using the locally installed MCS CSP Excel. The results of the simulation were then automatically uploaded to the BOINC project server.

The RAS BOINC project had been executed within the confines of the Brunel University firewall. Illustrated in figure 39, it comprised of the following:

- One PC running the CentOS 4.3 operating system with a 864MHz Pentium III processor and 256MB RAM. All BOINC server-side components and the MySQL database were executed here.
- Four laptop computers running the Microsoft Windows XP Professional operating system, each with a 1.73GHz Intel Celeron processors and 1GB RAM. Each laptop was pre-installed with the BOINC client and MCS CSP Excel. One of these laptops had two network cards, and acted as the router between the test LAN and the University LAN. The University LAN was used to access the BOINC server.
- Four low-end desktop PCs running either Microsoft Windows XP Professional or Microsoft Windows 2000, with either Pentium I or Pentium II processors and 128MB or 256MB RAM. Like the laptops, these were pre-installed with BOINC client and MCS CSP Excel.
- The laptops and desktop PCs were connected through a 100Mbps switch to form a private test LAN.

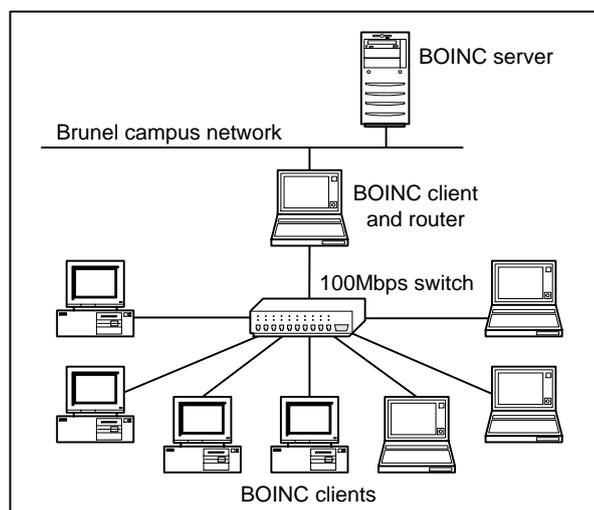


Figure 39: BOINC test network

5.4.4 Evaluation of SMD task farming service

The results of the experiments and related discussions are included in **Appendix C**. For the purpose of evaluating BOINC support for SMD task farming, the only criterion was that the solution is implementable in practise, and thus the task farming service using BOINC is realizable. Through a detailed discussion in the previous section it has been shown that

BOINC can be used for supporting the CSP-specific SMMD variant of task farming service. However, it has to be added that using BOINC in an enterprise setting has its drawbacks. Most of these drawbacks are implementation related. These drawbacks are listed below.

- BOINC requires a UNIX/Linux-based server installation, which may not fit with an enterprise's existing infrastructure or expertise.
- Creation and management of projects on the BOINC server and the operation of the BOINC client presently requires a degree of intervention from the user. This runs counter to the principle of transparent job processing that desktop grids should ideally provide. But it is possible that this burden could be lessened considerably through scripting and automation.
- BOINC clients are designed to pre-fetch workunits from the server so that the execution of BOINC-based applications can continue uninterrupted. However, when a BOINC infrastructure consists of both high and low configuration PCs, as was the case with the BOINC test bed that was used for this case study, workunits can be hoarded by faster running PCs. Essentially hoarding occurs because the BOINC system currently provides no fine control over how many work units are pre-fetched by each client, and thus "fast" clients and "slow" clients both pre-fetch multiple work units. If the work units are relatively large-grained, the fast clients may "run dry" before the slow clients have finished processing the first of their work units.

The reader is reminded that one of these drawbacks, namely, the requirement for a PC running UNIX/Linux operating system, has already been discussed in section 3.6. The discussions in this section have shown that BOINC is not an ideal middleware implementation for CSP-based simulation in industry. The other two drawbacks that have been highlighted in this section further adds to this argument.

5.5 Condor case study for evaluation of MMMD task farming service

The Condor case study evaluates the EDGC middleware Condor in relation to its potential to support the MMMD variant of task farming service. Table 33 below summarises the technologies used and the case study evaluation criteria.

Table 33: Condor case study

CSP-specific service	Grid Middleware	MCS / DES CSP used	Case study evaluation criteria
MMMD task farming service	Condor Java universe	Microsoft Excel (MCS CSP)	(1) Solution is implementable and the service is realizable

An overview of the case study is presented in section 5.5.1. MMMD task farming necessitates that two or more models are used for concurrent execution over the grid. This investigation, therefore, attempts to grid-enable two different applications – the Asian Option application (section 5.5.2) and the Range Accrual Swap application (section 5.5.3) – with the objective of

executing them concurrently using Condor middleware. Section 5.5.4 then discusses the technology used to grid-enable both these Excel-based applications. This is followed by an evaluation of Condor with regards to its suitability for supporting the CSP-specific MMMD task farming service (section 5.5.5).

5.5.1 Overview

The Multiple Model Multiple Data (MMMD) variant of task farming has the potential to execute different CSP models, which may belong to different simulation users, simultaneously over the grid. Furthermore, these models may be created and executed using different MCS and DES CSPs. However, in this hypothetical case study, models created using the same MCS CSP (Microsoft Excel) are used. The first model is called the Asian Option application which has been created by Professor Eduardo Saliby (Federal University of Rio de Janeiro, Brazil; visiting professor at Brunel University, UK). The second model is the RAS application that has been previously used in the BOINC case study. The RAS model has been created by the credit risk division of a major investment bank. The evaluation criterion for this case study is that the Condor-CSP solution for supporting MMMD task farming is implementable in practise. This would also mean that the CSP-specific task farming service is realizable with Condor.

5.5.2 Asian Options (AO) application

The Asian Options Application uses *Descriptive Sampling*, which can be seen as a variance reduction technique, to calculate options whose payoffs are path-dependent on the underlying asset prices during the life of the option (Marins et al., 2004). Variance reduction techniques are procedures that produce more precise estimates without a corresponding increase in computing effort (Nelson, 1987). Descriptive sampling achieves this goal through a fully deterministic selection of a simulation model's input variable values and the random permutations of those values (Saliby, 1997). Screenshot 10 shows the Microsoft Excel-based AO application.

The AO application estimates the value of the Asian options by simulating the model a number of times and then calculating the average of the results of the individual iterations. On a single PC, executing multiple iterations of the AO application takes a significant amount of time. CSP-specific task farming service has the potential to reduce the time taken to process the AO application by distributing its processing over multiple grid nodes. An average of the results returned from each node can then be calculated to determine the value of the options.

5.5.3 Range Accrual Swap (RAS) application

The RAS application has already been described in section 5.4.2. The application is the same but the technologies used for interfacing RAS with BOINC and RAS with Condor are different. The integration of RAS with BOINC has been discussed in section 5.4.3. The section that follows describes how both RAS and AO are used with the Condor Java universe execution environment.

	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	
1																				
2		Asian Call Option						2.75364	20%	30%	40%		0.48779	20%	30%	40%				Ant. M
3							40	15.0937	14.9877	14.8939		40	1.0000	0.9949	0.9695			K	2.500	
4		Inputs		Black-Scholes			45	10.1754	10.1730	10.3141		45	0.9936	0.9461	0.8754			45	10.47	
5		S_0	55.00	d_1	0.0919		50	5.4615	5.8785	6.4179		50	0.8845	0.7726	0.6940			50	6.248	
6		K	55.00	d_2	-0.0306		52	3.8297	4.4592	5.1372		52	0.7635	0.6654	0.6052			52	4.322	
7		Sigma	30%	N(d_1)	0.5366		54	2.4809	3.2635	4.0344		54	0.6008	0.5474	0.5143			54	3.228	
8		T	126	N(d_2)	0.4878		55	1.9325	2.7536	3.5499		55	0.5122	0.4878	0.4695			55	3.303	
9		R_f	3%	C(S,T)	2.7536		56	1.4714	2.3020	3.1090		56	0.4246	0.4295	0.4259			56	2.861	
10		a	0.0075				58	0.7947	1.5647	2.3520		58	0.2677	0.3212	0.3439			58	1.564	
11		sig_a	0.1732				60	0.3902	1.0254	1.7478		60	0.1503	0.2293	0.2711			60	1.184	
12							62	0.1742	0.6485	1.2766		62	0.0754	0.1565	0.2069			62	0.673	
13						Run	64	0.0710	0.3963	0.9174		64	0.0340	0.1024	0.1575			64	0.681	
14							66	0.0265	0.2344	0.6491		66	0.0138	0.0643	0.1164			66	0.377	
15							68	0.0091	0.1344	0.4526		68	0.0051	0.0389	0.0844			68	0.221	
16							70	0.0029	0.0749	0.3113		70	0.0017	0.0228	0.0602			70	0.054	
17																				
18																				
19	Trial	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	
20	1	1	-0.3719	-0.9346	0.1363	0.2404	0.8239	-1.0152	0.4261	0.4538	-2.1701	0.1383	-1.9600	0.5388	-1.1031	-0.1363	-0.4538	-0.9741	-0.51	
21	2	2	0.6588	0.0125	-0.1891	0.8965	0.0125	1.0152	1.0152	-1.9600	0.6588	-0.6280	-0.7554	0.3186	0.2924	0.6280	-0.2663	0.48		
22	3	3	0.1891	0.7554	-0.4261	-0.2404	-1.4395	0.7554	-0.0627	-0.4538	-0.5388	1.4395	0.2404	0.3451	-1.5141	-0.0376	0.2924	-0.5978	0.85	
23	4	4	0.0878	0.3186	-0.6903	0.1891	-0.4817	-0.0125	-0.6903	-0.6903	-0.1130	1.2536	0.2663	1.6954	0.6903	0.5388	-0.9741	-0.0125	1.10	
24	5	5	-0.5101	-0.6588	0.3186	-0.3719	-0.5388	0.1637	0.6280	1.4395	0.5388	0.1891	-0.8596	-0.8239	0.3969	-1.8119	2.1701	-0.4261	-0.53	
25	6	6	0.8239	1.0152	-0.2147	0.0627	1.2536	0.9741	-1.9600	0.5978	0.5978	-0.6588	1.6954	1.1503	-1.2004	1.3106	-1.8119	-0.8596	0.13	
26	7	7	-0.8965	0.2663	0.2663	-0.7225	-0.0878	-0.1383	0.0376	-0.0878	0.1383	0.3451	0.0125	0.8965	-1.3722	-1.1503	-0.1052	-0.3186	1.31	
27	8	8	0.5101	-0.1383	0.9346	1.3106	-0.3989	1.5141	-0.4261	0.1383	0.4261	-0.5101	0.9346	1.2536	0.7225	-0.3451	-0.5681	-0.7554	0.51	
28	9	9	0.3719	0.2924	-0.1637	0.2663	-0.5978	-2.5758	-1.4395	-0.1891	-1.5141	-0.2663	1.3106	-0.9346	1.2004	2.5758	-1.3106	-0.7892	-1.05	
29	10	10	0.1130	-0.4817	-0.1383	-0.1130	0.0627	1.8119	0.6903	1.1031	-0.1891	0.9741	1.9600	-0.2404	-0.6903	0.7892	2.5758	-0.0627	1.05	
30	11	11	-0.9741	-0.9741	-0.5978	-0.7554	0.3186	-0.2924	-0.9741	1.9600	1.3106	-2.5758	-1.0152	0.9741	-1.0581	1.9600	1.9600	0.6280	1.43	
31	12	12	0.8965	1.1503	-0.2924	1.4395	0.9346	0.6588	0.2663	1.5982	-2.5758	-0.0125	1.4395	-1.2536	-0.3186	0.3451	-0.8239	1.3106	-0.45	
32	13	13	-0.7225	1.3722	1.1031	1.9600	1.8119	-0.7225	0.9346	-0.6588	0.1891	0.0376	0.2147	-0.3719	-2.1701	-0.7554	-0.0376	1.8119	0.16	
33	14	14	-0.2404	-0.7225	0.0125	0.7225	-0.1130	-0.1891	-0.9346	0.3719	-0.8596	-2.1701	-0.2404	0.5101	0.9346	-1.4395	1.8119	-1.5882	-0.24	
34	15	15	-1.1503	0.0376	1.8119	-0.6280	0.1891	0.0376	-0.2404	-0.7554	-0.2147	0.5388	1.1503	-1.8119	-1.9600	1.5982	-1.1031	0.7554	-2.57	
35	16	16	-0.0376	0.4261	1.4395	-1.3722	-1.2536	1.4395	-0.2663	0.3451	0.3451	0.3969	1.5982	0.6903	1.1031	-0.3186	-0.7554	-0.3719	0.16	
36	17	17	-1.5141	0.5681	-0.0878	1.3722	0.5388	-1.3722	0.3451	0.9346	-0.0627	-0.5681	-0.9741	1.4395	-1.0152	-0.5978	-0.4817	2.5758	1.15	

Screenshot 10: Asian Options (AO) application

5.5.4 Grid-enabling AO and RAS applications

The Condor Java universe execution environment is designed for the execution of Java programs. The Java-VB DLL based integration technology that has been used previously for WinGrid and Simul8 can therefore be used for executing CSP applications over a Condor pool. The reader is referred to section 4.4 for more information pertaining to the adapter-based approach to CSP-grid integration.

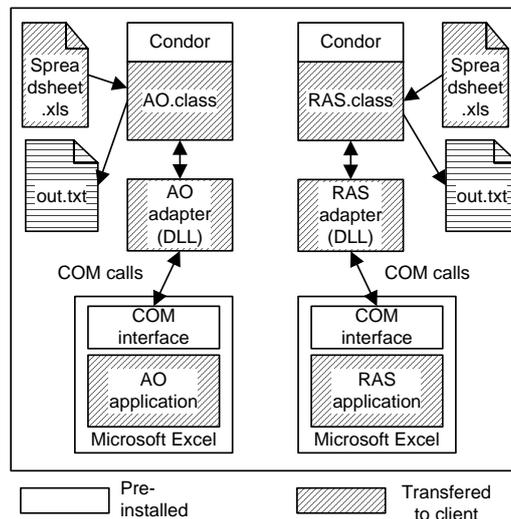


Figure 40: Execution of RAS and AO applications on a Condor pool

Different Java programs (AO.class and RAS.class) and adapters (AO adapter and RAS adapter) have been developed for AO and RAS applications respectively. As shown in figure 40 above, the AO.class/RAS.class communicates with the AO/RAS adapter to control the Excel-based AO/RAS application. The results of the simulation are written back to their

respective out.txt files, which are then transferred back to the Condor node from which the jobs were originally submitted. The figure also shows the files that have been transferred to the remote Condor nodes from the job submission node. Both the AO and RAS applications are executed concurrently over the Condor pool.

The discussion now focuses on the Condor mechanism that allows the submission of multiple jobs. There are two applications in this case study and for supporting MMD task farming it is generally required that it should be possible to submit multiple instances of each application over the Condor pool. The job submission file is used to achieve this. As has been discussed earlier in section 2.10.3 of this thesis, every job has a corresponding job submit file (.sub file) that defines variables that control different aspects of job submission. The most important of these Condor-defined variables, for the purpose of task farming, is the *queue* variable. The integer value assigned to this variable determines the number of replications of the same job that are to be executed over the Condor pool. Screenshots 11 and 12 show the .sub file for the AO and the RAS applications respectively. The value “50” assigned to the *queue* variable (the last variable in the screenshots) suggests that both the AO and the RAS applications will be executed for a total of 50 times over different grid nodes. Some of the other job submission variables shown in the .sub file are discussed next.

```
#####
# Asian Stock Option Monte Carlo simulation
# Submit 10 Monte Carlo simulation jobs to CONDOR.
# Author: Navonil Mustafee
# Date : 25th February' 2007
#####

# submit jobs to CONDOR java universe
universe = java

# The java class class file which will be executed by the JVM
executable = ..\AsianStockOption.class

# The number of Monte Carlo iterations
arguments = AsianStockOption 10

# Setup so each job has its own working directory. The first will have
# a initial working directory of dir.0, the second dir.1, etc.
initialdir = dir1.$(Process)

# The JAVA-COM bridge
jar_files = ..\jacob.jar

# The files that have to be transfered to execution directory of remote syste
transfer_input_files = ..\AsianStockOption.class, ..\jacob.jar, ..\Asian Call

# The output has to be transfered from local execution directory to shared di
when_to_transfer_output = ON_EXIT

# The files will have to be transfered
should_transfer_file = yes

# The console output file name
output = ExcelMonteCarloConsoleoutput_ASO.out.txt

# The error file name
error = ExcelMonteCarloErroroutput_ASO.err.txt

# The log file name
log = ExcelMonteCarloLogoutput_ASO.log.txt

# Say we Never want to receive email about this job...
notification = Never

# copy the user environment into the job's environment
getenv = True

# submit 50 instances of this job!
queue 50
```

Screenshot 11: Job submit file for AO application

```
#####
# Range Accrual Swap Monte Carlo Simulation
# Submit 10 Monte Carlo simulation jobs to CONDOR.
# Author: Navonil Mustafee
# Date : 24th February' 2007
#####
# Submit jobs to CONDOR java universe
universe = java

# The java class class file which will be executed by the JVM
executable = ..\RangeAccrualSwap_ExcelMonteCarloSimulation.class

# The number of Monte Carlo iterations
arguments = RangeAccrualSwap_ExcelMonteCarloSimulation 10

# Setup so each job has its own working directory. The first will have
# a initial working directory of dir.0, the second dir.1, etc.
initialdir = dir.%(Process)

# The JAVA-COM bridge
jar_files = ..\jacob.jar

# The files that have to be transfered to execution directory of remote system
transfer_input_files = ..\RangeAccrualSwap_ExcelMonteCarloSimulation.class, ..\jacob.jar,

# The output has to be transfered from local execution directory to shared dir.
when_to_transfer_output = ON_EXIT

# The files will have to be transfered
should_transfer_file = yes

# The console output file name
output = ExcelMonteCarloConsoleoutput.out.txt

# The error file name
error = ExcelMonteCarloErroroutput.err.txt

# The log file name
log = ExcelMonteCarloLogoutput.log.txt

# Say we Never want to receive email about this job...
notification = Never

# copy the user environment into the job's environment
getenv = True

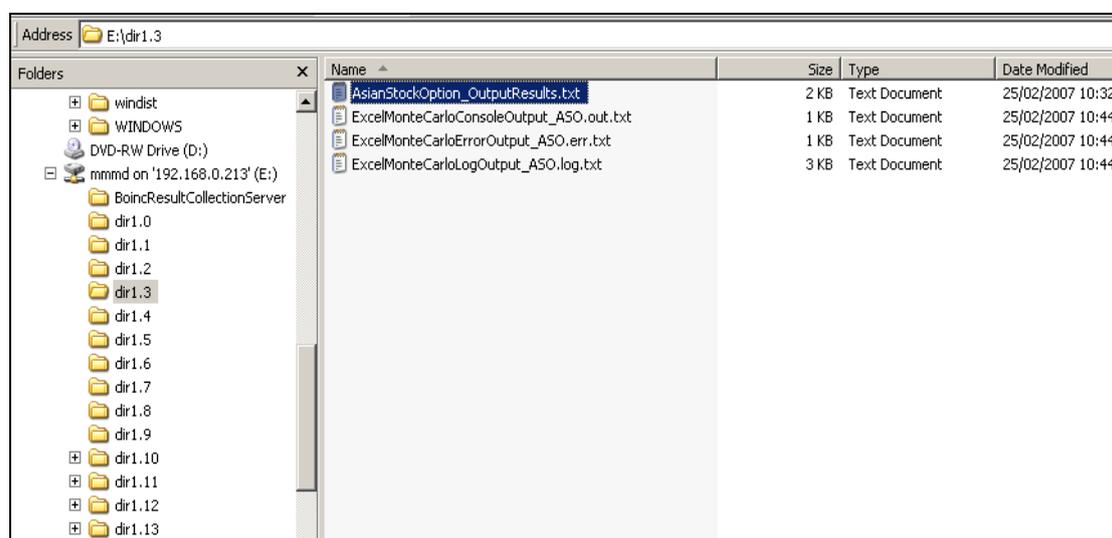
# Submit 50 instances of this job!
queue 50
```

Screenshot 12: Job submit file for RAS application

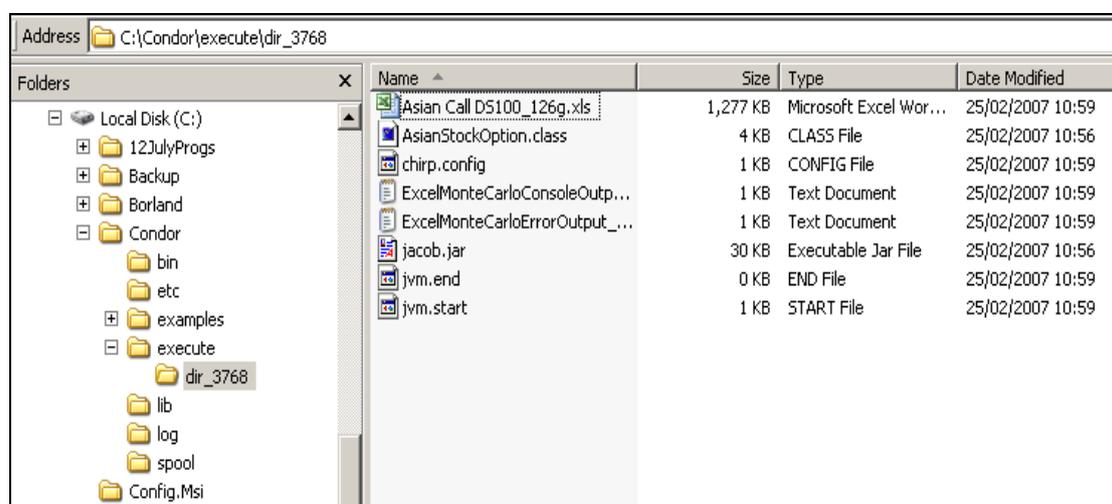
The *universe* variable is assigned a value “Java” because the Condor Java execution environment is being used to run the simulations. The *executable* variable defines the name of the Java class file that has the *main()* method. The reader may notice that the name of the .class files for AO and RAS applications are not AO.class and RAS.class (as shown in figure 40) but *AsianStockOption.class* and *RangeAccrualSwap_ExcelMonteCarloSimulation.class* respectively. The *argument* variable is used to pass a command line argument to the Java program. For this hypothetical case study, the number of iterations for each simulation model has been set to a modest value of “10” through the use of this *argument* variable. The reader is however reminded that both AO and RAS applications will be executed 50 times over, and therefore the total number of simulation iterations for each application, taken as a whole, will be 500 (50*10).

Each simulation experiment will have a unique working directory associated with it. These directories should be present on the Condor node from which jobs are submitted, or on a network drive that can be accessed by the job submission node. The working directories are represented by the variable *initialdir*. In the case of the AO and the RAS applications the values assigned to this variable are “dir1.%(process)” and “dir.%(process)” respectively.

$\$process$ is a Condor-defined integer variable that is incremented automatically depending on the number of instances of a particular job that have been submitted. Thus, if $queue=50$ then the value of the $\$process$ variable will start from 1 and will end at 50. This in turn suggests that the working directory for the first job will be “dir1.1” and for the last job it will be “dir1.50” (in case of AO application). These working directories are important because they will contain the results of the individual experiments and the log files that are output by Condor during execution of each experiment (screenshot 13). The variables that define the names of the three different Condor log files for console output, error information and Condor-specific messages are *output*, *error* and *log* respectively. It has to be added, however, that a Condor job is in-effect executed under a temporary directory that it created by Condor on the grid node that is assigned the task of processing the job (screenshot 14 shows a temporary directory called “dir_3768” that has been created for executing one instance of a simulation). Once the simulation is complete, the results from the temporary directory are transferred to the individual working directories and the temporary directory deleted.



Screenshot 13: Results from the simulation experiments



Screenshot 14: Condor jobs getting executed in temporary execution directory

The files to be transferred to the execution host are indicated by the *transfer_input_files* variable. These files are transferred to the temporary execution directory created by the job executing node. The variable *when_to_transfer_output* and its corresponding value "ON_EXIT" suggest that the simulation results (and the Condor log files) are transferred back from the temporary execution directory to their respective working directories. This concludes the discussion on the variables defined in the Condor submit files.

Jobs are submitted for execution using the Condor command *condor_submit*. The argument to this command is the job description file associated with each job. Screenshot 15 below shows that .sub files for both the AO application (aso.sub) and the RAS application (ras.sub) are submitted using this command, and that 50 instances of each application are created automatically by Condor (see message: "50 jobs(s) submitted to cluster 109/110"). Once the jobs have been submitted the status of the Condor pool can be determined using the command *condor_status*. Screenshot 15 shows that at present three grid nodes (computers with names 210-A, 214-E and 215-F) are executing the jobs that have been submitted (Activity="Busy"), while the remaining are "Idle". However, all the nodes have been claimed by Condor (State="Claimed") and it is expected that these will soon start executing the simulations.

```
E:\>condor_submit aso.sub
Submitting job(s).....
Logging submit event(s).....
50 job(s) submitted to cluster 109.

E:\>condor_submit ras.sub
Submitting job(s).....
Logging submit event(s).....
50 job(s) submitted to cluster 110.

E:\>condor_status
```

Name	OpSys	Arch	State	Activity	LoadAv	Mem	ActvtyTime
217-H	WINNT50	INTEL	Claimed	Idle	0.130	255[?????]	
210-A	WINNT51	INTEL	Claimed	Busy	0.060	1023	0+00:00:56
211-B	WINNT51	INTEL	Claimed	Idle	0.010	1023	0+00:00:21
212-C	WINNT51	INTEL	Claimed	Idle	0.000	1023	0+00:00:16
213-D	WINNT51	INTEL	Claimed	Idle	0.010	1023	0+00:00:09
214-E	WINNT51	INTEL	Claimed	Busy	0.450	255[?????]	
215-F	WINNT51	INTEL	Claimed	Busy	0.680	255[?????]	

Machines	Owner	Claimed	Unclaimed	Matched	Preempting
INTEL/WINNT50		1	0	1	0
INTEL/WINNT51		6	0	6	0
Total		7	0	7	0

Screenshot 15: AO and RAS applications execution over Condor pool

The status of jobs that have been submitted can be found using the command *condor_q*. However, only jobs that are yet to be completed or are presently running are displayed by this command (screenshot 16). The jobs that have been completed are not shown.

```
E:\>condor_q
-- Submitter: 210-A : <192.168.0.210:1040> : 210-A
ID      OWNER      SUBMITTED      RUN_TIME ST  PRI  SIZE  CMD
110.32  Admin      2/25 10:41     0+00:01:48 R  0    0.0  java RangeAccrual$
110.37  Admin      2/25 10:41     0+00:00:43 R  0    0.0  java RangeAccrual$
110.38  Admin      2/25 10:41     0+00:00:25 R  0    0.0  java RangeAccrual$
110.39  Admin      2/25 10:41     0+00:00:17 R  0    0.0  java RangeAccrual$
110.40  Admin      2/25 10:41     0+00:00:25 R  0    0.0  java RangeAccrual$
110.41  Admin      2/25 10:41     0+00:00:18 R  0    0.0  java RangeAccrual$
110.42  Admin      2/25 10:41     0+00:00:02 R  0    0.0  java RangeAccrual$
110.43  Admin      2/25 10:41     0+00:00:00 I  0    0.0  java RangeAccrual$
110.44  Admin      2/25 10:41     0+00:00:00 I  0    0.0  java RangeAccrual$
110.45  Admin      2/25 10:41     0+00:00:00 I  0    0.0  java RangeAccrual$
110.46  Admin      2/25 10:41     0+00:00:00 I  0    0.0  java RangeAccrual$
110.47  Admin      2/25 10:41     0+00:00:00 I  0    0.0  java RangeAccrual$
110.48  Admin      2/25 10:41     0+00:00:00 I  0    0.0  java RangeAccrual$
110.49  Admin      2/25 10:41     0+00:00:00 I  0    0.0  java RangeAccrual$

14 jobs; 7 idle, 7 running, 0 held
```

Screenshot 16: Status of job queue displayed using Condor command “condor_q”

Finally, it is possible to mark submitted jobs for removal from the job queue. This is done using the command *condor_rm*. The job number that represents the job to be deleted has to be provided as an argument to this command. The job number can be determined from the output of the command *condor_q* (field ID). The output of *condor_rm* command is shown in screenshot 17 below.

```
E:\>condor_q
-- Submitter: 210-A : <192.168.0.210:1040> : 210-A
ID      OWNER      SUBMITTED      RUN_TIME ST  PRI  SIZE  CMD
110.38  Admin      2/25 10:41     0+00:01:34 R  0    0.0  java RangeAccrual$
110.39  Admin      2/25 10:41     0+00:01:22 R  0    0.0  java RangeAccrual$
110.42  Admin      2/25 10:41     0+00:01:05 R  0    0.0  java RangeAccrual$
110.43  Admin      2/25 10:41     0+00:00:59 R  0    0.0  java RangeAccrual$
110.44  Admin      2/25 10:41     0+00:00:48 R  0    0.0  java RangeAccrual$
110.45  Admin      2/25 10:41     0+00:00:29 I  0    0.0  java RangeAccrual$
110.46  Admin      2/25 10:41     0+00:00:00 I  0    0.0  java RangeAccrual$
110.47  Admin      2/25 10:41     0+00:00:00 I  0    0.0  java RangeAccrual$
110.48  Admin      2/25 10:41     0+00:00:00 I  0    0.0  java RangeAccrual$
110.49  Admin      2/25 10:41     0+00:00:00 I  0    0.0  java RangeAccrual$

10 jobs; 5 idle, 5 running, 0 held

E:\>condor_rm 110.46 110.47 110.48 110.49
Job 110.46 marked for removal
Job 110.47 marked for removal
Job 110.48 marked for removal
Job 110.49 marked for removal

E:\>condor_q
-- Submitter: 210-A : <192.168.0.210:1040> : 210-A
ID      OWNER      SUBMITTED      RUN_TIME ST  PRI  SIZE  CMD
110.39  Admin      2/25 10:41     0+00:01:39 R  0    0.0  java RangeAccrual$
110.42  Admin      2/25 10:41     0+00:01:35 R  0    0.0  java RangeAccrual$
110.43  Admin      2/25 10:41     0+00:01:33 R  0    0.0  java RangeAccrual$
110.45  Admin      2/25 10:41     0+00:00:40 I  0    0.0  java RangeAccrual$

4 jobs; 1 idle, 3 running, 0 held
```

Screenshot 17: Jobs removed from the queue using Condor command “condor_rm”

The Condor pool that was used to evaluate MMD task farming comprised of the following:

- Four laptop computers running the Microsoft Windows XP Professional operating system, each with a 1.73GHz Intel Celeron processor and 1GB RAM. Each laptop was pre-installed with Condor middleware and Microsoft Excel. One of these laptops had the Condor matchmaking agent running. More details on matchmaking can be found under section 2.10.1.

- Three low-end desktop PCs running either Microsoft Windows XP Professional or Microsoft Windows 2000, with either Pentium I or Pentium II processors and 128MB or 256MB RAM. Like the laptops, these were pre-installed with Condor middleware and MCS CSP Excel.
- The laptops and desktop PCs were connected through a 100Mbps switch to form an isolated Condor test pool (figure 41).

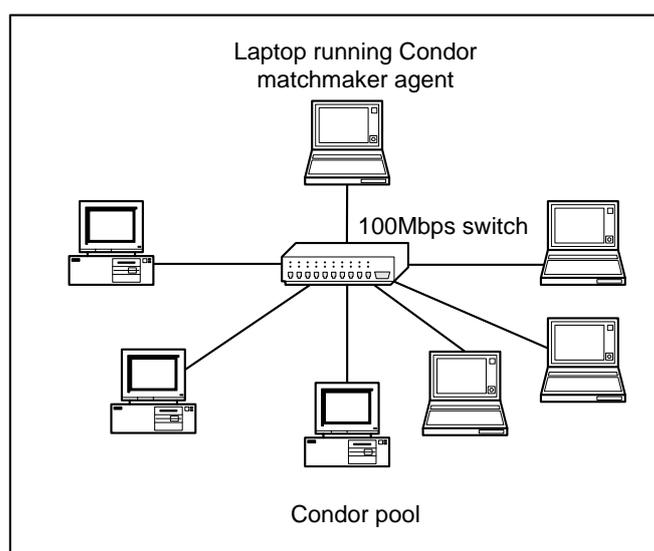


Figure 41: Condor test pool

5.5.5 Evaluation of MMMD task farming service

The evaluation criterion for this case study was whether practical implementation of the task farming service was possible, and in turn whether the CSP-specific task farming service was realizable using EDGC middleware Condor. The discussion presented in the previous section has shown that the middleware can support MCS CSPs to execute different models, each with multiple running instances, over the Condor pool. It can be argued that the Condor mechanism which allows submission and execution of multiple instances of two different MCS CSP-based models would also allow execution of multiple DES CSP-based models over the Condor pool. It can therefore be concluded that Condor can support CSP-based MMMD task farming service. This, understandably, also suggests that Condor can support SMMD task farming.

5.6 Ford case study for evaluation of SMMD task farming service

The Ford Motor Company case study evaluates WinGrid with respect to task farming service. It is a real-world case study. As shown in table 34, there are two evaluation criteria that will have to be satisfied. One, the task farming service can be implemented in practise, and two, the execution of batch simulation experiments is faster over dedicated grid nodes compared to their execution over a dedicated standalone computer.

Table 34: Ford case study

CSP-specific service	Grid Middleware	MCS / DES CSP used	Case study evaluation criteria
SMMD task farming service	WinGrid	Witness (DES CSP)	(1) Solution is implementable and the service is realizable (2) Execution is faster over dedicated grid resources compared to a standalone execution

This section is structured as follows. Section 5.6.1 presents an overview of the case study. Section 5.6.2 then describes the simulation application (FIRST) that is being used in Ford to create Witness models and to experiment with them. This is followed by a discussion on the technology used to grid-enable FIRST (section 5.6.3). The experiments that are conducted using FIRST and their results are presented in section 5.6.4 and section 5.6.5 respectively. This section concludes with a discussion on the viability of using WinGrid for supporting SMMD task farming (section 5.6.6).

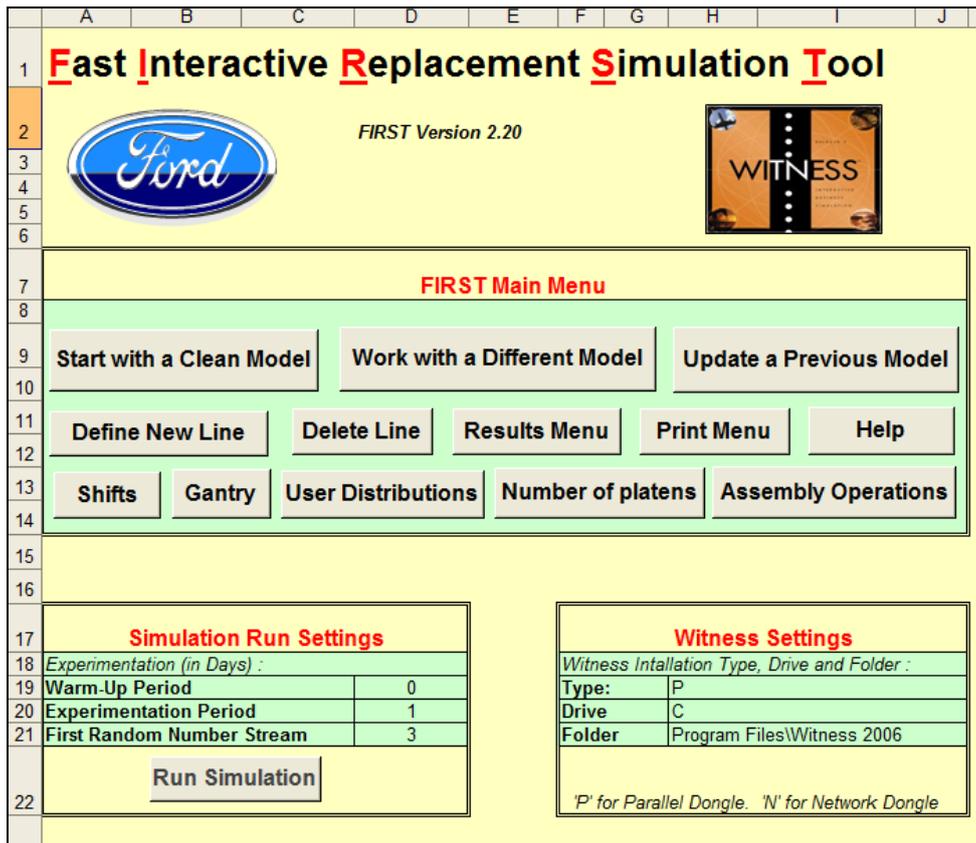
5.6.1 Overview

The Ford Motor Company makes use of computer simulation to design new engine manufacturing facilities and for process improvement in routine day-to-day operations. The production of an engine is a complex operation at Ford as it involves the manufacture and assembly of a wide variety of components into several possible engine types based on orders from the customer (Taylor et al., 2005a). Using simulation in this process helps to experiment with different machine configurations, buffer capacities, changeover schemes (switching production from one engine type to another), shift patterns, machine downtime, etc., and contributes to ensuring a smooth work-flow in the engine production line.

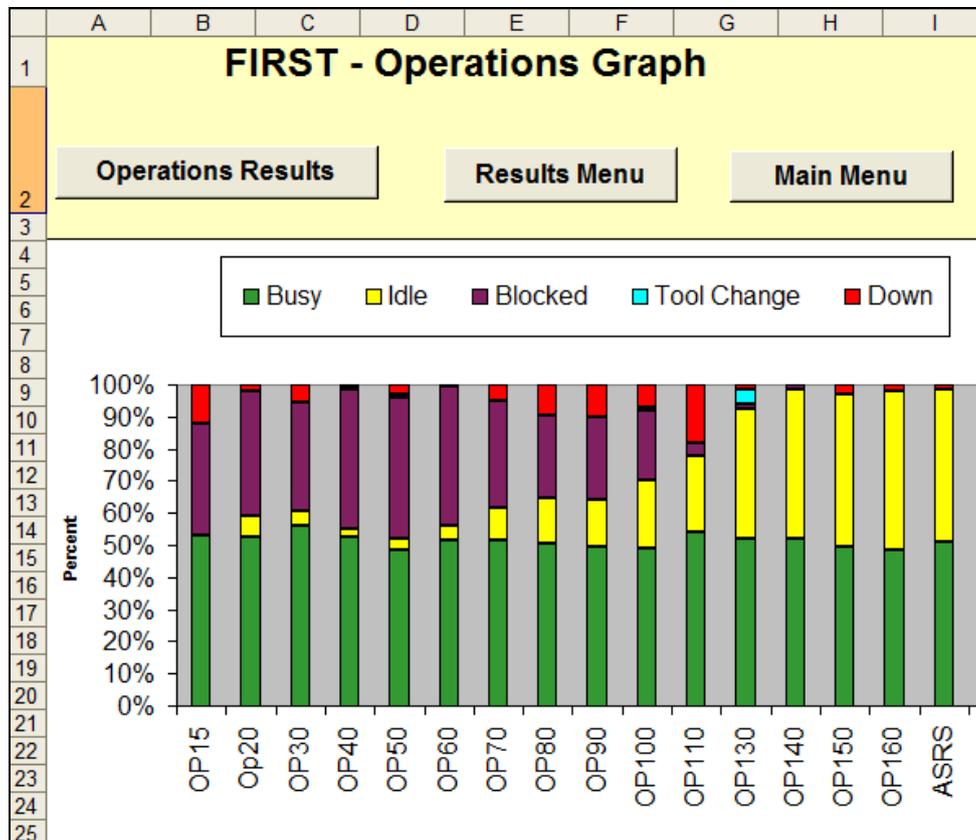
Ford uses the DES CSP Witness at the Dunton Engineering Centre in Essex. Wider adoption of simulation has been hindered due to the lack of expertise required for using Witness. Like any other CSP such knowledge is normally acquired over a period of time. In order to encourage faster adoption of simulation, Ford felt the requirement for an application which would make it easier and quicker for people to use simulation (Ladbrook and Janusszczak, 2001). As a response to this the FIRST application was developed by Ford with assistance from the Lanner Group (the developers of Witness).

5.6.2 Fast Interactive Replacement Simulation Tool (FIRST) application

Fast Interactive Replacement Simulation Tool (FIRST) is a Ford proprietary tool that builds a Witness model of an engine manufacturing line based on data input through Microsoft Excel. The Excel-based application consists of more than 30 worksheets, 10 VBA modules and many Excel macros. It uses Visual Basic for Application (VBA) to interface between Excel and the Witness CSP, and dramatically cuts down the time it takes to build and run a Witness simulation model by automating much of the process of model building.



Screenshot 18: FIRST application main menu



Screenshot 19: Graph generated by FIRST using data returned by Witness

To build a manufacturing line in Witness through FIRST, the application has to be provided with inputs such as the number of machines, corresponding buffer sizes, time and frequency of tool change, changeovers, shift patterns, user defined distributions, warm-up period, experimentation period, etc. Once all the data has been entered and the “Run Simulation” button clicked (see screenshot 18), the model is built in Witness and the simulation starts. Results of the simulation are returned back to FIRST and are displayed using various Excel-based features like tables, graphs (see screenshot 19), conditional formatting, etc. FIRST is under continuous development and new features are added to match the requirements of the modellers at Ford.

The complexity of an engine manufacturing line at Ford usually means that a number of experimental scenarios may have to be run before an ideal solution can be identified (this fits the requirements of SMMD task farming because only one Witness model is being simulated but with multiple parameters). Each run requires setting experiment values using FIRST and then executing the model to determine the outcome. This commences with the process of parsing the various Excel worksheets (defined within the application) and executing appropriate Witness commands with arguments based on the extracted values. This, in turn, progressively builds the Witness model, and when the model is complete, Witness starts simulating it. The time taken to generate the model using FIRST is dependent upon the amount of data to be parsed. For example, in the case of large models comprising multiple manufacturing lines it may take as long as 10-15 minutes to modify the model (re-parameterise for experimentation) and up to 60 minutes to run it. If 10 different scenarios were to be run using FIRST then the execution time is approximately 11 to 12 hours to finish all the experiments using one computer. Keeping in mind the fact that Ford has multiple Witness licences, it would be reasonable to assume that the time taken to build and conduct multiple simulation experiments can be significantly reduced by utilizing all the available computing resources. One way to achieve this is through grid computing and executing the FIRST application over the grid.

In this case study WinGrid is used to investigate whether SMMD task farming service could speed up simulation experimentation using FIRST. Most of the PCs in Ford’s simulation division can be used uninterrupted during the execution of a simulation. The WinGrid nodes can therefore be considered as dedicated resources. The next section describes how WinGrid is integrated with FIRST to enable SMMD task farming.

5.6.3 Grid-enabling FIRST using WinGrid

Integration of FIRST with WinGrid is achieved using the WinGrid-CSP integration technology that has been presented in section 4.4. Since FIRST is an Excel-based application, it can be accessed through Excel’s COM interface. A custom built *FIRST adapter* has been developed which encapsulates the COM function calls required by WTC to interact with the FIRST

application. In the WinGrid architecture, FIRST is the Worker Application (WA). Further discussions on WinGrid architecture can be found in section 4.2.

For the purpose of experimenting with multiple simulation scenarios, an Excel spreadsheet based controller called *FIRST experimentation tool* has been developed. It lists all of the experiment parameters. In the WinGrid architecture, the First experimentation tool is the Manager Application (MA) and it interacts with the WinGrid Job Dispatcher (WJD) to send different parameters for experimentation to different FIRST applications through their corresponding WinGrid Thin Clients (WTCs). Once the FIRST application has completed simulating a model, it sends back to the MA the result it received from Witness. This communication is done through the corresponding WTCs and the WJD. For each result received by the FIRST application tool a new worksheet is created and the values stored. The worksheets are named according to the experiment numbers. The interaction between the MA and WJD is by means of an *Excel Adapter*. This adapter contains specific COM calls required by the WJD to access the MA. A screenshot of the FIRST experimentation tool is shown in screenshot 20 below. The example shows experimentation with various buffer sizes of the machines.

	A	B	C	D	E	F	G	H
1	FIRST Experimentation Tool							
2								
3	Buffer Capacity							
4	Buffer Na	Exp 1	Exp 2	Exp 3	Exp 4	Exp 5	Exp 6	Exp 7
5	PreOP15	80	20	30	40	50	60	70
6	PreOp20	18	13	15	17	19	21	23
7	PreOP30	11	10	9	8	7	6	5
8	PreOP40	57	55	50	45	40	35	30
9	PreOP50	12	13	14	15	16	17	18
10	PreOP60	13	13	13	13	13	13	13
11	PreOP70	13	15	15	13	13	16	16
12	PreOP80	18	19	23	27	31	35	39
13	PreOP90	62	60	58	56	54	52	50
14	PreOP100	19	17	15	15	17	19	15
15	PreOP110	21	21	21	21	21	21	21
16	PreOP130	16	20	22	25	28	31	34
17	PreOP140	1	2	3	1	2	3	1
18	PreOP150	16	17	18	15	16	14	12
19	PreOP160	19	20	22	24	26	12	14
20	PreASRS	23	20	21	11	20	15	14
21	File / WS	Q:\Results						

Screenshot 20: FIRST experimentation tool showing a list of experiments

As has been noted earlier, WinGrid is written in Java which is a non-COM compliant language. Java Native Interface technology has therefore been used for communication between the Excel Adapter, WinGrid and the First Adapter. Figure 42 shows the integration architecture of WinGrid and FIRST.

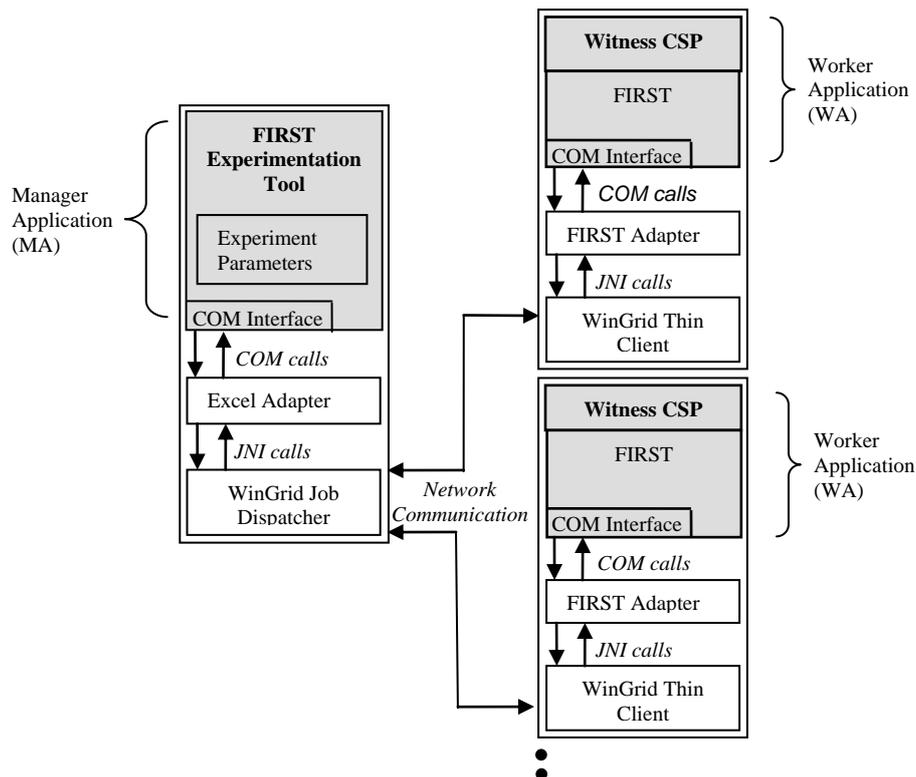


Figure 42: Integration architecture of WinGrid and First

5.6.4 Experiments

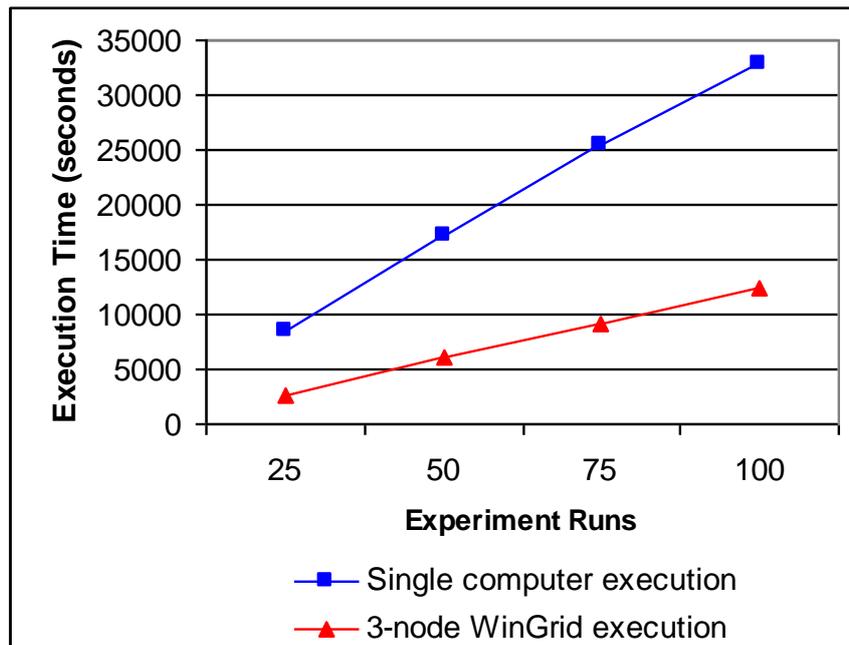
The FIRST application automatically builds a Witness model that consists of one main and one supplementary assembly line. These models are based on data that are preset in the FIRST application. The data provides, among other details, the number of machines in each assembly line and their corresponding buffer sizes. Multiple experiments with FIRST over WinGrid have been conducted by varying the size of the buffer, such that each experiment has a different set of buffer parameters. The FIRST experimentation tool (see screenshot 20) defines the buffer capacities of each machine in the main assembly line, for all the experiments that are to be conducted. The performance is measured in terms of the time taken to execute 25, 50, 75 and 100 runs of the experiment respectively. So as to demonstrate the potential of achieving speedup when using FIRST over dedicated WinGrid nodes, the same experiments are repeated using a standalone version of FIRST. An Excel spreadsheet similar to FIRST experimentation tool is used to automate the running of the standalone version. The results obtained by the WinGrid version and the standalone version of FIRST are shown in section 5.6.5.

In order to evaluate the performance of FIRST over WinGrid, a dedicated 4-node experimental test bed was set up consisting of PCs with PIII 648 MHz processors and 256MB RAM, connected through an isolated 100Mbps switch. Three of these nodes were configured as WinGrid workers and were installed with WTC, Witness and the FIRST application (Excel).

The fourth PC served as the WinGrid master and had the WJD and FIRST experimentation tool (Excel) installed on it.

5.6.5 Results

The results obtained from the experiments are shown below (graph 1). The performance results show that the WinGrid version of FIRST completed the execution of all the experiments approximately three times faster when compared to the standalone execution. This is to be expected since three dedicated WTCs were processing jobs sent by the master computer.



Graph 1: Time taken to execute FIRST application using different workloads

5.6.6 Evaluation of SMMD task farming service

The two evaluation criteria for this case study were, (1) the WinGrid solution is implementable and the CSP-specific task farming service is realizable, and (2) dedicated WinGrid nodes could achieve faster execution of FIRST simulations compared to dedicated, one computer, execution. Through a discussion on the WinGrid-FIRST integration technology (section 5.6.3) it has been shown that criterion one has been met. The results of the experiments in section 5.6.5 have demonstrated that criterion two has also been met. It can therefore be concluded that WinGrid can facilitate SMMD task farming and can help simulation users to execute simulation experiments faster.

Although the case study had shown the viability of the grid-enabled FIRST application within Ford, it was not considered for production-level deployment within their simulation group. The reasons for this are discussed in the next page.

After a successful demonstration over an experimental WinGrid test bed, the logical next step was to deploy WinGrid and the grid-enabled FIRST application on the computers at Dunton. This would demonstrate the application to the engineers of the group and the feedback gathered could be used for further development of the grid version of FIRST. Doing this required the approval of the IT support staff at Ford as they were responsible for maintaining the existing hardware, installing software and securing computer systems within the organization. Two issues were identified through discussions with the IT support staff with regard to WinGrid deployment at Ford. First, Ford did not allow any kind of server software to be installed on the office computers. Another requirement of Ford was the use of web services for communication between PCs. As has been stated earlier, web services enable application interaction using standard Internet protocols and open standards. In other words, a web service is accessible on the same terms as any other resource on the Internet. Since desktops at Ford have Internet access (with current security policy), web services deployed on any web server should also be accessible.

Both these constraints ruled out the deployment of the current WinGrid implementation. WinGrid did not fulfil the first requirement because WTCs have inbuilt server functionality and are meant to be installed on individual office PCs. The use of Java sockets for communication between the WinGrid nodes meant that it also failed to satisfy the second requirement. Multiple sockets are required because WinGrid implements the “push” job scheduling mechanism (section 4.2). The “push” mechanism has been found to be appropriate in the context of using grid middleware in an enterprise setting (section 3.6.3). But clearly, the security restrictions in place at Ford would not allow the production deployment of WinGrid.

It was realized that for the deployment of WinGrid to be possible at Ford, the existing “push” based architecture had to be substantially changed and requirements imposed by Ford incorporated into the system. The modified architecture was based on web services and was called web services extension to WinGrid, or WinGrid-WS for short. It was implemented by Anders Alstad as part of his Masters dissertation (Alstad, 2006). WinGrid-WS implements the “pull” job scheduling mechanism and uses web services for communication. Thus, it uses port 80 for all its communication (like BOINC). This shows that although the “pull” middleware architecture is not considered very efficient for CSP-based simulation in industry (see section 3.6.3), WinGrid-WS was still preferred over WinGrid. This indicates that grid middleware solution should be flexible and should be able to adapt to changing industry requirements.

Finally, it has to be added that the original multi-server implementation of WinGrid is again being considered for production deployment at Ford (May 2007). This was made possible through further discussion with the IT staff at Ford.

5.7 IB case study for evaluation of workflow and SMMD task farming services

The real-world Investment Bank (IB) case study investigates how workflows can be implemented using WinGrid. It is also an example of SMMD task farming using MCS CSP Analytics. The focus of this case study is, however, on workflows because MCS-based SMMD task farming has been previously discussed in section 5.4. The services being evaluated, the technologies being used and the case study evaluation criteria are listed in table 35 below. The primary evaluation criterion for both these services is that the solution can be implemented, and thus the CSP-specific services can be realized. The additional criterion for task farming service is that non-dedicated WinGrid nodes will be able to execute a set of simulation experiments faster.

Table 35: Investment bank case study

CSP-specific service	Grid Middleware	MCS / DES CSP used	Case study evaluation criteria
Workflow service	WinGrid	Analytics and Excel (MCS CSP)	(1) Solution is implementable and the service is realizable
SMMD task farming service	WinGrid	Analytics (MCS CSP)	(1) Solution is implementable and the service is realizable (2) Execution is faster over non-dedicated grid resources compared to a standalone execution

The next section (5.7.1) provides a brief overview of credit risk simulation and the MCS CSP Analytics. The Analytics-based IRS-RBF application currently being used by the bank for simulating five different financial products is discussed in section 5.7.2. The technology used for grid-enabling the IRS-RBF application to support workflow and task farming services is presented in section 5.7.3, followed by a discussion on the experiments that were conducted (section 5.7.4) and their results (section 5.7.5). This section concludes with an evaluation of the suitability of WinGrid to support the workflow service and SMMD task farming service (section 5.7.6).

5.7.1 Overview

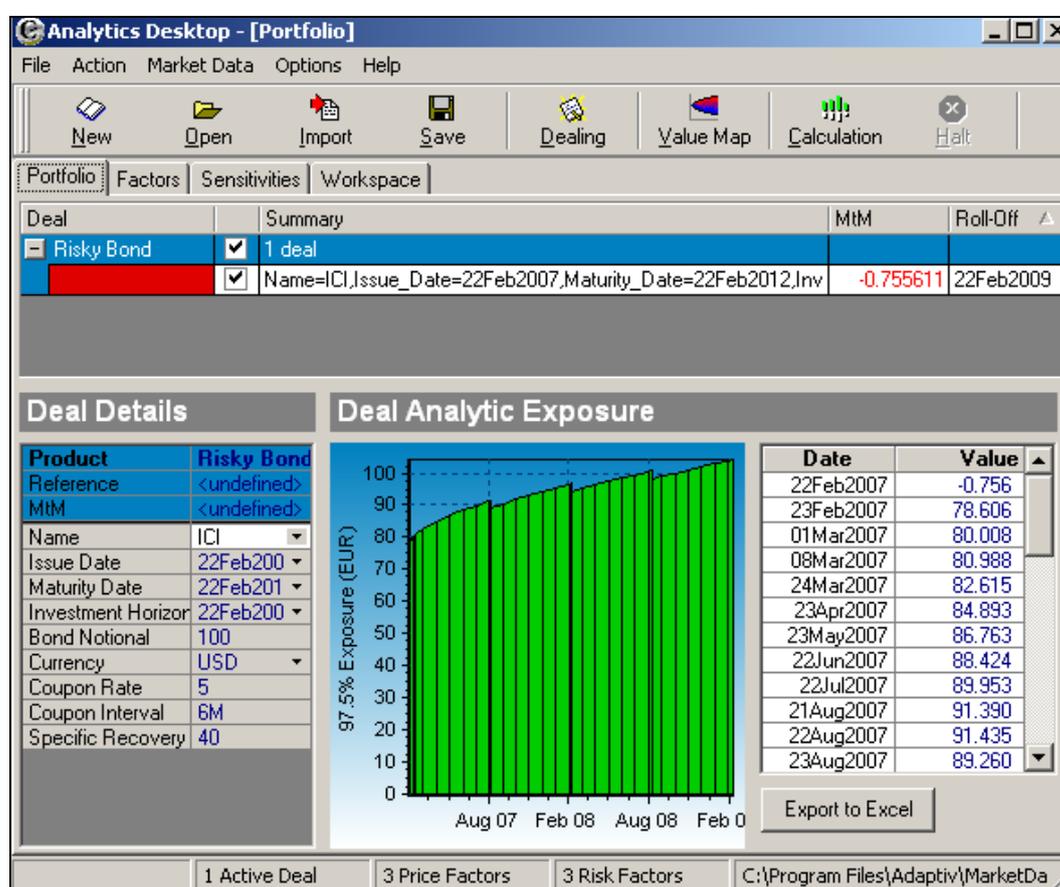
The investment bank uses MCS CSP Analytics for Monte Carlo-based credit risk simulations of counterparty transactions. The transactions between the investment bank and the counterparties may involve agreements to exchange different sequences of payments over a period of time. Credit risk is the potential that the counterparty will fail to meet its obligations in accordance with the agreed terms (Basel Committee on Banking Supervision, 1999).

In assessing credit risk from a single counterparty, an institution (in our case the investment bank) must consider three issues (Contingency Analysis, 2003):

- (1) *Default probability*: What is the likelihood that the counterparty will default on its obligation either over the life of the obligation or over some specified time period?
- (2) *Credit exposure*: In the event of a default, how large will the outstanding obligation be when the default occurs?

(3) *Recovery rate*: In the event of a default, what fraction of the exposure may be recovered through bankruptcy proceedings or some other form of settlement?

Credit risk simulations are usually used to calculate the credit exposure over a period of time. Analytics is the calculation engine for the Credent credit risk system that provides algorithms to calculate time-dependent profiles of credit exposure using MCSs (Credent Analytics, 2007). Analytics consists of three separate applications, namely, Analytics Desktop, Market Data Manager (MDM) and Analytics Server COM Object. The Analytics Desktop application (screenshot 21) is a standalone application that uses a calculation engine to construct and analyse financial portfolios. It links to the Market Data Manager to derive both current and historical market data which serve as inputs to these calculations. Analytics Server COM Object is essentially a COM interface to the Analytics Desktop and can be invoked by external systems.



Screenshot 21: MCS CSP Analytics Desktop application

Analytics Desktop application is installed on multiple workstations within the credit risk division of the investment bank. It is currently used to support five different financial products, namely, *currency swaps*, *default swaps*, *forward rate agreements*, *interest rate swaps (IRS)* and *risky bond forwards (RBF)*. For each of these products, a contractual agreement is reached between the investment bank and the counterparties to exchange payments over a period of time. These products involve a risk element and differ based on the mechanisms

that determine what is exchanged (for example, principal amount, foreign exchange, etc.), how the exchange payments are calculated (for example, interest rate payments calculated over a notional principle, fixed payments, etc.). The time taken by Analytics to create risk profiles varies considerably based on the product under consideration. Using multiple Analytics-installed workstations made available through desktop grid middleware, it might be possible to reduce the execution time of the MCSs through task farming.

In this case study WinGrid has been used with Microsoft Excel spreadsheets and Analytics for computation of complex risk calculations. The MCS CSP here is Analytics, and Excel is used to construct different parameters (using Excel VBA) for Analytics to simulate. The existing IRS-RBF application is described next.

5.7.2 IRS-RBF application

The investment bank uses the IRS-RBF application to simulate five different financial products. This application comprises of different Excel spreadsheets, VBA modules and MCS CSP Analytics. Analytics is invoked by the VBA modules (present in the Excel spreadsheets) through the Analytics Server COM Object. The IRS-RBF application takes its name from two different products, namely, Interest Rate Swaps (IRS) and Risky Bond Forwards (RBF), which it simulates. The name has been given by the author to represent the collective components that logically make up this application.

Simulations of the financial products are a two-stage process. In the first stage, risk profiles are generated by invoking Analytics through Excel. The parameters passed-on include different currency codes like GBP, INR and USD. Analytics outputs the results of the simulation in the form of text files. The first stage is subsequently referred to as the **generate profiles stage**.

In the second stage, referred to as the **create table stage**, PFE and EPE tables are generated by Excel. These tables are based on the values present in the text files that are created in the generate profiles stage. PFE or *Potential Future Expose* is the maximum amount of counterparty exposure (i.e., the maximum outstanding obligation if counterparties were to default) that is expected to occur on a future date with a high degree of statistical confidence; EPE or *Expected Positive Exposure* is the average counterparty exposure in a certain interval, e.g., a month or a year (Canabarro and Duffie, 2003).

Stage one and stage two processing of the IRS-RBF application involves three distinct operations that have to be “manually-executed”. These operations are (1) generate profiles, (2) create EPE tables, and (3) create PFE tables. The EPE/PFE create table operations can only start after successful execution of the generate profile operation. The time taken to execute both these phases for each of the five products that is simulated by the IRS-RBF application is shown in table 36. The total number of currencies used for simulating these

products is also indicated. The data for this table has been provided by the credit risk analysts who have developed the IRS-RBF application.

Table 36: Execution time for different products using the original IRS-RBF application

Products	Generate Profiles	Create Tables	Currencies
Currency Swaps	15 minutes	10 minutes	37
Default Swaps	15 minutes	5 minutes	1
Forward Rate Agreements	35 minutes	10 minutes	11
Interest Rate Swaps (IRS)	1 hour 15 minutes	12 hours	23
Risky Bond Forwards (RBF)	4 hours 30 minutes	1 hour 20 minutes	13

From the table it is clear that the IRS and RBF products take the maximum time to execute. The numbers of currencies that are simulated by these products are 23 and 13 respectively. Ideally, the bank would expect to run the IRS and RBF simulations with 37 currencies. This means that the execution time will be further increased. It has been demonstrated earlier in the Ford case study (section 5.6) that WinGrid's SMMD task farming service could be used to reduce execution time of simulation experiments over dedicated nodes. The same service could arguably be used for the investment bank case study to speed up the IRS-RBF application. However, unlike Ford, where the simulation department had access to dedicated resources over which to run their simulations, the computers being used by the credit risk division of the investment bank are non-dedicated resources. These resources are the desktop PCs that are used by the credit risk analysts at their work place. Thus, WinGrid's SMMD task farming service would have to be executed over these non-dedicated PCs.

The IRS-RBF application also provides us with an opportunity to assess whether CSP-based workflow service could be potentially supported through WinGrid. This opportunity arises because the IRS-RBF simulation involves the manual invocation of three distinct operations (generate profiles, create EPE tables, create PFE tables), and there is data dependency between these operations. A workflow could potentially combine the manual operations into one all-encompassing automated operation. In this case study WinGrid is examined in relation to its potential for executing such a workflow.

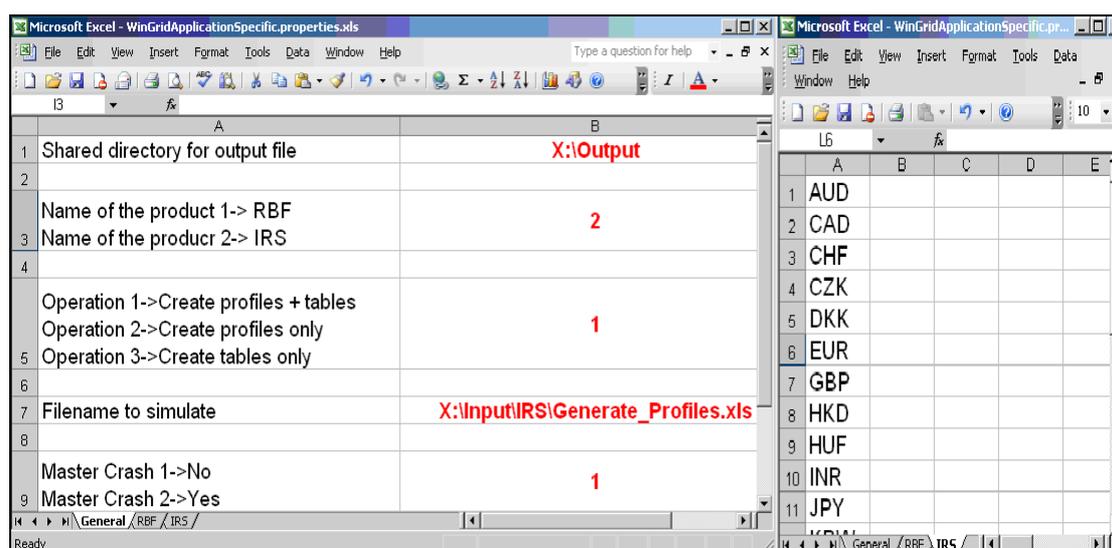
The grid-enabled version of the IRS-RBF application only simulates the IRS and RBF products because they can gain most from execution over the grid. The next section discusses the implementation of SMMD task farming service and workflow service using IRS-RBF application and WinGrid.

5.7.3 Grid-enabling IRS-RBF application

For the IRS-RBF application to utilize the resources made available through WinGrid, it has to be integrated to the WTC and the WJD. Integration of the Excel-based IRS-RBF application with WTC is achieved using Excel's COM interface. A custom built *IRS-RBF adapter* has been developed which encapsulates the COM function calls required by WTC to interact with

the IRS-RBF application. In the WinGrid architecture, the IRS-RBF application is the Worker Application (WA). Further discussion on WinGrid architecture can be found in section 4.2.

In this case study the WinGrid Master Application (MA) that controls the IRS and RBF simulation execution is called the *WJD Application Specific Parameter (ASP) Tool for IRS-RBF application* (screenshot 22). It is an Excel-based tool that consists of specific parameters that are required for processing the IRS-RBF application; for example, the name of the output directory, the name of the product to simulate (IRS or RBF), the operation to perform (create table, create profiles or both), the filename to simulate, whether the WJD process had crashed during an earlier run, etc. All this information is present in the worksheet called “General”. The WJD APS tool also consists of two other worksheets, namely “RBF” and “IRS”. These worksheets contain data specific to the RBF and the IRS simulations respectively. Each worksheet has a list of currencies. Each currency is a separate unit of computation (job). The interaction between the MA and WJD is by means of an *Excel Adapter*. This adapter contains specific COM calls required by WJD to access the MA. Figure 43 shows the integration architecture of WinGrid and IRS-RBF application.



Screenshot 22: WJD Application Specific Parameter (APS) tool for IRS-RBF application

As has been pointed out earlier, although this case study primarily focuses on the implementation of the workflow service using WinGrid, it is also an example of SMMD task farming service with MCS CSP Analytics. SMMD task farming has been previously discussed in the context of the BOINC case study and therefore only a brief overview of SMMD task farming with WinGrid will be presented in the next section (section 5.7.3.1). This will be followed by a discussion of the workflow service implementation for the IRS-RBF application using WinGrid (section 5.7.3.2).

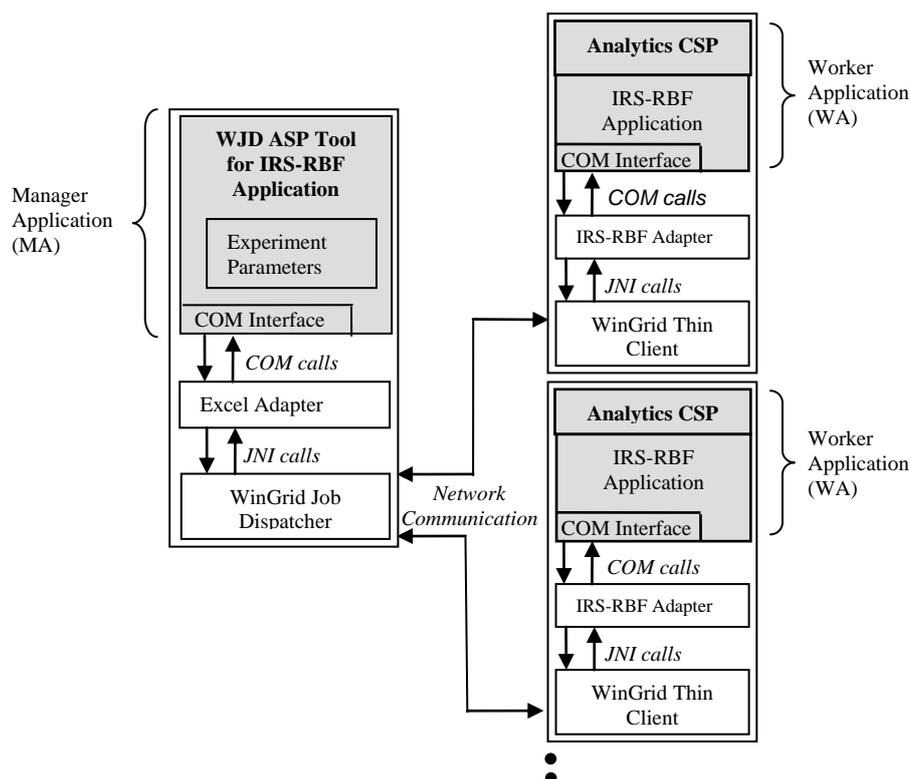


Figure 43: Integration architecture of WinGrid and IRS-RBF application

5.7.3.1 SMMD task farming with MCS CSP Excel and WinGrid

The WJD is the WinGrid master process that interacts with the MA (WJD APS tool for IRS-RBF application) to extract the currency list. This interaction is performed through the Excel adapter. The adapter defines procedures to extract currency values present in the RBF and IRS worksheets of the MA and to transfer them to the WJD. The WJD then allocates the job to each connected WTC. Each job consists of one currency name and one control message (control messages are discussed in the next section). The WTCs process the jobs and return the results. The results are only job completion messages because the outputs of the simulations that are executed by the WTCs are stored in the shared output folder that the WJD can also access. After one currency has been successfully computed the WJD will send the next currency in its queue. This process repeats until all the currencies have been successfully processed. This is an example of SMMD task farming because each WTC will process the same IRS-RBF application (single model) using different parameters (multiple data) that are passed-on to Analytics as inputs to the MCS.

5.7.3.2 Workflows with WinGrid

The grid-enabled version of the IRS-RBF application consists of a total of 9 Excel files (with their associated VBA code) and the Analytics defined procedures for MCS. This application requires 5 Excel files for simulating either the IRS or the RBF financial product. Out of these 5 files only one is for common use. Thus, 4 files (for IRS) + 4 files (for RBF) + 1 file (common for IRS and RBF) = 9 Excel files in the IRS-RBF application. The files are stored in shared

network directories to enable both WAs and the MA to access them. The IRS-RBF application processing consists of the following five phases irrespective of whether it is being used for the IRS or the RBF simulation.

Phase 1: Create Profiles

Phase 2: Create EPE tables from the output files generated in Phase 1.

Phase 3: Create PFE tables from the output files generated in Phase 1.

Phase 4: Collate data from the EPE tables generated in Phase 2.

Phase 5: Collate data from the PFE tables generated in Phase 3.

These five phases are processed either by the WAs or the MA. Of the 5 files required for both the IRS and RBF simulation, 3 are required by WAs and 2 by the MA. The files on which the different IRS and RBF phases are dependent are listed below.

WA requirement for IRS:

- Phase 1: *Generate_Profiles.xls*
- Phase 2: *Generate_Profiles_[IRS_Create_Table_EPE].xls*
- Phase 3: *Generate_Profiles_[IRS_Create_Table_PFE].xls*

MA requirement for IRS:

- Phase 4: *Generate_Profiles_[IRS_MASTER_EPE].xls*
- Phase 5: *Generate_Profiles_[IRS_MASTER_PFE].xls*

WA requirement for RBF:

- Phase 1: *Generate_Profiles.xls*
- Phase 2: *Generate_Profiles_[RBF_Create_Table_EPE].xls*
- Phase 3: *Generate_Profiles_[RBF_Create_Table_PFE].xls*

MA requirement for RBF:

- Phase 4: *Generate_Profiles_[RBF_MASTER_EPE].xls*
- Phase 5: *Generate_Profiles_[RBF_MASTER_PFE].xls*

For both IRS and RBF application processing, phases 1, 2 and 3 are processed by the WAs on WTCs through the SMMD task farming mechanism implemented by WinGrid. Thus, the WJD allocates jobs for all three phases to the WTCs. Each job consists of a currency name and an associated control message. The control message identifies the processing phase (phase 1, phase 2 or phase 3) and an Excel file. The WAs read the control message, open the Excel file and then execute the VBA procedures defined in the file.

In **Phase 1** of processing, the execution of the VBA code by WinGrid invokes the Analytics-defined procedures in a finite loop. Every iteration of the loop transfers a new set of parameters for Analytics to simulate. The output of the MCS is written into a file by Analytics. Thousands of such files are created during Phase 1 in case of both IRS and RBF simulations.

Phase 2 and **Phase 3** of the IRS-RBF application processing involves creation of EPE and PFE tables from the output files generated in phase 1. The WAs create these Excel –based tables by reading phase 1 output files from a shared network drive. Irrespective of the number of jobs processed by WAs in phases 2 and 3, only one Excel file per WA per phase is created.

Phase 4 and **Phase 5** of processing for both IRS and RBF simulations is carried out by the MA after the previous phases have been successfully executed by the WAs. The MA creates two master files – one for EPE and another for PFE – with the objective of presenting collective results to the user. The MA does this by transferring data from the temporary Excel files (created by each WA during phase 2 and phase 3 of the simulation) to the Master EPE and PFE files. Thus, if there are 8 WTCs then the MA will have to combine results generated by each of the 8 WAs. And it has to do it twice - once for the master EPE table generation and again for the master PFE table generation.

As can be seen from the discussion above, there exists dependencies between these five processing phases in the IRS-RBF application. For example, Phases 2 and 3 can start only when phase 1 has completed. Similarly, processing for phases 4 and 5 can begin only after phase 2 and phase 3 processing has completed. The workflow between phases is represented in the form of a diagram below (figure 44). The dotted box signifies that multiple WAs can process the first three phases of the IRS-RBF application. Also, jobs in phases 2 and 3 can be processed in parallel by different WAs. The MA running in WJD is represented by a square box. The MA is responsible for processing phases 4 and 5 sequentially, after the earlier three phases have been processed by the WAs.

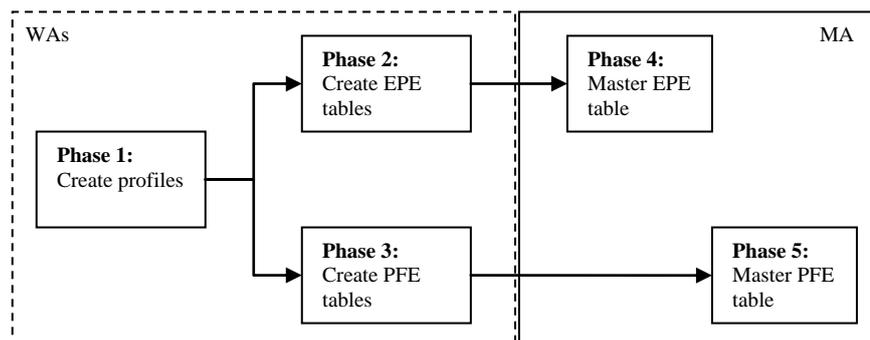


Figure 44: IRS-RBF application workflow

The workflow is currently implemented by the WJD using an algorithm which dispatches jobs to the WTCs based on the underline dependencies between phases 1, 2 and 3. The WJD algorithm then instructs the MA to sequentially process phases 4 and 5 of the IRS-RBF application. Execution of the IRS-RBF application workflow is shown in screenshot 23.

Table 37: Workunits to be processed by IRS and RBF simulations

	Workload one	Workload two
IRS	30 [30 workunits(IRS)]	69 [69 workunits(IRS)]
RBF	15 [15 workunits(RBF)]	39 [39 workunits(RBF)]

Identical IRS-RBF experiments for this case study were conducted on, (1) one dedicated WinGrid node (running both WJD and WTC), (2) 4 non-dedicated WinGrid nodes connected through the investment bank's corporate LAN, and (3) 8 non-dedicated WinGrid nodes connected with the corporate LAN. The grid-enabled IRS-RBF application was used for running experiments over the different test beds. The reasons for not using the original IRS-RBF application for execution over one dedicated, standalone PC were as follows.

- The original application was modified to a large extent by the author to enable faster execution of the grid-version of the application. The execution time of the original IRS-RBF application is presented in table 36.
- To run the IRS and RBF simulations using the original application meant that three different operations (create profiles, create EPE tables and create PFE tables) had to be manually invoked by the user. The execution of the grid-version of this application, on the other hand, was fully automated.

The experiments were conducted over a period of two days during normal working hours of the investment bank. The 4-node and the 8-node WinGrid experiments were run using production machines that were also being used by the analysts to do their jobs. The one node experiments were conducted using a PC that was not being used. The configurations of the machines used for the experiments are shown in table 38 below.

Table 38: Configuration of WinGrid nodes

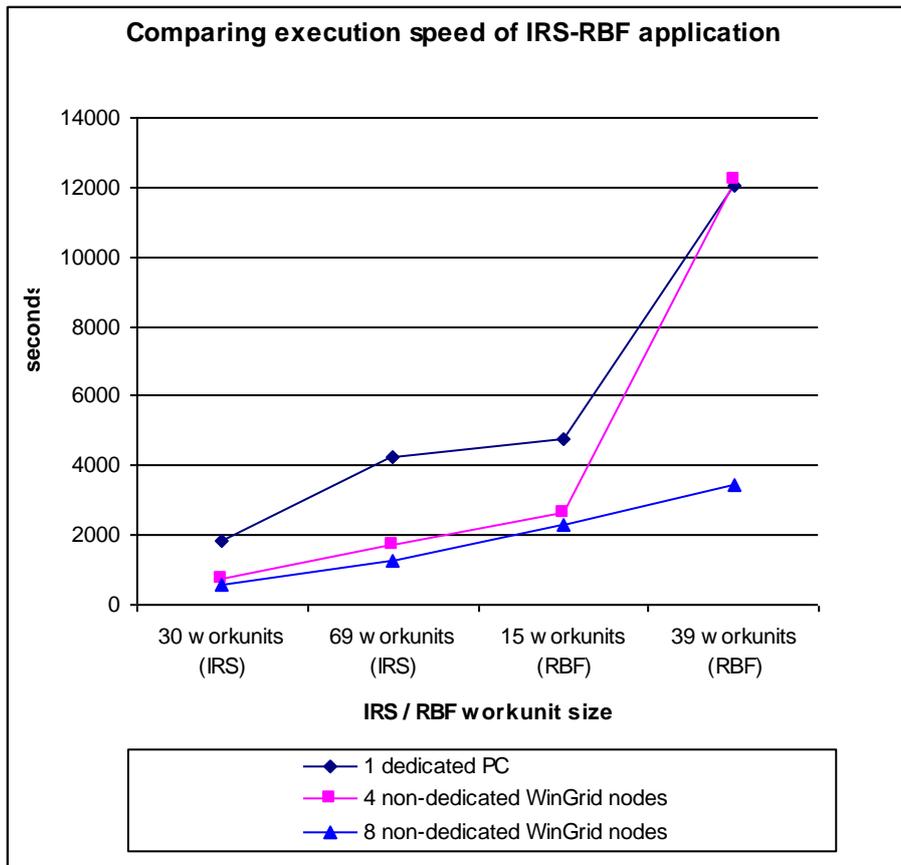
PC no.	CPU	RAM	Operating System
PC1	2.99GHz Intel Pentium IV Processor (hyper-threaded)	512MB	Microsoft XP Professional
PC2	2.99GHz Intel Pentium IV Processor (hyper-threaded)	512MB	Microsoft XP Professional
PC3	2.79GHz Intel Pentium IV Processor (hyper-threaded)	512MB	Microsoft XP Professional
PC4	2.13GHz Intel Pentium II Processor (hyper-threaded)	2GB	Microsoft XP Professional
PC5	2.13GHz Intel Pentium II Processor (hyper-threaded)	2GB	Microsoft XP Professional
PC6	2.13GHz Intel Pentium II Processor (hyper-threaded)	2GB	Microsoft XP Professional
PC7	2.13GHz Intel Pentium II Processor (hyper-threaded)	2GB	Microsoft XP Professional
PC8	2.13GHz Intel Pentium II Processor (hyper-threaded)	2GB	Microsoft XP Professional

As can be seen from the table, all the CPUs were hyper-threaded. Hyper-Threading Technology (HTT) is a new CPU technology that makes a single physical processor appear as two logical processors, wherein the physical execution resources are shared and the architecture state is duplicated for the two logical processors (Marr et al., 2002). The operating system treats a hyper-threaded CPU as two processors instead of one and a program can schedule processes or threads on both the logical processors and the CPU will execute them simultaneously (as if there were two physical processors present in the system).

The dedicated WinGrid node used for performing the standalone experiments had a 2.99GHz HTT Intel Pentium IV processor with 512MB RAM. The 4 non-dedicated WinGrid nodes used for the experiments comprised of different subsets of the machines at different times. The results of the experiments are presented next.

5.7.5 Results

The results of the IRS and RBF simulations are presented in graph 2. These results are based on two separate runs for each workload. The execution of all the four workloads, pertaining to either IRS or RBF simulation, was fastest using the 8 non-dedicated WinGrid nodes. The slowest execution was recorded by the standalone, dedicated WinGrid node.



Graph 2: Time taken to execute the IRS-RBF application using different workloads

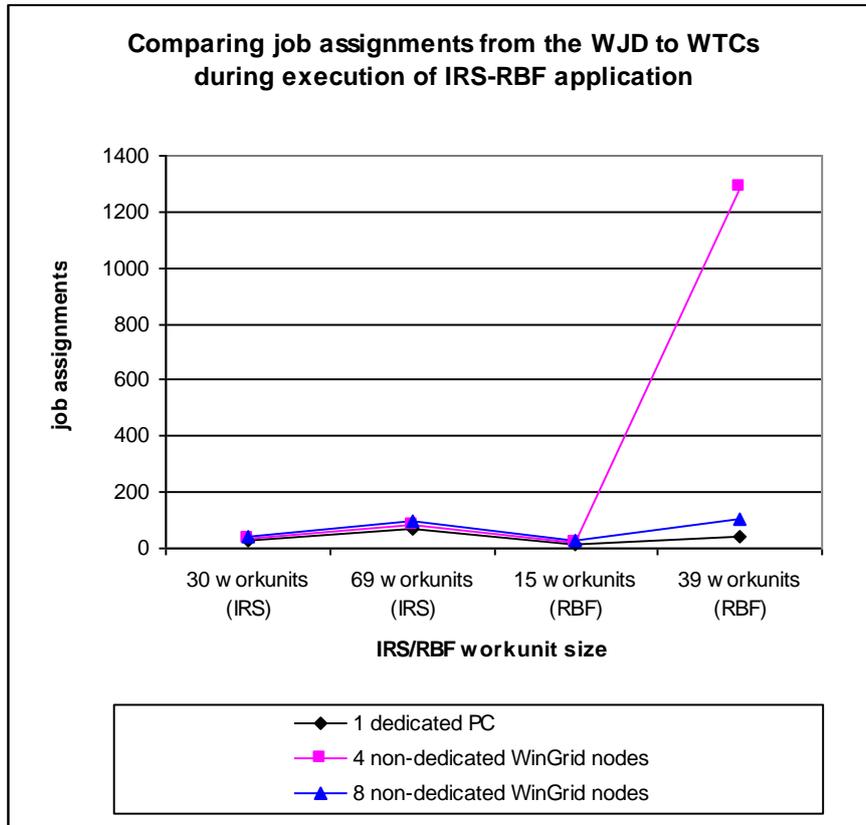
For workloads [30 workunits (IRS)], [69 workunits (IRS)] and [15 workunits (RBF)] the time taken to execute the IRS-RBF simulations using the 4 node WinGrid test bed was comparable to its 8 node counterpart. One reason for this may be that, with 8 nodes the number of Excel files created in Phase 2 (create EPE table) and Phase 3 (create PFE table) of the workflow are double the number of Excel files created when running the simulation using 4 nodes. Thus, the sequential MA operation in phases 4 and 5 (collate data from the EPE and PFE tables) would generally take more time in the case of the former. An additional reason could be the specific usage pattern of the PCs during the experiments. It is therefore possible that the majority of the PCs in the 8 node set-up had their WTC clients manually or automatically shut down because the analysts were using the computers for their own work. The WTC

program can be shut down manually through WinGrid's graphical user interface (see **Appendix D**). This can also happen automatically as the WTC program is designed to continuously monitor CPU and the memory usage on a PC, and if the resource usage crosses the pre-determined CPU/RAM threshold levels then the user jobs are immediately stopped. Similarly, jobs are started automatically again when the CPU and memory usage decreases as a result of a resource not being used. Thus, the time taken to execute the simulations on non-dedicated WinGrid nodes is very much related to the usage pattern of the underlying desktop PCs. Arguably, this is best shown by the results of workload [30 workunits (RBF)] in relation to its execution over 4 non-dedicated WinGrid nodes, where the time taken to complete the simulation is comparable to that of its standalone counterpart.

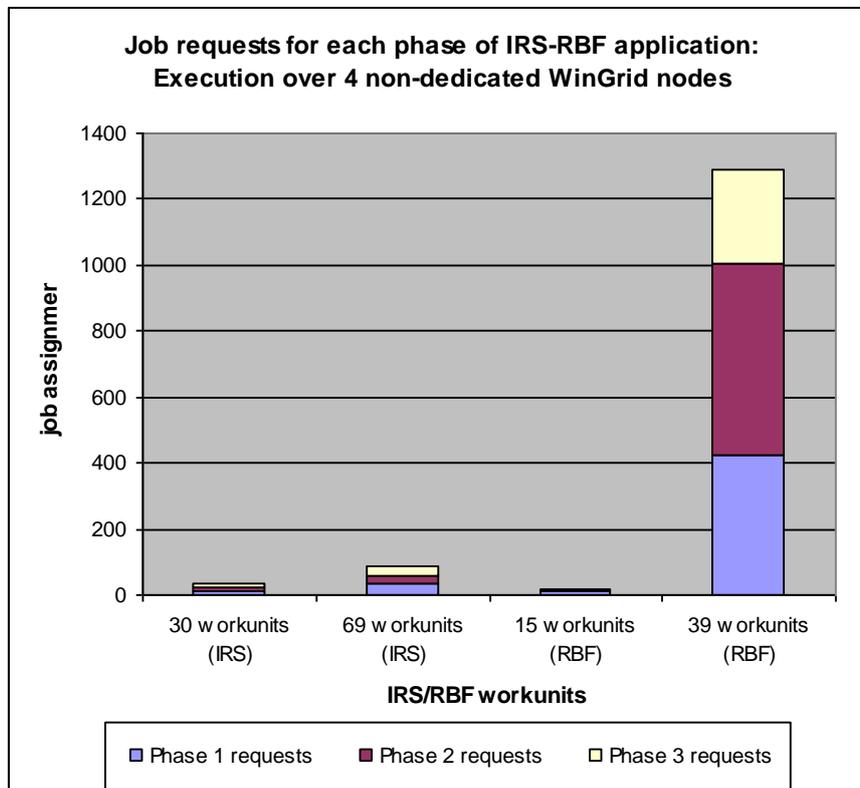
The IRS-RBF workflow consists of five different phases, of which three can be executed in parallel by the WTCs. The results presented in the subsequent discussions are only relevant to the first three phases of the workflow. Graph 3 shows the number of job assignments made by the WJD to the WTCs during the execution of both the IRS and RBF simulations. In general, the number of jobs received for processing by the WTCs (as a whole) is more than or equal to the number of jobs assigned by the WJD. This is because once a job is assigned to a WTC, it may either successfully process the job or may terminate processing before the simulation is complete. The latter happens if the WTC process is stopped either automatically or manually. In this case the unfinished job is again pushed back to the WJD queue for assignment to other WTCs. If dedicated WinGrid nodes are used then the number of job assignments by the WJD is generally equal to the number of jobs received and processed by the WTCs. Thus, when the IRS/RBF simulations are run on a dedicated, one computer WinGrid node the number of job assignments from the WJD (running on the same computer) for workloads [30 workunits (IRS)], [69 workunits (IRS)], [15 workunits (RBF)] and [39 workunits (RBF)] are 30, 69, 15 and 39 respectively. The total number of job assignments for the corresponding workloads in case of 4 node and 8 node WinGrid experiments are higher. The graph also shows that the total job assignments for workload [39 workunits (RBF)] are substantially higher for the 4 non-dedicated WinGrid node experiments. As noted earlier (and depicted in graph 2), this would generally contribute to the increased execution time for RBF simulation with 39 workunits [39 workunits (RBF)] over the 4-node WinGrid test bed.

The number of work assignments made by the WJD to the WTCs, with respect to 4 node and 8 node WinGrid executions, in each of the three phases of RBF-IRS simulation are shown in graph 4 and graph 5 respectively. The graphs show that the number of job assignments made during phases one (create profiles) is usually higher than the other two phases. This is because the IRS-RBF application takes more time to complete the execution of phase one. Using non-dedicated WinGrid resources for IRS-RBF application processing suggests that the likelihood of WTCs discontinuing processing of larger work units in phase 1, because of user

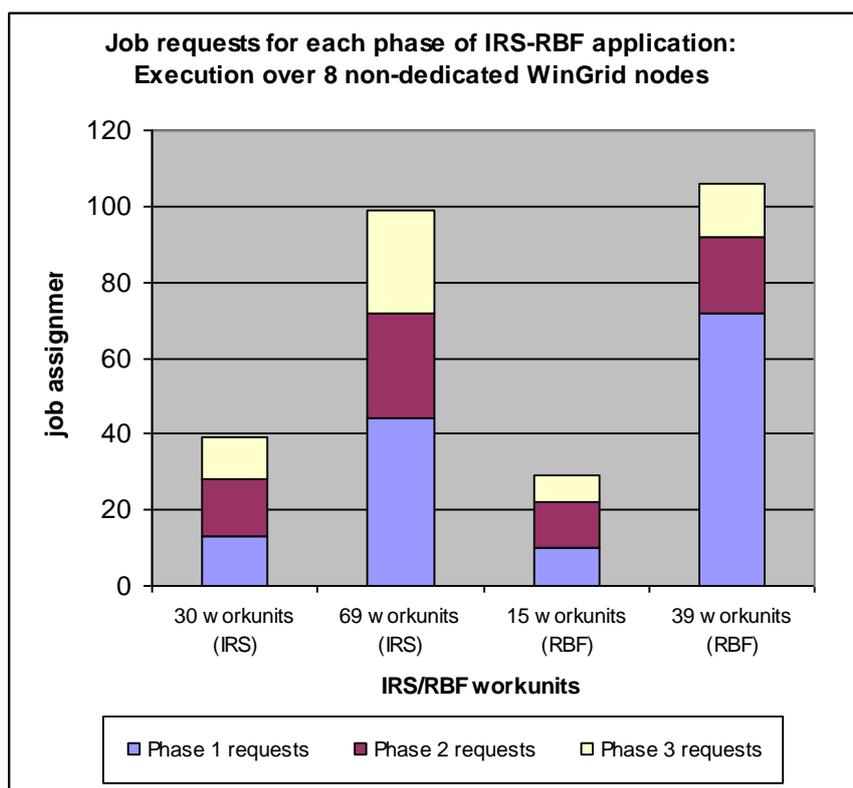
intervention, is generally higher compared to the other two phases. However, this ultimately depends on the resource usage pattern during the actual execution of a simulation.



Graph 3: Total job assignments for IRS-RBF simulation



Graph 4: Job assignments for different phases of IRS-RBF simulation (4 nodes)



Graph 5: Job assignments for different phases of IRS-RBF simulation (8 nodes)

5.7.6 Evaluation of workflow and SMMD task farming service

The real-world investment bank case study had two evaluation criteria. One, that the solution is implementable for both workflow service and SMMD task farming service. This would, in turn, mean that both these services were realizable using WinGrid. Two, in the case of SMMD task farming service the speed of execution of the IRS-RBF application using non-dedicated WinGrid nodes is faster than using one dedicated computer. The discussions in section 5.7.3 have shown that the solution is implementable. The results of the simulation presented in the earlier section have shown that criterion two has also been met. It can therefore be concluded that the case study has met both the evaluation criteria and WinGrid can be used to support the CSP-specific services pertaining to workflow and SMMD task farming.

5.8 NBS case study for evaluation of distributed simulation service

Distributed simulation can be used to address various requirements. These are highlighted in section 2.6 of this thesis. The purpose of the NBS distribution simulation is to attempt faster model execution using a set of distributed processors (case study evaluation criteria two). The simulation models used in the investigation of the NBS case study have been developed by Korina Katsaliaki from the University of Southampton as part of her PhD (Katsaliaki, 2007). Table 39 presents a summary of the technologies used and the evaluation criteria for this case study.

Table 39: NBS case study

CSP-specific service	Grid Middleware	MCS / DES CSP used	Case study evaluation criteria
Distributed simulation service	WinGrid with HLA-RTI	Simul8 Professional (<i>DES CSP</i>)	(1) Solution is implementable and the service is realizable (2) Execution is faster over dedicated grid resources compared to standalone execution

The overview of the NBS supply chain is presented in section 5.8.1. This is followed by a discussion on the standalone NBS model, originally built for execution on a single computer, (section 5.8.2) and its distributed counterpart (section 5.8.3). Sections 5.8.4 and 5.8.5 then describe the alternative HLA time advance mechanisms for the grid version of the NBS simulation and the technology used for their respective implementations. The experiments conducted using the standalone and the distributed versions of the NBS simulation and their results are presented in sections 5.8.6 and 5.8.7, followed by a discussion on the suitability of WinGrid in providing a distributed simulation service to DES CSPs (section 5.8.8).

5.8.1 Overview of the National Blood Service (NBS) supply chain

The UK NBS is a part of the National Health Service Blood and Transplant (NHSBT) organization. NHSBT was formed on 1st October 2005 as a result of the merger of the National Blood Authority (NBA) (which manages the NBS, Bio Products Laboratory and the International Blood Group Reference Laboratory) and UK Transplant (NHS Blood and Transplant, 2006). The NBS is responsible for collecting blood through voluntary donations, testing the blood for ABO and Rhesus grouping and infectious diseases such as HIV, processing the blood into around 120 different products (of which the main three are Red Blood Cells (RBC), plasma and platelets), storing the stockpile and transferring excess stock between different NBS centres, and finally issuing the different blood products to the hospitals as per their needs. The NBS infrastructure consists of 15 Process, Testing and Issuing (PTI) centres which together serve 316 hospitals across England and North Wales. Each PTI Centre thus serves around 20 hospitals. The NBS simulation has been modelled with inputs from the Southampton PTI Centre.

Blood products are stored in PTI centres until they are requested by the hospitals served by that centre. A hospital places an order for blood products when its inventory falls below a predetermined order point, or when rare products not held in stock are requested for particular patients. Hospitals normally receive their orders daily and the blood remains in the hospital bank until it is cross-matched (tested for compatibility) for a named patient. It is then placed in “assigned inventory” for that patient for a fixed time after the operation. If it is not used, it is returned to “unassigned inventory” and can be cross-matched again for another patient. On average a unit will be cross-matched four times before it is either used or outdated. In practice, however, only half of the cross-matched blood is actually transfused.

This clearly represents a huge potential for savings since the cost of a single unit of RBC is approximately £132.

5.8.2 Conventional approach to modelling the NBS supply chain

In their original study, Katsaliaki and Brailsford (2007) modelled the NBS Southampton PTI and included only RBC and platelets, which together comprise 85% of issues and are the chief source of wastage and shortages. The model was originally built using the DES CSP Simul8 Professional and was meant for standalone, one computer execution. In this section the original NBS model (henceforth referred to as the conventional NBS model) is described. This discussion is important because the conventional NBS model will be subsequently compared to its distributed counterpart with regards to execution speed. This comparison is needed because the second criteria for evaluating the CSP-specific distributed simulation service is that, a distributed simulation running over dedicated grid resources will run faster than its standalone counterpart. Otherwise, the simulation practitioner may not have any need for distributed simulation. Other potential benefits of using distributed simulation, for example saving time and cost associated with creating new simulations through the linking of existing models, wider participation of simulation practitioners in a geographically distributed experiment, etc. are not considered for the purpose of this research.

There are two main categories of entities in the model; items and orders. Items are the individual blood units (RBC and platelets) delivered from the NBS PTI centre to the hospitals in a one-way direction, since returns of products are not allowed. Orders are placed by the hospital blood bank managers to the NBS PTI centre for blood products. Requests are matched with items according to their characteristics (attributes) and delivered as appropriate.

The NBS model starts from a representative state to eliminate the need for warm-up. While the model runs, data such as the day and time of placing an order, the requested blood product (RBC or platelets), the amount by blood group, etc. are reported to an Excel file. The model advances time in simulated minutes but the hospitals' blood bank stock for placing orders to the NBS PTI is checked only every simulated hour. Likewise, the blood stocks which are ready to be delivered from the NBS PTI centre to the hospital(s) are also checked only once every simulated hour. Blood products are perishable by nature and it is important to keep an account of their remaining shelf-life. The shelf-life of a blood product is therefore decreased by the minute. Thus, it is likely that Simul8 schedules a "bound" event for each unit of RBC or platelet present in the system at every simulated minute, which brings down the shelf life of the blood product by one minute.

The conventional model contains the processes of the NBS PTI Centre, from the collection of blood to the delivery of blood products, and the processes of a single medium-volume hospital. The model captures physicians' requests for blood and the processes whereby the hospital blood bank checks its stock levels and places orders. The order entities and item

entities are represented as information flow (hospital orders) and material flow (blood products) respectively. A single supply centre and hospital is shown in figure 45. Figure 46 shows a simplified diagram showing the relationship between four hospitals and one supply centre, all of which are being executed by a single running instance of Simul8 on a single PC.

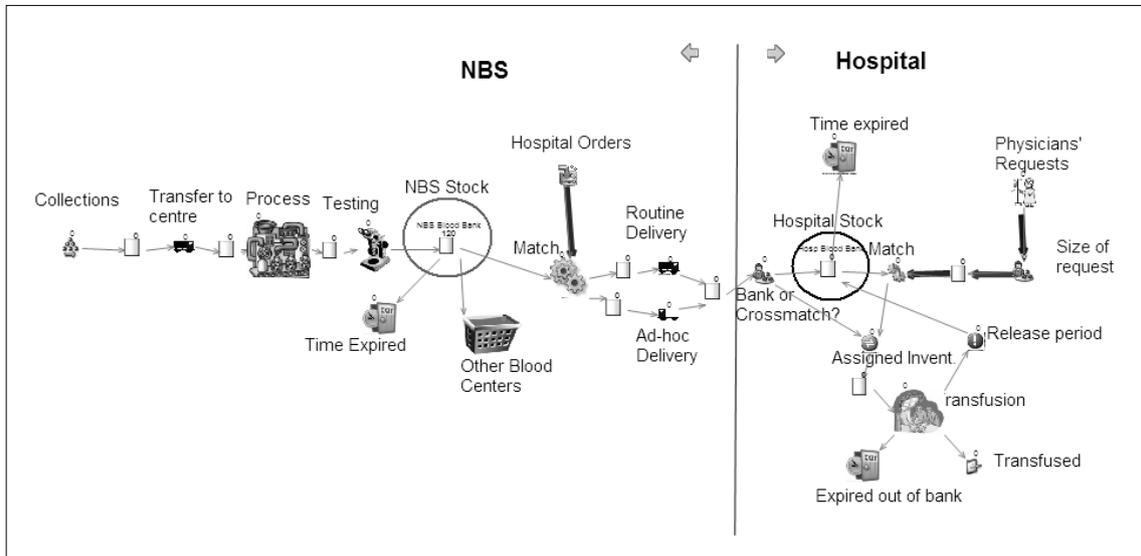


Figure 45: Simplified model of the NBS supply chain with NBS PTI (left) and one hospital

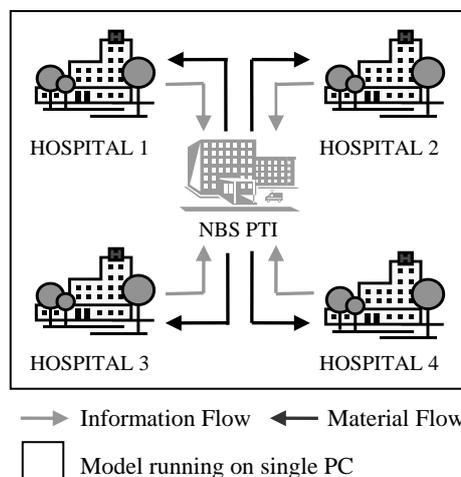


Figure 46: Conventional simulation approach with NBS PTI and four hospitals

The runtimes recorded during the execution of the conventional models (i.e., model with NBS PTI and one hospital, model with NBS PTI and two hospitals, etc.) were as follows. A single NBS supply centre with a single hospital, as shown in figure 45, took approximately 14 minutes to run for a whole simulated year on a 1.7GHz processor desktop PC with 1GB RAM. However, the runtime increased dramatically as more hospitals were added to the model. For a model with a single supply centre and two hospitals the execution time was 78 minutes, with three hospitals it was 17.5 hours and for a single supply centre and four hospitals, as represented in figure 46, the execution time was 35.8 hours (even after considerable help and

advice from the package vendor on model profiling). There are around 16 hospitals in the Southampton area, and the observation that the execution time dramatically increases with the addition of every new hospital to the model may mean that modelling the entire NBS simulation using one model is infeasible.

As has been discussed in chapter two, distributed simulation is a technique that can use the resources of many computers to execute a simulation model and can potentially reduce simulation runtime (Fujimoto, 2003). The distributed simulation service, which can be potentially provided through the use of grid computing, may make it possible to reduce the execution time through access to multiple grid resources. For faster execution of the NBS models over the grid, the processes modelled by the conventional NBS model will have to split into individual well-defined models. For example, it may be possible to split the conventional model of PTI and two hospitals into three separate models, where one model executes the PTI and two other models simulate one hospital each. These models may be able to run over three separate dedicated grid nodes to realize a distributed simulation federation. This further requires the use of a distributed simulation middleware like HLA-RTI because Simul8 does not presently support distributed simulation (refer to table 14 in chapter two). However, it does offer a set of interfaces that can be called upon by an external program to control the Simul8 simulation engine (refer to table 10 in chapter two). Using these Simul8 open interfaces together with the WinGrid middleware (that has earlier been shown to offer potential support for HLA-RTI based distributed simulation service) and the WinGrid-CSP integration technology (presented in section 4.4), it is considered worthwhile to investigate the distributed simulation service in the context of NBS distributed supply chain simulation.

5.8.3 Distributed approach to modelling the NBS supply chain

For the benefit of the reader, a short overview of HLA-based distributed simulation and CSPI-PDG is presented next. For a more thorough discussion the reader is referred to section 2.6.3.1. A distributed simulation generally requires the use of a distributed simulation middleware to synchronize the simulation time of the individual simulations and to transfer messages between them. HLA is an IEEE 1516 standard for distributed simulation. HLA-RTI is a distributed simulation middleware that implements the HLA standards. The two frequently used HLA terminologies are *federation* and *federate*. A federation is defined as a HLA-based distributed simulation. Each individual simulation, usually running on a separate computer, is a federate and together they make up a federation. CSPI-PDG is a SISO product development group that has proposed standards for the interoperation of CSPs with HLA. It is thus considered appropriate to use the CSPI/HLA standards for implementing a distributed simulation with Simul8 and the HLA-RTI middleware.

The NBS distributed simulation that is meant to execute over WinGrid nodes is modelled by dividing the conventional NBS model into individual models of the Southampton NBS PTI and hospitals. Each model is simulated by a Simul8 federate. Each federate is a combination of

DES CSP Simul8, the Simul8 model and the associated WinGrid-CSP integration code (Simul8 adapter). The Simul8 federates, running over multiple WinGrid nodes, together form the NBS federation. The interaction between federates is through messages that represent the interaction of one model part with another (e.g., messages are sent when an entity leaves one part of a model and arrives at another). The HLA-RTI middleware is responsible for transporting these messages between federates (and therefore between different computers) and to synchronize their simulation time. The reader is reminded that the HLA-RTI does not use the underlying communication channels that have been opened by the grid middleware. Furthermore, the NBS distributed simulation uses the *application integration approach* to provide distributed simulation service over the grid (section 3.3.5.2). In this approach WinGrid is only responsible for starting simulations over the WinGrid nodes; the management of the distributed simulation itself is the responsibility of the code that is associated with the individual models. The HLA-RTI middleware that has been used in this research is the DMSO HLA-RTI 1.3-NG middleware (US Department of Defense Modelling and Simulation Office, 1999). The NBS supply chain federation consisting of the individual Simul8 federates and the HLA-RTI middleware is shown below (figure 47). The Simul8 federates are executed over the WinGrid nodes and the DMSO HLA-RTI 1.3-NG process (*rtiexec*) is executed on a separate computer.

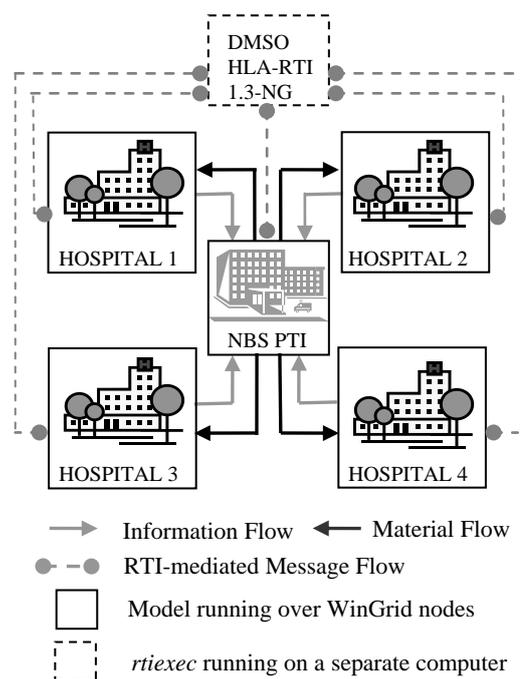


Figure 47: NBS distributed simulation with NBS PTI and four hospitals

In the distributed NBS simulation the HLA-RTI is presented as a black box. Each Simul8 federate running over a different WinGrid node executes the model of either the NBS PTI supply centre or a hospital. In this investigation the communication between the Simul8 models is achieved through HLA interactions (Kuhl et al., 1999). Interactions are HLA defined

transport mechanism for communication between federates. The reader can think of interactions as messages that are passed between two or more federates. Interactions can be time-stamped and may carry parameters. These parameters can be used to exchange information between federates. All interactions are routed through the HLA-RTI and this ensures that messages are received by federates in proper time-stamped order. In the NBS distributed simulation the interaction parameters are read from an Excel file. For example, entities representing orders are written into the file by Simul8 during the execution of hospital models. The HLA-RTI then correctly transfers this information to the NBS PTI model in the form of HLA interactions. The incoming orders from each hospital are collected into their corresponding queues in the NBS PTI model and the orders are matched with the available stock of blood. The resultant matched units are written into an Excel spreadsheet in the NBS PTI federate. This information is then sent to the different hospital models in a similar manner.

The interactions between the hospitals and the NBS PTI centre are sent every 60 minutes of simulated time, provided orders/deliveries exist. Thus, although the DES generates orders and deliveries as the model progresses in time, these are only released at specific time-steps. It is to be noted here that this time-stepped information exchange behaviour occurs as a result of the blood ordering and delivery policies in place with NBS.

The next two sections (5.8.4 and 5.8.5) will focus on HLA time management mechanisms employed for the grid-enabled NBS simulation and the Simul8-DMSO RTI integration work respectively. These discussions are important because they attempt to highlight the following:

- (1) The model characteristics are important while selecting HLA time management strategies as it bears a direct effect on the model execution speed. For the evaluation of grid-facilitated distributed simulation service, the HLA time management scheme that executes the distributed NBS simulation in the shortest time will be used for comparison with the conventional NBS model.
- (2) The complexity involved in integrating CSPs with distributed simulation middleware. This discussion complements the earlier discussion on WinGrid-CSP integration technology (section 4.4). This, in turn, has a direct bearing on the practicality of implementing CSP-based distributed simulation over the grid.

5.8.4 HLA time management mechanisms used in NBS distributed simulation

As with most distributed simulations, the NBS models being executed on different WinGrid nodes need a mechanism to synchronize their simulation time and to ensure the current ordering of events. In a standalone simulation the event list consists only of events generated internally by the “single” running instance of the model. These events are termed as *internal events*. In a distributed simulation each federate also receives events from other federates. These events are termed as *external events*. The event list of a distributed federate, therefore, has to correctly time order both the internal events and the external events and execute them without any error. However, due to (1) latencies in the network, (2) different

processing requirements for different models, and (3) different hardware configurations of machines (which may mean that simulations are being executed at different speeds), it is not possible to guarantee when external events will arrive at a federate.

In order to ensure that in a federation events arrive at each federate at the causally correct order, the time management services of the HLA are employed. The HLA defines at least two variants of the conservative time synchronization mechanism that can be invoked by a federate to request a simulation time advance from HLA-RTI – the *Next Event Request* (NER) and the *Time Advance Request* (TAR) (Kuhl et al., 1999). Both these mechanisms are implemented by the HLA-RTI middleware in the form of HLA service calls.

NER and TAR service calls are invoked by a federate with a time component that represents the logical time the federate wishes to move to. Depending on whether NER or TAR is called by the simulating federate, the time granted to it by the HLA-RTI can be different. NER will grant the federate a time that is either less than or equal to the requested time depending on whether external events are present, and if yes, their timestamps. If an external event with a timestamp less than the requested time exists, then the time granted by the HLA-RTI will be equal to the timestamp of the external event. If no external events exist or an external event with timestamp equal to the requested time is received, then the HLA-RTI will grant the federate the requested time. TAR, on the other hand, will grant the simulation federate a time that is exactly equal to the time requested by a federate. Until the requested time can be safely granted to the federate (i.e., it can be assured that no causality error will occur), the HLA-RTI will not send the time grant message.

This research has implemented both NER and TAR versions of NBS distributed simulation over the grid. This has been done in order to demonstrate that the selection of an appropriate HLA conservative time management mechanism should be made not only based on the internal characteristics of the simulation, but consideration should also be given to the characteristics of the message flow between models. A further motivation has been to use the time management method that would give better performance results in terms of execution speed. Irrespective of the time management mechanism used, the simulation results are identical in case of the NBS distributed simulation.

5.8.5 Grid-enabling NBS distributed simulation

This section of the thesis builds on the WinGrid-CSP integration architecture (presented earlier in section 4.4) with the objective of enabling distributed simulation over the grid. This is subsequently referred to as WinGrid-DMSO_HLA_RTI-Simul8 integration architecture. The software component that implements this architecture is referred to as *CSP Controller Middleware (CCM)*. The CCM interacts with both the Simul8 Professional Edition and the DMSO HLA-RTI to realize a Simul8-based distributed simulation. Each of these two tasks is performed by two distinct components of the CCM: the *Simul8 adapter* and the *RTI adapter*.

The communication between these adapters is via Java Native Interface (JNI) and Jacob technologies. The Simul8 adapter utilizes the Simul8 Professional COM interface to access the Simul8 simulation engine. The interaction of the RTI adapter with DMSO HLA-RTI is through java-defined HLA interface bindings. The CCM has a separate implementation for TAR and NER time advance mechanisms, referred to as *CCM-TAR* and *CCM-NER* respectively. The architecture of the CSP Controller Middleware is shown in figure 48. The message exchange protocol followed by CCM-TAR and CCM-NER can be found in **Appendix B**.

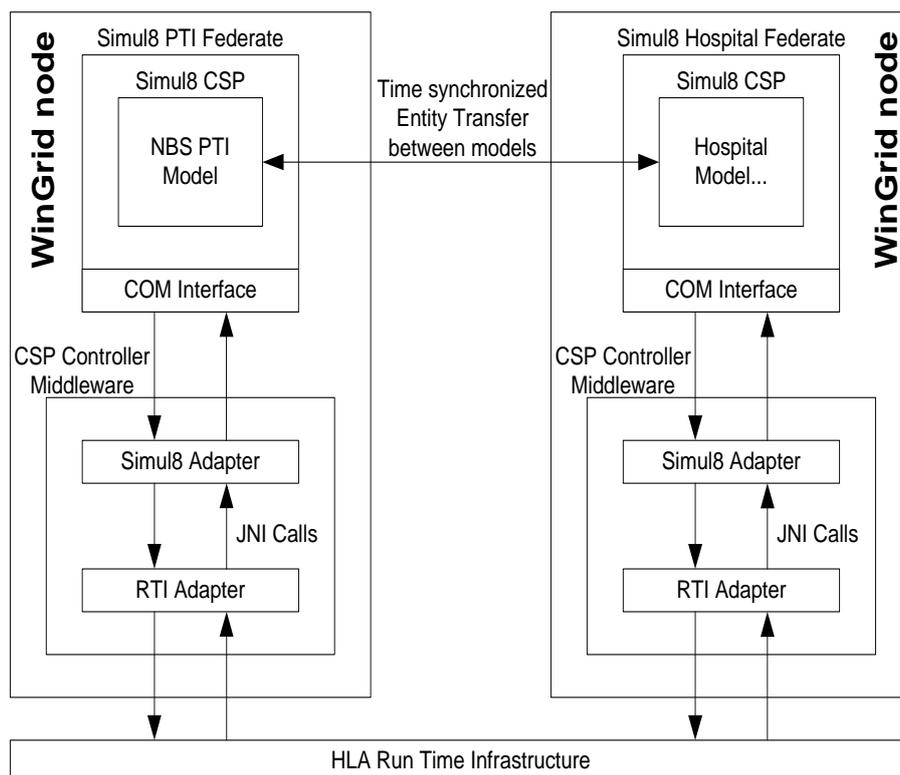


Figure 48: CSP Controller Middleware (CCM) architecture

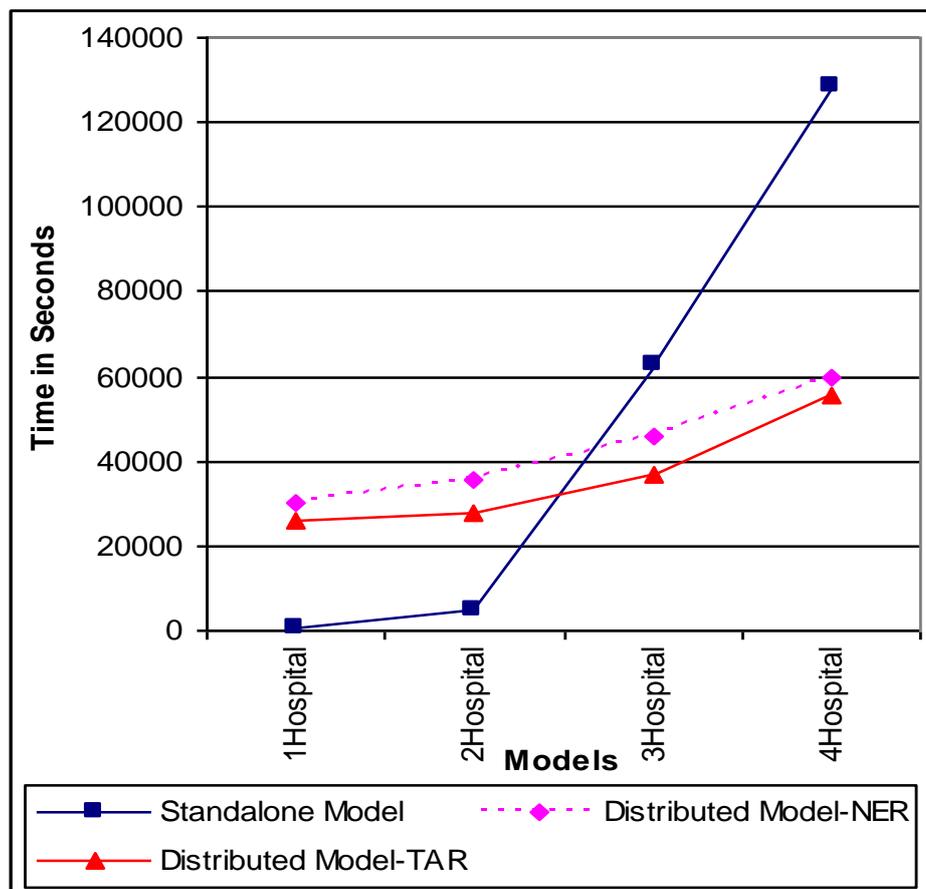
5.8.6 Experiments

To investigate the performance of the NBS standalone simulation with (1) NBS distributed simulation over WinGrid using the NER time management service (implemented by CCM-NER) and, (2) NBS distributed simulation over WinGrid using the TAR time management service (implemented by CCM-TAR), four different scenarios were designed. Each scenario was represented by one NBS PTI centre serving one, two, three or four hospitals respectively. The name of the scenario reflects the number of hospitals that the NBS PTI caters for. For example, scenario *2Hospital* implies that 2 hospitals are being served by one NBS PTI centre. In case of distributed NBS simulation, scenario *2Hospital* implies three separate Simul8 models, each modelling either the NBS PTI centre, Hospital1 or Hospital2 and running on three separate WinGrid nodes. In case of standalone NBS simulation, scenario *2Hospital*

suggests that a single Simul8 model, running on a single PC, has modelled the behaviour of the NBS PTI centre and two hospitals.

Experiments were conducted over dedicated WinGrid nodes. The nodes comprised of Dell Inspiron laptop computers running the Microsoft Windows XP operating system with 1.73GHz processors and 1GB RAM. The test bed also included a medium specification desktop PC to host the DMSO RTI *rtiexec* software. These computers were connected through a 100Mbps CISCO switch. The results of the execution times for each of the scenarios are based on the average of five runs.

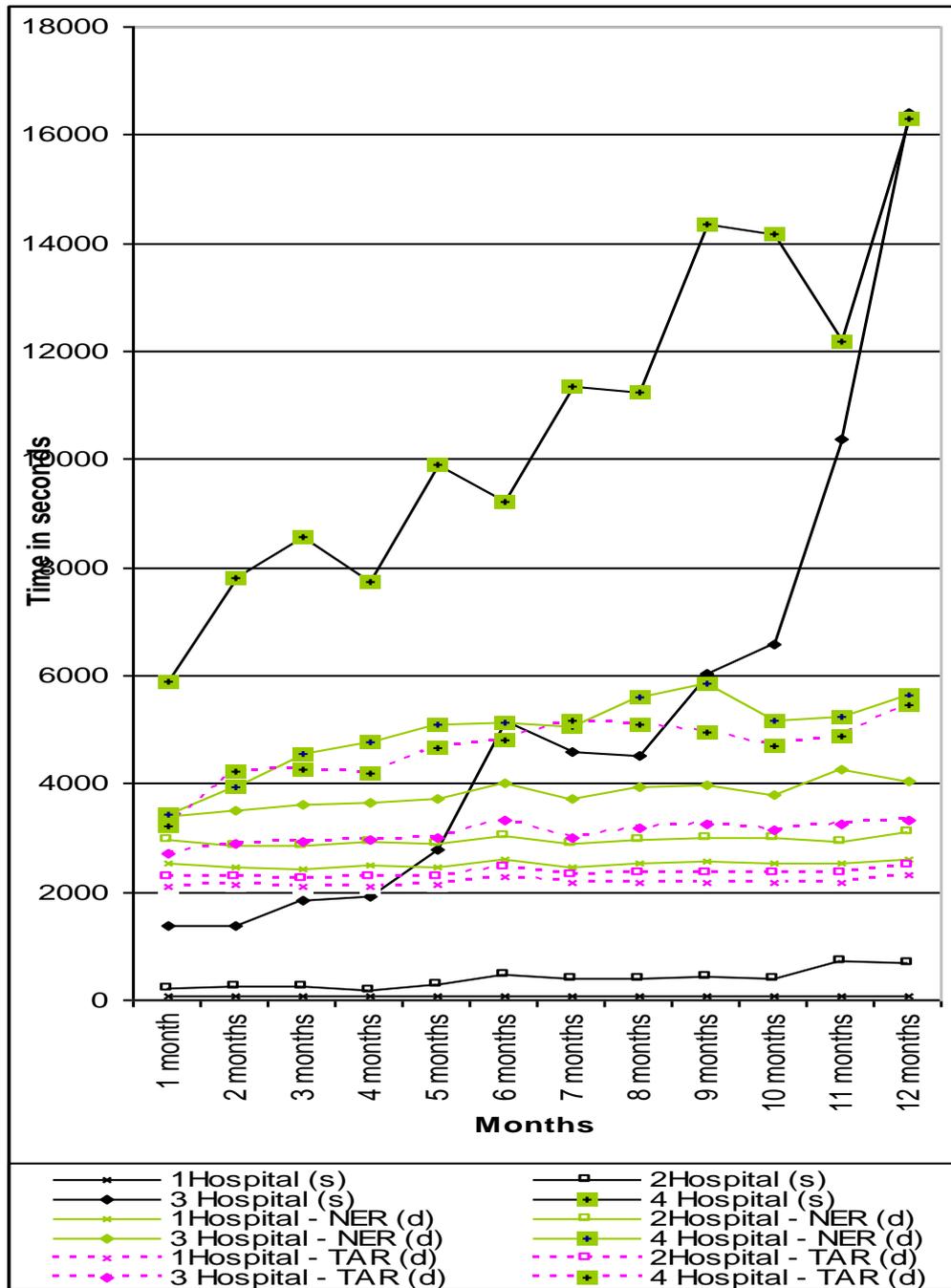
5.8.7 Results



Graph 6: Execution time of NBS distributed simulation and NBS standalone simulation

Graph 6 above shows the execution time (in seconds) using both standalone and distributed approaches for all the four scenarios. The results show that the conventional model with one hospital takes approximately 14 minutes to run for a whole simulated year. The run time rises to 78 minutes when the model runs with two hospitals and to approximately 17.5 hours with three hospitals. The addition of the fourth hospital increases the execution time to 35.8 hours. The NER version of the distributed model with one NBS supply centre and one hospital runs in approximately 8.4 hours, with two hospitals in 9.8 hours, with three hospitals in 12.7 hours and with four hospitals in 16.5 hours. The execution time for the TAR version of the

distributed model is 7.2, 7.8, 10.3 and 15.5 hours for the *1Hospital*, *2Hospital*, *3Hospital* and *4Hospital* scenarios respectively.



Graph 7: Monthly execution time of NBS distributed and standalone simulations

Graph 7 compares the time taken to execute the three versions of the NBS simulation (standalone, distributed-NER [implemented by CCM-NER] and distributed-TAR [implemented by CCM-TAR]), for every consecutive month of the year (1month to 12months) and for each of the four scenarios (*1Hospital*, *2Hospital*, *3Hospital* and *4Hospital*). The results obtained from scenarios *1Hospital* and *2Hospital* show that the conventional (standalone) version executes much faster compared to its distributed counterparts. In case of scenarios *3Hospital*

and *4Hospital* both the distributed versions outperform the execution of the standalone model. However, in all the 4 scenarios the TAR-based simulation executes faster compared to the NER-based simulation. There also appears to be a large increase in the runtime for the conventional version while increasing the number of hospitals in the model. This is quite a contrast to the substantially smaller and smoother rise in the runtime in case of both NER and TAR versions of the distributed model.

5.8.8 Evaluation of distributed simulation service

The evaluation criteria for this case study were the following:

- The WinGrid-CSP integration solution for supporting distributed simulation service can be implemented in practice. This would also mean that the CSP-specific task farming service is realizable using WinGrid.
- The execution speed of running the NBS distributed simulation using dedicated WinGrid nodes is faster compared to its standalone counterpart.

This investigation has shown that criteria one can be met by extending the WinGrid-CSP integration solution to include application code that invokes DMSO RTI defined Java bindings. In this case study this has been made possible through the implementation of the CCM (section 5.8.5). The CCM includes a RTI adapter that enables two-way communication with the DMSO RTI – it invokes HLA-defined services on the DMSO RTI and receives callback messages from it. The Simul8 adapter, also a part of the CCM, is invoked by both the WTC and the RTI adapter to execute a grid enabled version of the NBS simulation. The result of the case study has shown that criterion two has also been met. This is because the time taken to execute the distributed versions of the NBS simulation for one year of simulation time is less than its standalone counterpart.

From the results it can also be concluded that, in the case of large and complicated CSP-based models (like scenario *3Hospital* and *4Hospital*), using distributed simulation service over dedicated nodes has the potential to achieve faster performance compared to running a standalone version of the model that encapsulates all the modelling logic into one model. However, if the model is not very large or complicated then a standalone simulation should suffice for the requirements of most simulation modellers. This is aptly demonstrated by scenarios *1Hospital* and *2Hospital* where the execution of the standalone simulation is faster than its distributed counterparts, so much so that there is hardly any room for comparison.

The results of the experiments have further shown that the choice of the time management mechanism is important in the case of HLA-based distributed simulation. This is highlighted by the fact that the NBS model that implements the TAR time advance mechanism executes faster than its NER counterpart. A more detailed discussion of the results is presented in **Appendix B**. The appendix also discusses the performance results of running the standalone NBS simulation over two high specification computers. The first computer that was used had

one 3.2GHz Hyper-threaded processor with 2GB RAM. The configuration of the second computer was even higher – two dual core 2.8GHz processors, i.e., four processors, with 12GB RAM. The results concluded that, in the case of the NBS standalone simulation a higher configuration machine does not speed up the execution of the simulation. Thus, distributed simulation appears to be the only viable solution for executing even larger NBS simulations which model even more number of hospitals.

The discussions in this section have shown that both the case study evaluation criteria can be met by WinGrid and therefore it can be said that WinGrid can offer distributed simulation service to CSPs using the HLA-RTI middleware. However, it has to be added that it is not a trivial task for a simulation practitioner to effectively utilize the multiple grid nodes, made available through the use of a grid middleware like WinGrid, for running HLA-based distributed simulations. This is mainly because of the lack of integrated distributed simulation support in DES CSPs. The important prerequisites for implementing a CSP-based distributed solution over the grid are as follows:

- Knowledge of distributed simulation theory.
- Knowledge of HLA-RTI defined interfaces and time advance mechanisms.
- Knowledge of Java or C++ is essential because DMSO HLA-RTI presently provides bindings for only these two languages. These bindings can be invoked by user application code to create and join a simulation federation, to request time advance from DMSO HLA-RTI, to transfer entities, to resign and destroy a simulation federation, etc.
- Knowledge of CSP-defined interfaces (and their purpose) that could be invoked through the CSP adapter.

5.9 Chapter summary

This chapter has used both real-world and hypothetical case studies to examine the potential of grid middleware to support three different grid-facilitated CSP-specific services. These services were among the six CSP-specific services that have been proposed by the CSP-GC framework. Chapters 3 and 4 have identified grid middleware that can be potentially used to support some of these services. A subset of these middleware solutions has been used together with DES or MCS CSPs, in the context of different case studies, to examine their viability in relation to the realization of these services.

The case studies have been evaluated based on different evaluation criteria. These criteria are derived from the hypothesis evaluation criteria for CSP-specific services that are presented in section 5.2. Section 5.3 has then outlined the five case studies that have been used in this chapter to investigate different grid middleware in relation to CSP-specific services. The first case study (BOINC case study) has investigated PRC middleware BOINC in relation to SMMD task farming service (section 5.4). This is arguably the first attempt to use a PRC middleware in an enterprise setting. EDGC middleware Condor has been examined in

the context of MMMD task farming service in the second case study (section 5.5). The third case study (Ford case study) is presented in section 5.6 where WinGrid was successfully used to grid-enable a propriety simulation tool used by the Ford Motor Company at their Dunton Technical Centre in Essex. The investment bank case study is the fourth case study where WinGrid was used to successfully provide workflow service and task farming service to a MCS CSP Analytics-based simulation application (section 5.7). In the fifth and the final case study (NBS case study) WinGrid has been used together with DES CSP Simul8 Professional to facilitate realization of a distributed blood supply chain simulation (section 5.8). In summary, the case studies described in this chapter have shown that the case study evaluation criteria for the different CSP-specific services have been met.

These case studies have demonstrated the following:

- Grid middleware can be used with unmodified MCS and DES CSPs, which expose package functionalities through well-defined interfaces, to provide CSP-specific services.
- All the case studies have followed a common approach with regards to the use of grid technology together with the CSPs, namely, interfacing of grid middleware with MCS and DES CSPs was done using the CSP-grid integration technology. It has been shown that this approach not only works with WinGrid, a middleware that has been developed by the author, but also with two other widely used grid middleware (BOINC and Condor).
- The CSP-grid integration technology allows GUI programs, which are installed on local grid resources, to be executed over the grid. It can be argued that the use of grid technology has primarily been associated with the execution of non-interactive and non-GUI applications, wherein the entire application code is transferred over to the grid node responsible for its execution. In case of CSP-grid integration technology, only the trigger code which interfaces the grid middleware with the MCS or DES CSP is transferred.
- Functionalities that are not presently supported by MCS or DES CSPs, for example, DES CSP Witness does not support task farming, MCS CSP Analytics does not support workflows, DES CSP Simul8 does not support distributed simulation, etc., can be potentially provided by grid middleware through the use of CSP-specific services (as has been shown in the case studies).

The next chapter evaluates the CSP-GC framework based on earlier discussions pertaining to the suitability of BOINC (chapter 3), Condor (chapter 3), WinGrid (chapter 4) and WinGrid-WS (chapter 4) in providing the CSP-specific services, and complemented by the results of the experimental evaluation of some of the grid-facilitated services that are presented in this chapter.

6 REVISITING THE CSP-GC FRAMEWORK

6.1 Introduction

The previous chapter has used case study experimentation to evaluate how some of the CSP-specific services can be supported through grid middleware. Five real-world and hypothetical case studies were used to assess the following three services – SMMD and MMMD task farming service, workflow service and distributed simulation service. The case studies experimented with a total of three different grid middleware (BOINC, Condor and WinGrid), two MCS CSPs (Excel and Analytics) and two DES CSPs (Witness and Simul8). Finally, they were evaluated using case study evaluation criteria.

This chapter evaluates the CSP-GC framework. The purpose of this evaluation is to examine, based on the grid-CSP specific discussions and case studies presented in the earlier chapters, whether the six CSP-specific services that were identified by this framework could be supported by the existing grid computing middleware. The evaluation of the framework, in turn, tests the hypothesis of this research. This chapter is structured as follows. Section 6.2 identifies a potential new CSP-specific service that is a combination of distributed simulation service and SMMD / MMMD task farming service, and investigates grid middleware in relation to this service. A case study is presented in section 6.3 to evaluate whether it is technically feasible for grid middleware to support this new service. Section 6.4 then evaluates the CSP-GC framework based on (1) middleware support for CSP-specific services that were discussed in chapter 3 (BOINC and Condor) and chapter 4 (WinGrid and WinGrid-WS), (2) middleware support for the new CSP-specific service, identified in section 6.2 of this chapter, and (3) the results of the case studies presented in chapters 5 and 6. Finally, the modified CSP-GC framework is presented in this section. The existing CSPs that support some of the CSP-specific services through custom solutions are outlined next in section 6.5. The chapter concludes with the chapter summary in section 6.6.

6.2 Distributed simulation with SMMD and MMMD task farming service

The case studies in chapter 5 of this thesis have shown that grid middleware can provide task farming service and distributed simulation service. Combining both these services could enable a simulation practitioner to run multiple instances of a distributed simulation over the available grid nodes. Federates taking part in each such distributed simulation federation could run separate sets of experiments. Although it could be argued that using the distributed simulation service together with the task farming service would implicitly enable the distributed simulation with task farming service, further investigation is needed because the grid middleware being used should be able to schedule the execution of *multiple instances of groups of distributed models* concurrently. Furthermore, the HLA-RTI middleware for distributed simulation should allow simultaneous execution of more than one distributed

simulation federation. SMMD and MMMD variants of task farming in relation to distributed simulation are discussed next.

In the context of distributed simulation, SMMD task farming refers to the execution of *multiple instances of the same distributed model*, where each distributed model comprises of two or more individual models, over the desktop grid. Figure 49 below shows an example of this. The distributed simulation consists of three different federates, A, B and C. There are three federations, X, Y and Z, running three different instances of the same distributed model over the desktop grid. The names of these federates are appended with the federation they belong to (X.A, X.B, Y.A, Z.A and so on.).

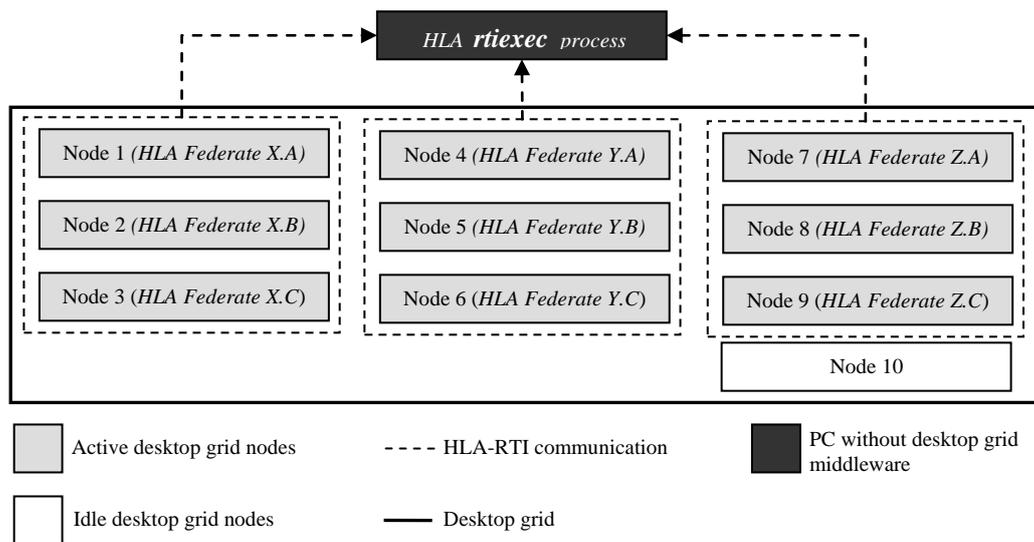


Figure 49: Distributed simulation with SMMD task farming

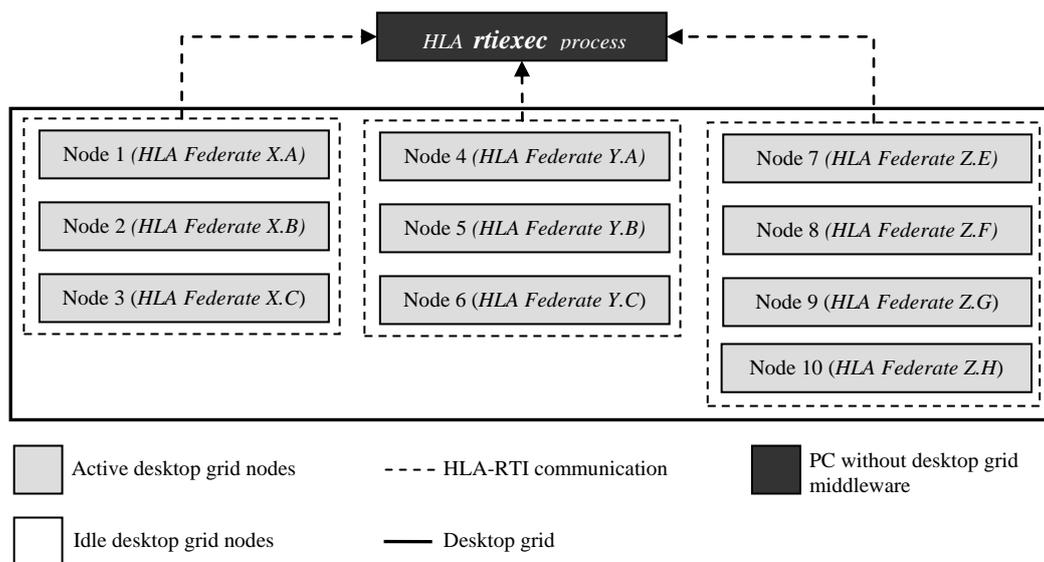


Figure 50: Distributed simulation with MMMD task farming

In the context of distributed simulation, MMMD task farming refers to the execution of *multiple instances of different distributed models*, where each distributed model comprises of two or more individual models, over the desktop grid. Figure 50 shows an example of this. The first set of the distributed simulation model comprises of federates A, B and C. Two instances of this model are being executed under HLA federations X and Y. Federation Z is executing the second set of the distributed simulation model that comprises of federates E, F, G and H.

Previous investigations of BOINC, Condor and WinGrid have shown that distributed simulation service can be potentially provided by all three middleware (section 5.3). In the case of WinGrid it has also been demonstrated through the NBS case study (section 5.8) that this solution is implementable. Since all three middleware can support both distributed simulation and task farming services, it is important to examine all of them in relation to this new service. The discussion is limited to the SMMD form of task farming. However, it can be argued that a middleware that supports execution of “groups of separate models” concurrently may equally be able to support MMMD task farming.

6.2.1 Investigation of distributed simulation with task farming using Condor

Jobs submitted to a grid by users are generally held in a queue and scheduled for execution on available desktop nodes on First In First Out (FIFO) basis. In this case the only consideration for execution of a user job is its order in the job queue.

Condor provides a command (*condor_prio -p [+/- priority value] [job number]*) to assign priorities to each submitted job in order to control the order of execution (Condor Version 6.9.1 Manual, 2007c). The default priority of a submitted job is 0. The priority of a job, identified by a job number, can be set to a value between -20 and +20 using the *-p* switch of the *condor_prio* command. Once a job has been submitted the job number can be known using the command *condor_q* (see screenshot 16 in section 5.5.4). This allows the Condor job scheduler to schedule a job based on both its order in the job queue and its priority, i.e., jobs with same priority are usually scheduled in FIFO order. However, this can only happen if the Condor matchmaking agent has successfully matched jobs to compute resources (section 2.10.1).

It has been discussed earlier in section 3.4.5.2 that Condor Java universe (together with HLA-RTI) can potentially provide distributed simulation service to CSPs. It is therefore important to investigate whether SMMD task farming capabilities can be layered on top of it. SMMD task farming of distributed models requires that multiple distributed simulation federations are executed over desktop grids with varying experiment parameters. This necessitates the submission of multiple sets of work, wherein each set comprises of the different models that make up a distributed simulation. For example, model-A and model-B are two distributed models that are considered as one set of work. For running three federations over a desktop grid, three such sets of work will have to be submitted by the user. The models that logically

constitute each set of work can be executed as separate condor jobs. Thus, model-A and model-B (and associated code) are submitted as job-A and job-B respectively. For running three distributed simulation federations over Condor, three instances each of job-A and job-B will have to be submitted along with different experiment parameters.

A user can submit multiple instances of the same job by providing a value for the Condor defined *queue* variable in the job's "job description file" (section 5.5.4). The value assigned to this variable determines the number of instances of a particular job. For example, if the job description file for job-A specifies a value of 3, then 3 instances of job-A are created and placed in the queue. Similarly, three instances of job-B can be appended to the queue (screenshot 24 below). All the 6 jobs now have default priority and jobs will be scheduled on the available resources on FIFO basis (if matchmaking is successful). In this case, FIFO job scheduling can present problems if less than four machines are available over the grid. This is because the first three jobs in the queue are instances of job-A and they will be scheduled on the first three idle resources (say, R1, R2, R3) after successful matchmaking. But the distributed simulation cannot run without execution of job-B. However if a fourth resource (say, R4) becomes available, then the first instance of job-B (which is now the first job in the queue because the job-A instances have already been assigned) will be scheduled on it. This will start the execution of the first simulation federation over R1 and R4. At this point resources R2 and R3 are still waiting for execution of the two remaining instances of job-B over other desktop nodes.

```
C:\Condor\examples\nav>condor_q
-- Submitter: navonil : <192.168.0.254:1072> : navonil
ID      OWNER          SUBMITTED  RUN TIME ST PRI  SIZE  CMD
17.0    nice-user.Admi  3/4  23:05  0+00:00:00 I  0    0.4  job-A
17.1    nice-user.Admi  3/4  23:05  0+00:00:00 I  0    0.4  job-A
17.2    nice-user.Admi  3/4  23:05  0+00:00:00 I  0    0.4  job-A
18.0    nice-user.Admi  3/4  23:06  0+00:00:00 I  0    0.4  job-B
18.1    nice-user.Admi  3/4  23:06  0+00:00:00 I  0    0.4  job-B
18.2    nice-user.Admi  3/4  23:06  0+00:00:00 I  0    0.4  job-B
6 jobs; 6 idle, 0 running, 0 held
```

Screenshot 24: Condor queue after submission of multiple instances of job-A and job-B

By using the *condor_prio* command it may be possible to assign priorities to the submitted jobs, such that **sets of jobs** (in our example job-A and job-B are one set) are scheduled on a FIFO basis. Thus, instance one of job-A (first position in queue) and job-B (fourth position in queue) can be assigned a priority of +20. The second instance of job-A (second position in queue) and job-B (fifth position in queue) can be assigned a priority + 15, and so on. The Condor scheduler will then try to assign the jobs with the highest priorities first. In our example, resource R1 and R2 will be assigned job-A and job-B with priority +20, resource R3 and R4 will be assigned jobs with priority +15 and so on.

In SMMD task farming the number of experiments to be performed can be in their tens or hundreds, and each experiment will involve at least two distributed models. Manually changing the priorities of such a large number of submitted jobs is not practical. Furthermore, the *condor_prio* command only attempts to change the job priority. Finally, any priority change is only compared to the priority of other jobs owned by the same user and submitted from the same machine (Condor Version 6.9.1 Manual, 2007c). Owing to these limitations, the use of Condor Java universe for providing distributed simulation with task farming service to CSPs is not considered appropriate.

6.2.2 Investigation of distributed simulation with task farming using BOINC

Investigation of BOINC has previously shown that it can potentially provide distributed simulation service to CSPs using HLA-RTI (section 3.4.5.1). A BOINC project usually consists of one application client that is downloaded along with associate files (initialization files, CSP models, DLLs, etc.) by the client computers. In the context of distributed simulation, BOINC workunits can be created such that they pass different CSP model names and experiment parameters as arguments to the application client for execution.

In the case of BOINC, layering SMMD task farming over distributed simulation support will usually involve the creation of multiple **sets of BOINC workunits**, each set comprising of (1) the individual models and associated code that make up the distributed simulation, and (2) the simulation parameters. For example, with 10 SMMD experiments to be conducted on two distributed models (model-A and model-B), a total of 20 BOINC workunits will be required. Logically, they can be thought of as 10 sets of workunits for model-A and model-B respectively. Unlike Condor there is no easy way to change the priorities of these workunits and therefore FIFO scheduling of **sets of BOINC workunits** may not be possible. Thus, the BOINC core clients running on multiple PCs will generally “pull” the workunits from the server without any consideration to the underlying simulation models that the workunits represent. The user may therefore be presented with a situation wherein the different desktop grid nodes attempt to execute different instances of the same simulation model, i.e., all clients try to execute model-A. Because of this limitation BOINC is considered unsuitable to effectively implement distributed simulation with SMMD task farming.

6.2.3 Investigation of distributed simulation with task farming using WinGrid

Investigation of BOINC and Condor in the previous sections has shown the limitations of a general purpose desktop grid middleware in running multiple sets of distributed simulation experiments. This research has learned from these shortcomings and has implemented a version of WinGrid that schedules jobs taking into consideration the individual distributed simulation models that the jobs represent. Thus, if there are 10 SMMD task farming experiments to be performed on a 3-federate distributed simulation, WinGrid places 30 jobs in the WinGrid job queue based on **sets of job** (each set of job comprises of 3 individual jobs,

wherein each job represents one distributed simulation model) and schedules them on available resources on a FIFO basis. This is shown in screenshot 29 (section 6.3.3).

The discussion in this section have shown that WinGrid middleware can be potentially used to provide distributed simulation with SMMD task farming support to CSPs. As indicated in table 40 below, HLA-RTI will have to be used for providing this service.

Table 40: Grid middleware support for distributed simulation with task farming service

CSP-specific service	Grid Middleware	Comments
Distributed simulation with task farming service	<ul style="list-style-type: none"> ▪ WinGrid with HLA-RTI 	HLA-RTI distributed simulation middleware will also have to be used

The next section presents the last case study (Manufacturing Unit [MU] case study) in this thesis. The purpose of this case study is to determine whether WinGrid can be used with a DES CSP to facilitate the running of multiple distributed simulation federations over a grid.

6.3 MU case study for evaluation of distributed simulation with task farming service

The MU (manufacturing unit) case study investigates WinGrid in relation to distributed simulation with SMMD task farming service. It is a hypothetical case study. DES CSP Simul8 Professional will be used in this case study. The case study evaluation criterion is that the solution is implementable and the service is realizable (table 41).

Table 41: Manufacturing unit case study

CSP-specific service	Grid Middleware	MCS / DES CSP used	Case study evaluation criteria
Distributed simulation with task farming service	WinGrid with HLA-RTI	Simul8 Professional (DES CSP)	(1) Solution is implementable and the service is realizable

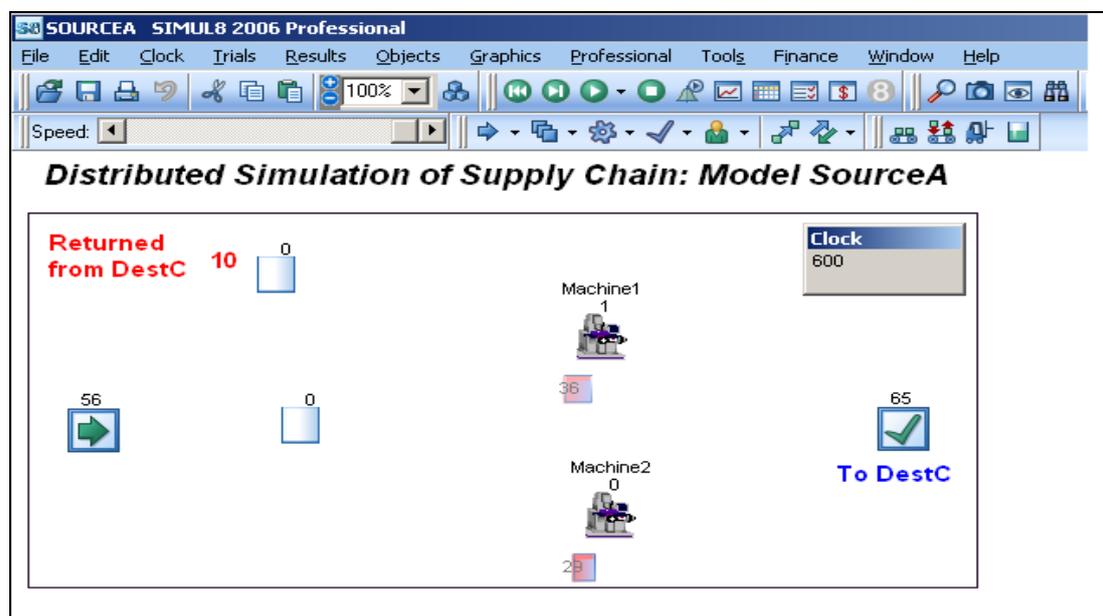
Section 6.3.1 presents an overview of the manufacturing unit case study. This is followed by a description of the distributed production line (DPL) application that would be investigated (section 6.3.2) and the technology used to grid-enable it (section 6.3.3). The section concludes with an evaluation of WinGrid in relation to distributed simulation with task farming service (section 6.3.4).

6.3.1 Overview

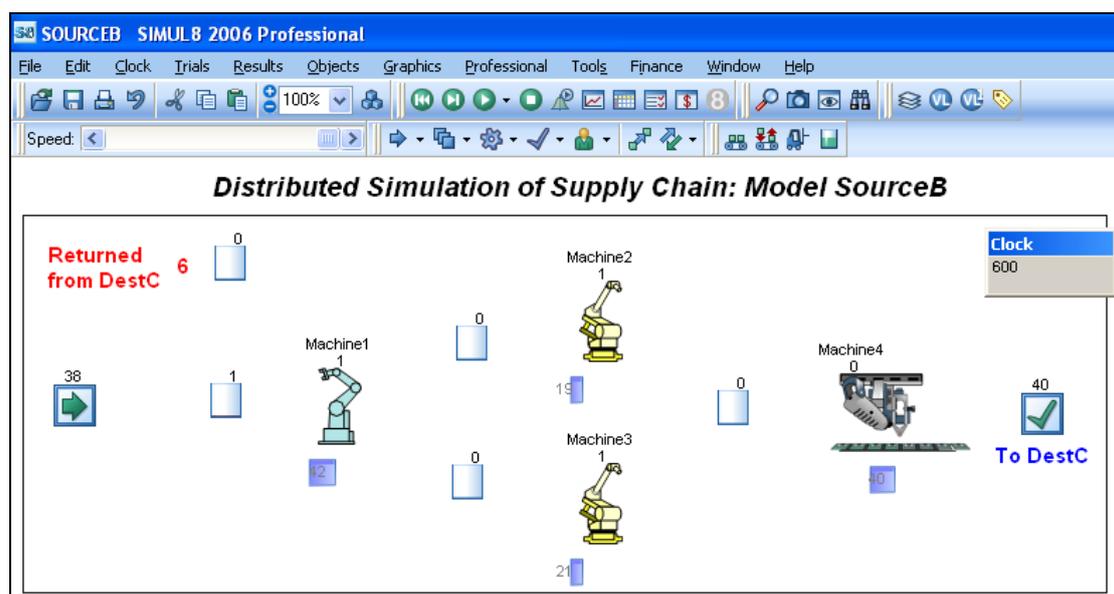
It has been shown earlier in the NBS case study that distributed simulation can potentially execute faster compared to a standalone simulation if the models being simulated are large and complex. The results have indicated that the opposite is also true. Thus, a standalone simulation model will generally execute many times faster compared to a distributed execution if the model being simulated is simple and comparatively small. However, faster model execution through the use of multiple processors is only one of the reasons for using distributed simulation. Another reason could be model reuse (section 2.6). This would allow previously created models to be linked together through the use of distributed simulation. This case study is aimed at model reuse.

6.3.2 Distributed production line (DPL) application

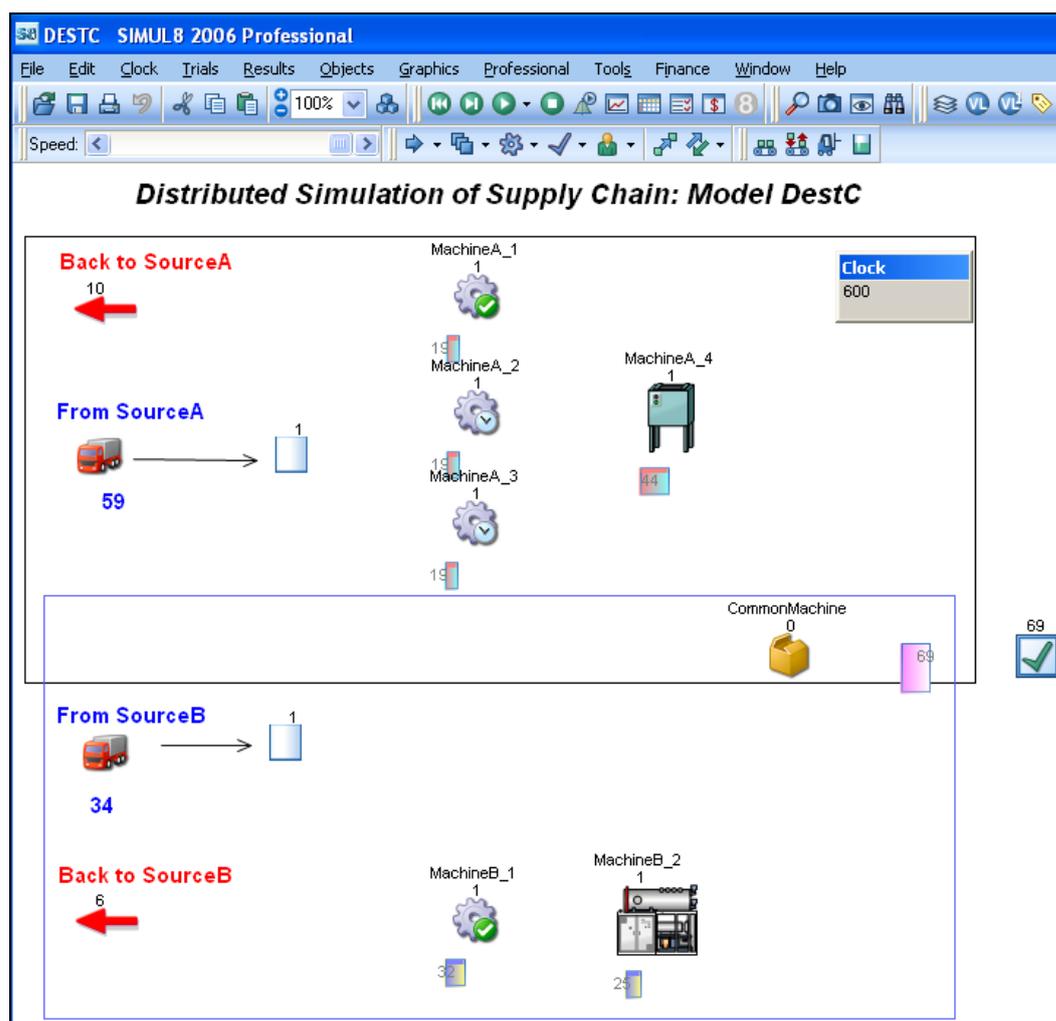
The Distributed Production Line (DPL) application consists of three separate models. It is assumed that these models were created using DES CSP Simul8 Professional by different modellers at different times. Each simulation models one manufacturing production line. The models have been named *sourceA*, *sourceB* and *destC* respectively. Models *sourceA* and *sourceB* feed entities (parts) into model *destC*. Model *destC* also feeds back entities (damaged parts) into models *sourceA* and *sourceB*. This interaction between the models represents a hypothetical production line, which comprises of three individual production lines that are geographically apart, where two different parts are manufactured separately for assembly into one final product. Screenshots of all the three Simul8 models are shown below (screenshots 25, 26 and 27).



Screenshot 25: DES CSP Simul8 model “sourceA” (DPL application)



Screenshot 26: DES CSP Simul8 model “sourceB” (DPL application)



Screenshot 27: DES CSP Simul8 model “destC” (DPL application)

6.3.3 Grid-enabling DPL application

Like the other case studies, the CSP-grid integration technology (section 4.4) has been used for implementing distributed simulation with task farming service with WinGrid and DES CSP Simul8 Professional. More specifically, the TAR version of the *CSP Controller Middleware (CCM-TAR)*, presented earlier in section 5.8.5 in the context of NBS distributed simulation, has been used. The DPL application consists of three Simul8 models and the CCM-TAR. The CCM-TAR consists of two separate components – the *Simul8 adapter* and the *HLA-RTI adapter*. The Java-based HLA-RTI adapter communicates with HLA-RTI for operations associated with distributed simulation (for example, creating a federation, joining a federation, time advance request, etc.). The VB DLL-based Simul8 adapter is used to control the simulation package. The communication between the Simul8 adapter and the HLA-adapter is through JNI. The DPL application is the WA that executes on different WinGrid nodes (WTCs). An Excel-based application called Distributed Production Line – Experimentation Tool (DPL-ET) has been created to provide experiment parameters for the different simulation experiments. After a distributed simulation run has completed, the results of the simulation are also sent back to this application. The DPL-ET application is presented in Screenshot 28

below. It shows that the experiment parameters for each model have been entered in the top half of the spreadsheet; and the results being returned are displayed in the bottom half.

Simul8 Distributed Simulation with SMMD Task Farming							
Distributed Simulation Experiment Parameters							
Model and Parameters	Experiment1	Experiment2	Experiment3	Experiment4	Experiment5	Experiment6	Experiment7
sourceA							
sourceA_Machine1 ProcessingTime	5	6	7	8	9	10	11
sourceA_Machine2 ProcessingTime	10	10	12	12	14	14	16
sourceB							
sourceB_Machine1 ProcessingTime	5	5	5	5	5	7	7
sourceB_Machine2 ProcessingTime	15	15	15	15	15	10	10
sourceB_Machine3 ProcessingTime	15	15	15	15	15	10	10
destC							
destC_MachineA_1 Processing Time	15	20	23	25	27	15	20
destC_MachineA_2 Processing Time	15	20	23	25	27	15	20
destC_MachineB_1 Processing Time	20	20	20	20	20	10	12
destC_MachineB_2 Processing Time	7	7	7	7	7	10	12
Distributed Simulation Experiment Results							
Model and Results	Experiment1	Experiment2	Experiment3	Experiment4	Experiment5	Experiment6	Experiment7
sourceA							
sourceA_Machine1 Completed	58	57	57		57		
sourceA_Machine2 Completed	48	49	49		47		
sourceA_Total Completed	106	106	106		104		
sourceB							
sourceB_Machine2 Completed	60	60	60		60		
sourceB_Machine3 Completed	8	8	8		8		
sourceB_Total Completed	67	67	67		67		
destC							
destC_Total SOURCEA Processed	78	77	76		74		
destC_Total SOURCEB Processed	36	36	36		36		
destC_Total SOURCEA Returned	14	13	13		14		
destC_Total SOURCEB Returned	7	7	7		7		

Screenshot 28: Excel-based Distributed Production Line-Experimentation Tool (DPL-ET)

In the WinGrid architecture, the DPL-ET is the MA. It communicates with the WJD running on the WinGrid master node through the *DPL-ET adapter*. Through the adapter it passes the experiment parameters to the WJD and receives the simulation results back from it. The integration architecture of WinGrid and DPL is shown in figure 51 below.

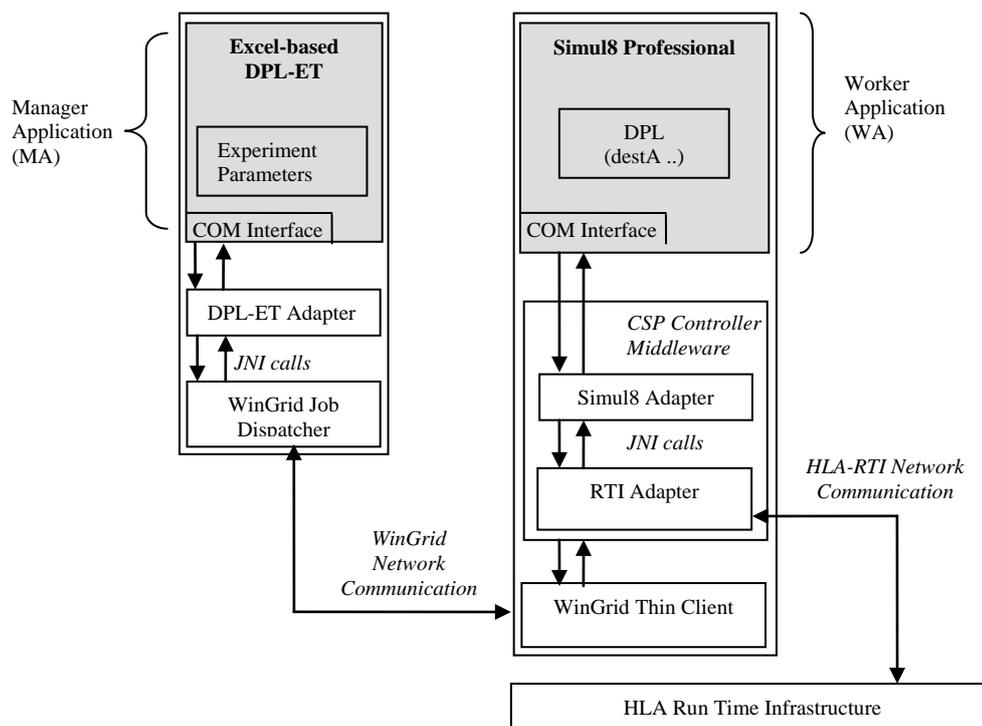


Figure 51: Integration architecture of WinGrid and DPL

The execution of multiple distributed simulation federations using WinGrid is shown in screenshot 29 below. The WinGrid console displays the jobs that have been executed, are running or are in the queue; the WTCs over which jobs had been previously executed or are currently running, etc. It shows that the first job set (Experiment 1), comprising of three different jobs (sourceA, sourceB and destinC) is placed first in the WinGrid queue, followed by the other sets. It also shows that the first job set has been cooperatively executed by three computers (192.168.0.213, 192.168.0.212 and 192.168.0.216) under HLA federation EXP2, and so on. The two experiments that are shown currently running are experiment 6 and experiment 7, under HLA federations EXP2 and EXP2 respectively. The experiment test bed had used a total of six computers, and therefore only two HLA federations were created (each federation has three federates running on individual WTCs).

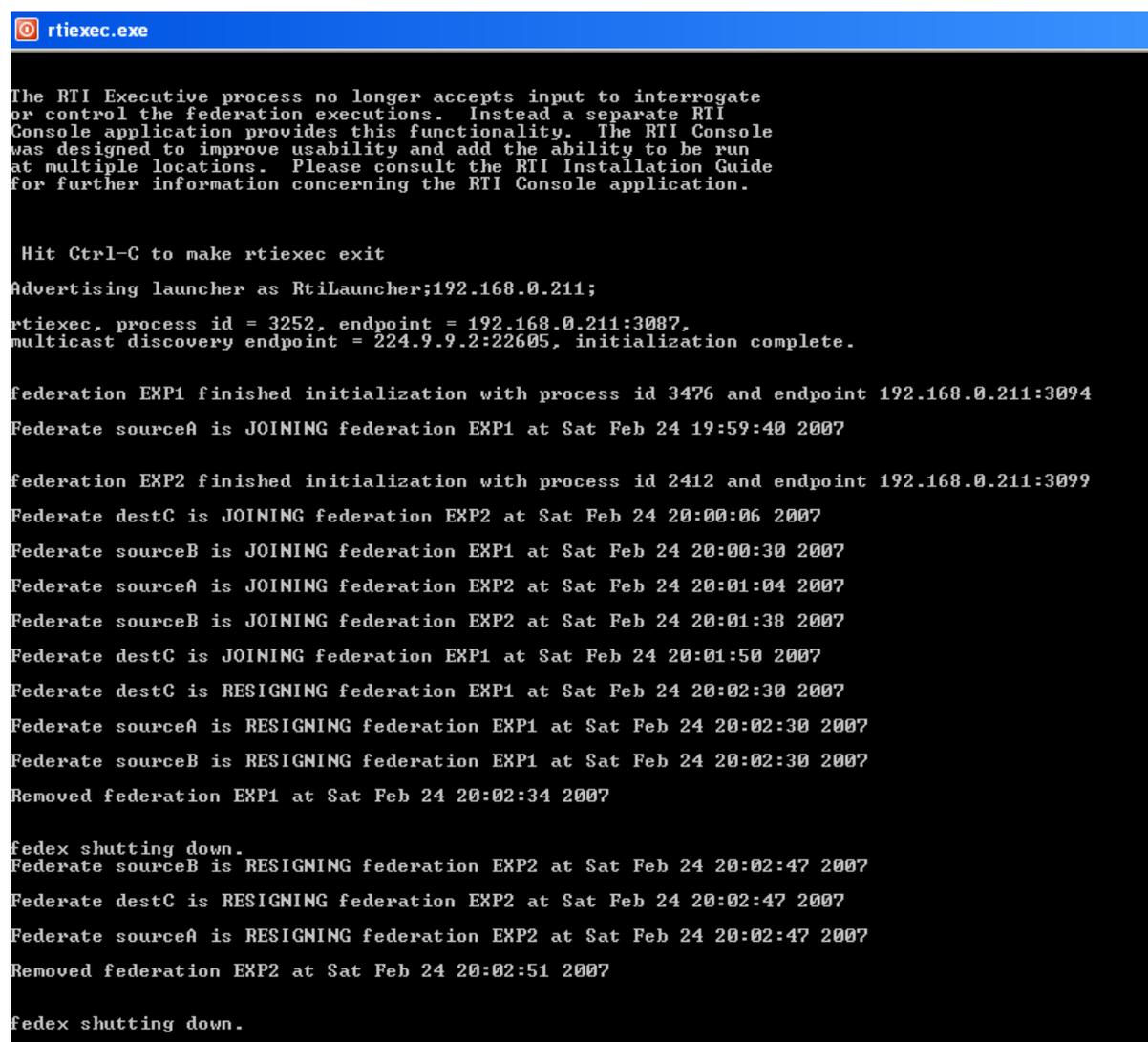
```

27-Feb-2007 01:27:44 wingridjobdispatcher.util.WinGridJobDispatcherLogger logInformationMessage
INFO: EXPERIMENT_NUM      MODEL      IPADDRESS      HLA_FED      STATUS
Experiment1      sourceA    192.168.0.213:5000    EXP2      WorkDne
Experiment1      sourceB    192.168.0.212:5000    EXP2      WorkDne
Experiment1      destinC    192.168.0.216:5000    EXP2      WorkDne
Experiment2      sourceA    192.168.0.211:5000    EXP1      WorkDne
Experiment2      sourceB    192.168.0.213:5000    EXP1      WorkDne
Experiment2      destinC    192.168.0.212:5000    EXP1      WorkDne
Experiment3      sourceA    192.168.0.216:5000    EXP2      WorkDne
Experiment3      sourceB    192.168.0.212:5000    EXP2      WorkDne
Experiment3      destinC    192.168.0.211:5000    EXP2      WorkDne
Experiment4      sourceA    192.168.0.213:5000    EXP1      WorkDne
Experiment4      sourceB    192.168.0.216:5000    EXP1      WorkDne
Experiment4      destinC    192.168.0.212:5000    EXP1      WorkDne
Experiment5      sourceA    192.168.0.211:5000    EXP2      WorkDne
Experiment5      sourceB    192.168.0.210:5001    EXP2      WorkDne
Experiment5      destinC    192.168.0.213:5000    EXP2      WorkDne
Experiment6      sourceA    192.168.0.216:5000    EXP1      Running
Experiment6      sourceB    192.168.0.212:5000    EXP1      Running
Experiment6      destinC    192.168.0.210:5000    EXP1      Running
Experiment7      sourceA    192.168.0.210:5001    EXP2      Running
Experiment7      sourceB    192.168.0.211:5000    EXP2      Running
Experiment7      destinC    192.168.0.213:5000    EXP2      Running
Experiment8      sourceA    -----      -----      InQueue
Experiment8      sourceB    -----      -----      InQueue
Experiment8      destinC    -----      -----      InQueue
Experiment9      sourceA    -----      -----      InQueue
Experiment9      sourceB    -----      -----      InQueue
Experiment9      destinC    -----      -----      InQueue
Experiment10     sourceA    -----      -----      InQueue
Experiment10     sourceB    -----      -----      InQueue
Experiment10     destinC    -----      -----      InQueue
27-Feb-2007 01:27:44 wingridjobdispatcher.util.WinGridJobDispatcherLogger logInformationMessage

```

Screenshot 29: WinGrid console showing execution of distributed simulation federations

Screenshot 30 (next page) shows the HLA-RTI process (*rtiexec.exe*) during the execution of the DPL application over WinGrid. It shows that two different HLA federations, EXP1 and EXP2, were first created (*message: federation EXP1 / EXP2 finished initialization with process id*) and then individual federates joined either of the two federations (*message: Federate sourceA / sourceB / destC is JOINING federation EXP1 / EXP2 at*). After the simulation is completed, the federates resigned (*message: Federate sourceA / sourceB / destC is RESIGNING federation EXP1 / EXP2 at*) and the federation was subsequently destroyed (*message: Removed federation EXP1 / EXP2 at ..*). The intra-federation messages that are routed through the HLA-RTI are not displayed by the *rtiexec* process. Each distributed simulation starts with the creation of a federation and ends with destroying the federation.



```

rtiexec.exe

The RTI Executive process no longer accepts input to interrogate
or control the federation executions. Instead a separate RTI
Console application provides this functionality. The RTI Console
was designed to improve usability and add the ability to be run
at multiple locations. Please consult the RTI Installation Guide
for further information concerning the RTI Console application.

Hit Ctrl-C to make rtiexec exit
Advertising launcher as RtiLauncher;192.168.0.211;
rtiexec, process id = 3252, endpoint = 192.168.0.211:3087,
multicast discovery endpoint = 224.9.9.2:22605, initialization complete.

federation EXP1 finished initialization with process id 3476 and endpoint 192.168.0.211:3094
Federate sourceA is JOINING federation EXP1 at Sat Feb 24 19:59:40 2007

federation EXP2 finished initialization with process id 2412 and endpoint 192.168.0.211:3099
Federate destC is JOINING federation EXP2 at Sat Feb 24 20:00:06 2007
Federate sourceB is JOINING federation EXP1 at Sat Feb 24 20:00:30 2007
Federate sourceA is JOINING federation EXP2 at Sat Feb 24 20:01:04 2007
Federate sourceB is JOINING federation EXP2 at Sat Feb 24 20:01:38 2007
Federate destC is JOINING federation EXP1 at Sat Feb 24 20:01:50 2007
Federate destC is RESIGNING federation EXP1 at Sat Feb 24 20:02:30 2007
Federate sourceA is RESIGNING federation EXP1 at Sat Feb 24 20:02:30 2007
Federate sourceB is RESIGNING federation EXP1 at Sat Feb 24 20:02:30 2007
Removed federation EXP1 at Sat Feb 24 20:02:34 2007

fedex shutting down.
Federate sourceB is RESIGNING federation EXP2 at Sat Feb 24 20:02:47 2007
Federate destC is RESIGNING federation EXP2 at Sat Feb 24 20:02:47 2007
Federate sourceA is RESIGNING federation EXP2 at Sat Feb 24 20:02:47 2007
Removed federation EXP2 at Sat Feb 24 20:02:51 2007

fedex shutting down.

```

Screenshot 30: HLA-RTI executive process executing federations EXP1 and EXP2

The next section evaluates the manufacturing unit case study based on the case study evaluation criteria.

6.3.4 Evaluation of distributed simulation with SMMD task farming service

The evaluation criterion for the case study was that the distributed simulation with SMMD task farming service was practically implementable, and thus the service realizable, through the use of WinGrid middleware. The discussions presented in this section have shown that WinGrid can support this service through the use of HLA-RTI middleware. It can therefore be concluded that the case study evaluation criterion has been met and WinGrid can be used to support the CSP-specific distributed simulation with SMMD task farming service.

The next section evaluates the CSP-GC framework, which is proposed in chapter 3 of this thesis, based on discussions in chapters 3, 4 and 6 and case study experimentation results presented in chapters 5 and 6.

6.4 CSP-GC framework revisited

Chapter three of this thesis has proposed the CSP-GC framework. This framework has identified six potential services that could be provided to CSPs through the use of grid computing technologies. The six services are parallel computing service, task farming service, workflow service, collaboration service, distributed simulation service and web-based simulation service. In this section, the six services are evaluated based on earlier discussions pertaining to the suitability of grid middleware in providing these services and the results of the case studies. The hypothesis presented in this thesis is either supported or rejected based on the evaluation of this framework, since the CSP-GC framework was proposed to provide a logical structure for the evaluation of the hypothesis. The six services are discussed below.

6.4.1 Parallel computing service

Parallel computing service is the first CSP-specific service identified by the framework. This service has the potential of speeding up the execution of one CSP-based simulation using multiple processors. The discussion in section 3.3.1 have shown that a grid middleware that supports parallel computing environments like MPICH, PVM, etc. can potentially offer this service. Of the four grid computing middleware that have been discussed in this thesis, Condor is the only middleware that may support this through its parallel universe execution environment (section 3.4.1). However, only CSPs that have a MPI or PVM-based parallel implementation may possibly be able use this service. The survey of simulation packages have shown that presently none of the CSPs have a parallel MPI/PVM implementation (section 2.5.1), and consequently parallel computing service cannot be utilized by the present generation of CSPs. This service is therefore omitted from the original CSP-GC framework, as this thesis focuses on solutions which are implementable in practice. The modified CSP-GC framework is presented in figure 52.

6.4.2 Task farming service

Task farming service has the potential to speed up the execution of a batch of simulation experiments by running multiple copies of the CSPs, each simulating a separate set of experiments, over different grid nodes. Four case studies have been used to experimentally show that both SMMD and MMMD variants of task farming are possible using grid middleware. In the BOINC case study, the MCS CSP Excel was grid-enabled using PRC middleware BOINC to facilitate SMMD task farming (section 5.4). EDGC middleware Condor was used with two separate MCS CSP Excel-based applications to enable MMMD task farming (section 5.5). DES CSP Witness was used together with WinGrid in the Ford Motors case study to enable SMMD task farming (section 5.6). Finally, in the investment bank case study WinGrid was again used to provide SMMD task farming service to MCS CSP Analytics (section 5.7). The results from all four case studies have shown that the evaluation criteria have been met. It is therefore concluded that task farming service can be used by CSPs through the use of grid middleware. As shown in the modified CSP-GC evaluation framework

(figure 52), interfacing the CSPs with grid middleware can be made possible through use of CSP-grid integration technology.

6.4.3 Workflow service

The grid-facilitated work flow service enables the phased execution of applications that have data dependency among them. The investment bank case study has used Analytics and Excel to implement workflows (section 5.7). The results of the case study experimentation have shown that the case study evaluation criterion with regards to workflows has been met. It can therefore be said that grid computing can support the CSP-specific workflow service. The modified CSP-GC framework identifies the workflow service as one of the CSP-specific services and shows that the CSP-grid integration technology will be required to interface CSPs to grid middleware.

6.4.4 Collaboration service

The two forms of collaboration service that have been identified in this research are, (1) search and download of CSP-based model components, and (2) support for virtual meetings (section 3.3.4). It has been discussed earlier that user-developed web services, which may be hosted by an OGSA-compliant grid middleware, can facilitate the search and download of model components created using MCS and DES CSPs. However, unless such middleware is available for PRC and EDGC forms of grid computing, collaboration service through the use of web services is considered infeasible. Providing virtual meeting support using grid middleware would generally require the integration of audio, video and messaging capability with the grid middleware. None of the middleware that have been examined in this thesis presently has such capabilities. The only grid middleware that is known to have such integrated collaboration support is the *Access Grid Collaboration System* (discussed in section 2.2.2).

Access Grid is for group-to-group collaboration. In this thesis, CSP-based collaboration service in the form of virtual meeting is primarily seen as a one-to-one collaboration between various modellers and problem stake holders using their desktop resources. Such a one-to-one collaboration can be achieved using groupware like Microsoft NetMeeting, which has support for audio, video, messaging, virtual whiteboards and can provide remote access to PCs and applications running on them (Taylor, 2000). It is difficult to argue for a grid middleware like Access Grid that supports group-to-group collaboration and provides computational services only through the use of other grid middleware, when the requirement is primarily for one-to-one collaboration that can be achieved using groupware. Access Grid has therefore not been investigated in this research.

The original CSP-GC framework had identified a collaboration service. The modified CSP-GC framework omits this service as the discussions have shown that collaboration service through the use of web services to enable search and download of models, or through virtual

meetings may not be adequately supported by existing grid middleware. However, in the case of virtual meetings at least, there exist groupware tools that a simulation modeller would possibly find quite effective. The groupware tools are not shown in the modified framework.

6.4.5 Distributed simulation service

Distributed simulation service can be used together with a distributed simulation middleware like HLA-RTI to facilitate a co-ordinated execution of individual CSP-based models over the grid. The NBS case study has shown that distributed model execution is possible over the grid (section 5.8). The results of the case study have also shown that the evaluation criterion has been met. Thus, it can be concluded that distributed simulation service can be supported through the use of grid middleware. The CSP-grid integration technology will, however, be required to interface the CSP to the grid middleware. The original CSP-GC framework is modified to show that distributed simulation service requires use of both the HLA-RTI middleware for distributed simulation and the CSP-grid integration technology.

6.4.6 Web-based simulation service

Web-based simulation service involves access to CSPs using either web services or through grid portals (section 3.3.6). It has been pointed out earlier that a middleware based on the OGSA architecture may be able to host user-developed web services, which can in turn access the open interfaces that are presently made available by many CSPs. However, the middleware for the two forms of grid computing that have been identified as suitable for this research, namely PRC and EDGC, are generally implemented using custom protocols. Furthermore, they do not implement custom web service hosting environments. This may change in the future with the development of PRC/EDGC middleware that is based on OGSA and that implements a sub-set of the services defined by it. Until such time grid-facilitated web-based simulation using web services is considered infeasible.

The second mechanism that can be used to access CSPs, in the context of web-based simulation service, is through the use of grid portals. Grid portals provide a web browser-based front-end that could be used to load simulation experiments for execution over different grid nodes using the CSPs locally installed on them. The grid portal usually interfaces with the grid middleware to submit jobs, monitor job execution and to retrieve the results. WinGrid-WS is one middleware which supports the use of grid portals for running simulation experiments (section 4.5.6). Although an EDGC middleware like Condor does not include a grid portal at present, such a web-based front that interacts with Condor using specific Condor-defined commands could be implemented by the user. Thus, it can be concluded that web-based simulation service in the form of grid portals can be provided using grid middleware.

The original CSP-GC framework (figure 29) shows grid portals as one of the two grid middleware access mechanisms (the other mechanism is through the use of middleware-specific Command Line Interface [CLI] commands). However, use of grid portals is

considered as an optional grid access mechanism because, unlike the core grid mechanisms like resource discovery, job scheduling, job monitoring, etc. that are generally an integral part of most middleware, grid portals usually only provide a web-based front-end to some of the basic grid services (for example, computation service, data service, etc.) for the benefit of the user. Since web-based simulation service also involves the use of grid portals and this service can potentially support other CSP-specific services (in other words, simulation models and experiment parameters for distributed simulation, task farming, etc. can be uploaded through grid portal), the original CSP-GC framework is modified to indicate that web-based simulation service can be provided using the grid portal middleware access mechanism.

Finally, the manufacturing unit case study (section 6.3) has shown that the CSP-specific distributed simulation with task farming service can also be supported through the use of grid middleware. The result of this case study has shown that the evaluation criterion has been met. Thus, the modified CSP-GC framework identifies the distributed simulation with task farming service as a new service. This service was not identified in the original CSP-GC framework. The modified framework also shows that this new service requires the use of both HLA-RTI middleware for distributed simulation and the CSP-grid integration technology. The modified CSP-GC framework is shown in figure 52 below. The service descriptions of the five CSP-specific services presented in the modified CSP-GC framework are shown in table 42.

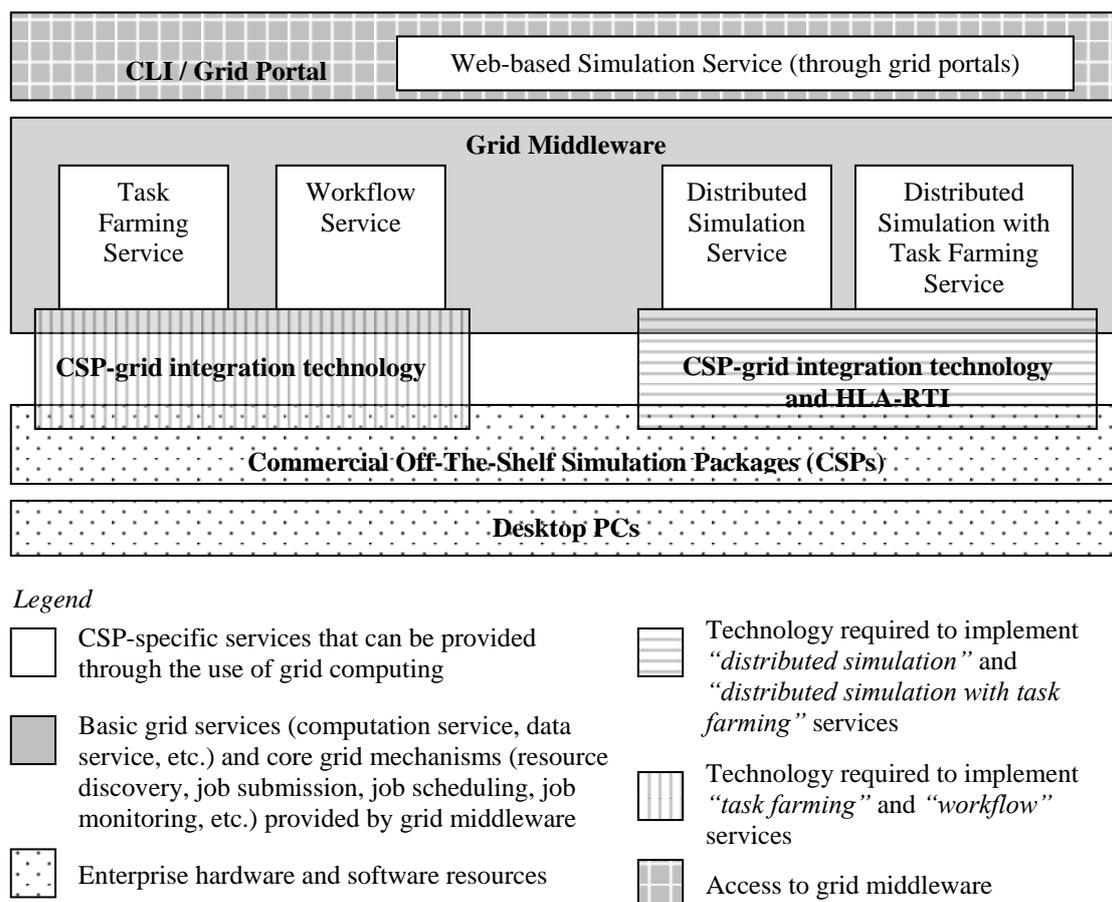


Figure 52: CSP-GC framework (modified)

Table 42: Modified CSP-GC framework defined services and their descriptions

Modified CSP-GC framework defined services	Service description
Task farming service	Task farming service can reduce the time taken to execute batch simulation experiments by distributing the execution of multiple CSP-based DES and MCS experiments over different grid nodes. This service supports concurrent execution of multiple instances of the same simulation model (SMMD task farming) or different simulation models (MMMD task farming). This service requires the use of CSP-grid integration technology.
Workflow service	Workflow service can enable phased execution of different CSP-based DES/MCS models and other external applications based on the underlying data dependencies. This service requires the use of CSP-grid integration technology.
Distributed simulation service	Distributed simulation service can enable execution of DES CSP-based distributed simulations using the HLA-RTI middleware for distributed simulation. This service requires the use of CSP-grid integration technology and HLA-RTI middleware for distributed simulation.
Web-based simulation service	Through the use of grid portals, web-based simulation service can provide simulation users with web-based access to grid middleware. This service can be used by other CSP-specific services to upload simulation models and experiment parameters, to monitor simulation execution, to retrieve the results of the simulation, etc. The web-based simulation service interfaces with grid middleware and not with MCS and DES CSPs (as is the case with the other services).
Distributed simulation with task farming service	Distributed simulation with task farming service can enable execution of multiple instances of DES CSP-based distributed simulations concurrently over the grid. This service requires the use of CSP-grid integration technology and HLA-RTI middleware for distributed simulation.

6.4.7 Evaluation of hypothesis

Section 6.4 has evaluated the original CSP-GC framework that was presented in chapter 3. Based on this evaluation a modified CSP-GC framework has been proposed which shows that four of the six original CSP-specific services can be supported through the use of grid middleware. In addition, one new CSP-specific service (that was not identified in the original framework) can also be supported. Thus, the modified CSP-GC framework identifies five CSP-specific services that can be provided through the use of grid middleware. These services are (1) task farming service (both SMMD and MMMD variants), (2) workflow service, (3) distributed simulation service (relevant only to DES CSPs), (4) web-based simulation service through use of grid portals, and (5) distributed simulation with task farming service (relevant only to DES CSPs).

The hypothesis that “grid computing will benefit CSP-based simulation practice in industry” can therefore be considered true because it has been shown that grid computing can support some of the CSP-specific services with the present generation of grid and CSP technology. However, it is also possible to criticize this conclusion based on the following (for each criticism, an argument is presented in *italics*).

- **The middleware support for CSP services vary. In other words, not all middleware can support all five services. For example, it has been discussed that using BOINC**

to support workflows may be technically difficult to achieve (section 3.4.3); web-based simulation is not supported by BOINC, Condor or WinGrid, etc.

The hypothesis does not state that one particular middleware should be able to address all the requirements of the simulation practitioner. Although it is accepted that one middleware for all five services would be desirable, the evaluation of the middleware has shown that it is not presently possible.

- **Not all the middleware that have been identified as potential candidates for supporting CSP-specific services, through discussions and arguments, have been experimentally evaluated.**

*The reader should note that in the case of **task farming service**, where all the investigated grid middleware were found to be suitable candidates, a total of four case studies have been devoted to experimental evaluation using BOINC, Condor and WinGrid. Thus, the service which was found to be widely supported was experimented more.*

*In the case of **workflow service**, experiments could not be conducted using Condor DAGMan (with Java universe execution environment). This is because the IT department in the investment bank in which the real-world investment bank case study was conducted, and which required the use of workflows, were concerned with network security. As has been discussed earlier, Condor middleware, on which both Condor DAGMan and Condor Java universe execution environment are dependent, uses multiple, bi-directional, static and dynamic ports for communication. The network administrators are usually reluctant to use software that opens up too many non-standard ports for communication.*

*In the case of **distributed simulation service**, although experiments were conducted only using WinGrid, the results can largely be applied to Condor and BOINC because the approach that has been taken in the case of distributed simulation using HLA-RTI is that the user application will implement the logic required with managing the federation. WinGrid, Condor and BOINC would need to only execute these simulations on different nodes – which is possible, as has been experimentally shown using case studies dealing with task farming service. However, as has been discussed in section 6.2, in the case of **distributed simulation with task farming service**, only WinGrid can be used.*

*In the case of **web-based simulation service**, WinGrid-WS was not experimented because it only provides a web-based front end to the underlying grid middleware. Furthermore, this service requires interfacing with grid middleware (which is not the primary research issue being addressed in this thesis) and not with the MCS and DES CSPs.*

- **It has been shown in this research that the level of grid support for CSP-specific services is very much dependent on the actual grid middleware being used to implement a solution. Thus, how far can these results be generalised to apply to different middleware implementations of PRC and EDGC forms of grid computing?**

A grid middleware generally has mechanisms to execute user programs on different grid nodes. The user program is written in a programming language, which when compiled can be executed natively by the operating system (for example, C and C++ code) or can be executed at runtime by an interpreter (for example, Java). In the case of the latter the executing grid node should have the interpreter installed locally. Thus, if programs are written in Java then the grid nodes should have Java Virtual Machine (JVM) installed on them. BOINC, Condor standard universe and Condor vanilla universe support user applications that are written using C/C++. Condor Java universe and WinGrid support execution of Java programs and consequently the grid nodes require JVM to be installed locally.

*It is the responsibility of the user program to invoke the MCS or DES CSPs and perform operations using them. The CSP-grid integration technology that has been presented in this thesis uses an adapter-based approach to communicate between C++/Java code and the CSPs. This adapter is a Visual Basic dynamic link library (VB dll). Thus, the application logic itself is contained within the user C++/Java code and the VB dll. Grid middleware is only responsible for executing the application on different grid nodes. Therefore it is very likely that any PRC or EDGC middleware would provide support for **task farming service** and **distributed simulation service**, wherein the application logic is contained in the user code and the middleware is only responsible for distributing the executing of the user program over different grid nodes. In the case of distributed simulation service, however, the user code will have to interface with HLA-RTI middleware for executing a DES CSP-based distributed simulation on the grid.*

*In the case of **workflow service**, only those middleware that can support the execution of user applications in phases, and transfer data between them (through middleware-defined mechanisms or through the use of shared network drives), will generally be able to provide this service. However, it may also be possible for a user to write a program which invokes operations on different external applications (like CSPs, visualization applications, data analysis software, etc.) in a phased manner, thereby implementing a basic workflow, and then execute it over the grid. The limitation of this approach is that all the external applications that may be used will usually have to be locally installed on all the grid nodes (because the user job which accesses all these applications can be executed on any grid node). However, in the case of grid-facilitated workflow service the different applications may be installed on different grid nodes.*

***Web-based simulation service** through the use of web-portals can be supported by middleware which have a web-based front-end. However, it may be possible for the user*

to create a web application which invokes middleware-defined operations for job submission, monitoring, result retrieval, etc.

Finally, in the case of **distributed simulation with task farming service**, grid middleware that provide mechanisms to schedule jobs taking into consideration the individual distributed simulation models (the distributed simulation models together form the distributed simulation federation) that the jobs represent, should ideally be able to implement this service.

- **The CSP-grid integration technology has been shown to work with only a few CSPs. Can it work with all CSPs?**

The CSP-grid integration technology has been used in six case studies. Apart from the BOINC case study which uses C++ user code to invoke operations on MCS CSP Excel through the VB dll, the rest of the case studies have used code written in Java that call methods defined in the VB dll through JNI. Thus, it has been shown using two different programming languages that integration with VB dll is possible. Interfacing VB dll with CSPs is, however, only possible if the CSPs have well-defined interfaces that can be invoked by external applications. Furthermore, only those operations can be performed on the CSP that have been exposed by the package. Table 10 lists the CSPs that have open interfaces.

6.5 Evaluation of CSPs based on CSP-GC framework defined services

This thesis has, in total, identified seven CSP-specific services. These are parallel computation service, task farming service, workflow service, collaboration service, distributed simulation service, web-based simulation service and distributed simulation with task farming service. Of these seven services, the modified CSP-GC framework only shows five services which can be potentially provided using grid middleware that have been examined in this thesis. Thus, parallel computation service and collaboration service are omitted from the modified CSP-GC framework. Parallel computation service is not considered, although it may be possible for Condor to support this service using parallel universe execution environment, because the MCS and DES CSPs will generally need a MPI/PVM implementation to execute over a set of distributed processors (the CSPs at present do not have such parallel implementations). Similarly, collaboration service is omitted from the modified framework because it is not supported by Condor, BOINC, WinGrid or WinGrid-WS.

Some of the CSPs also have inbuilt support for certain CSP-specific services. However, such support is provided through custom solutions. These solutions only work for the packages for which they are implemented. Table 43 lists the CSPs that support some of the CSP-specific services through custom vendor implementations. The reader is reminded that data pertaining to the CSPs have been collected from the product information published by the vendors of the CSPs on their websites. As such, there may be some error in the CSP-related information

presented below, as product descriptions in the vendor websites may be incomplete, vague or exaggerated. Furthermore, an inadvertent error on the part of the author could be another reason for this error. Table 43 also lists the grid middleware that can support these services. The case studies, wherein grid middleware are experimentally evaluated with regards to CSP-specific services, are also indicated. The description of the columns of the table follows next.

Column one: column [All CSP-specific services] lists all seven CSP-specific services that have been discussed in this thesis, irrespective of whether the modified CSP-GC framework identifies it as a service or not.

Column two: column [MCS / DES CSP] identifies whether a CSP-specific service is being discussed in relation to MCS CSPs, DES CSPs or both.

Column three: column [CSP support on multiple processor machines] lists CSPs that support CSP-specific services over multi-processor machines using custom solutions. Although this research is mainly concerned with running CSP-based services over distributed processors, CSP support on multiple processors is included for reference purposes.

Column four: column [CSP support over distributed processors] lists CSPs that support CSP-specific services over distributed processors using custom solutions.

Column five: column [Grid middleware] lists grid middleware (including specific middleware components like Condor DAGMan, Condor MW, etc.) that have been identified as potential candidates for grid-enabling CSPs with respect to specific services.

Column six: column [Comments] is for general comments. The case studies that have been used in this research to experiment with grid middleware in context to different CSP-specific services are indicated in this column. This column also lists the middleware and specific middleware components (presented in column five) that has been identified for future investigations. Those middleware/middleware components that could not be experimentally evaluated due to unsupported CSP implementations (like Condor parallel university execution environment), etc. have been marked for future investigation.

Table 43: Custom CSP support and grid middleware support for CSP-specific services

All CSP-specific services	MCS / DES CSP	CSP support on multiple processor machines	CSP support over distributed processors	Grid Middleware	Comments
Parallel computation service	MCS CSP	2 (@Risk Industrial and TreeAge Pro - Refer to table 6)	0	(1) Condor parallel universe	<ul style="list-style-type: none"> ▪ MCS and DES CSPs may need to have MPI/PVM-based implementation ▪ (1) is for future investigation
	DES CSP	0	0	(1) Condor parallel universe	

All CSP-specific services	MCS / DES CSP	CSP support on multiple processor machines	CSP support over distributed processors	Grid Middleware	Comments
Task farming service	MCS CSP	0	2 (<i>Vanguard Studio and GoldSim – Refer to table 7</i>)	(1) BOINC (2) Condor Java universe (3) Condor MW (4) WinGrid (5) WinGrid-WS	<ul style="list-style-type: none"> BOINC case study [MCS CSP Excel with (1)] Condor case study [MCS CSP Excel with (2)] Investment bank case study [MCS CSP Analytics with (4)]
	DES CSP	1 (<i>Simul8 – Refer to table 6</i>)	2 (<i>Simprocess and Simul8 – Refer to table 7</i>)	(1) BOINC (2) Condor Java universe (3) Condor MW (4) WinGrid (5) WinGrid-WS	<ul style="list-style-type: none"> Ford case study [DES CSP Witness with (4)] [DES CSP Witness with (5)] Investigated in (Alders, 2006) and (Mustafee et al., 2006a) (3) is for future investigation
Workflow service	MCS and DES CSP	0	0	(1) Condor DAGMan (2) WinGrid	<ul style="list-style-type: none"> Investment bank case study [MCS CSP Analytics and Excel with (2)] (1) is for future investigation
Collaboration service (virtual meetings)	MCS and DES CSP	N/A	0	(1) Access Grid	<ul style="list-style-type: none"> (1) is for future investigation
Distributed simulation service	MCS CSP	N/A	N/A	N/A	<ul style="list-style-type: none"> Distributed simulation is not applicable to MCS CSPs
	DES CSP	0	1 (<i>AutoMod - Refer to table 14</i>)	(1) BOINC with HLA-RTI (2) Condor Java universe with HLA-RTI (3) WinGrid with HLA-RTI	<ul style="list-style-type: none"> NBS case study [DES CSP Simul8 with (3)]
Web-based simulation service	MCS and DES CSP	N/A	8 (<i>QMS, MineSim, Vanguard Studio, AnyLogic, AgenaRisk, Witness, Analytica, Simprocess – Refer to table 15</i>)	(1) WinGrid-WS (grid portal)	<ul style="list-style-type: none"> [DES CSP Witness with (1)] Investigated in (Alders, 2006) and (Mustafee et al., 2006a)
Distributed simulation with task farming service	MCS CSP	N/A	N/A	N/A	<ul style="list-style-type: none"> Distributed simulation is not applicable to MCS CSPs
	DES CSP	0	0	(1) WinGrid with HLA-RTI	<ul style="list-style-type: none"> Manufacturing works case study [DES CSP Simul8 with (1)]

6.6 Chapter summary

This chapter has evaluated the CSP GC framework. It has identified a new CSP-specific service called distributed simulation with task farming service (section 6.2). This service is a combination of task farming service and distributed simulation service, both of which had been identified in the original CSP-GC framework (chapter 3). Section 6.2 then investigates BOINC, Condor and WinGrid middleware in relation to this new service. WinGrid support for distributed simulation with task farming service is examined through case study experimentation in section 6.3. This is followed by an evaluation of the original CSP-GC framework based on the results of the case study experimentations and the discussions presented in the earlier chapters of this thesis (section 6.4). This evaluation has shown that only four of the original six CSP-specific services can be potentially supported through the use of existing grid technology. Based on the evaluation of the original CSP-GC framework, a modified CSP-GC framework is then presented. The modified framework includes the four previously identified and “realizable” services and the new distributed simulation with task farming service. The technology that is required to provide these services are also identified in the modified framework. The evaluation of the framework has shown that the hypothesis presented in the thesis is acceptable because some of the CSP-specific services identified in the original CSP-GC evaluation framework (and all services in the modified framework) can be provided through use of grid middleware. Finally, CSPs that support some of the CSP-specific services through custom solutions are listed in section 6.5.

The next chapter summarizes the research that has been presented in this thesis. It revisits the aim and the objectives that were outlined in chapter one, discusses the contribution of this work and future research that can be conducted on the basis of this work.

7 SUMMARY AND CONCLUSION

7.1 Introduction

Chapter six has evaluated the original CSP-GC framework based on middleware-specific discussions presented earlier in this thesis and case study experimentation results. Based on this evaluation, the original CSP-GC framework is modified to show only those services that can be potentially supported with existing grid middleware and unmodified CSP packages. The evaluation of this framework has shown that the hypothesis presented in this work, namely “grid computing will benefit CSP-based simulation practice in industry”, is valid.

Chapter seven is the last chapter of this thesis. Section 7.2 summarizes the research that has been presented in this thesis. Section 7.3 then revisits the aim and objectives that were outlined in chapter one. The purpose of this is to show how the different objectives were met in the various chapters. The contribution of this research is discussed next in section 7.4. Section 7.5 is the final section of this thesis and it discusses future research in the area of grid computing and CSP-based simulation modelling.

7.2 Research summary

This research has been motivated by the advances being made in the field of grid computing and the realization that simulation in industry could potentially benefit through the use of grid computing technologies. This research recognises that end-user adoption of grids could be facilitated by focusing on software tools that are commonly used by employees at their workplace. In the context of simulation in industry, the end-users are the simulation practitioners and the tools that are generally used to model simulations are the Commercial Off-The-Shelf (COTS) Simulation Packages (CSPs). Thus, this research investigates how grid computing can further the field of CSP-based simulation practice and, thereby, offer some benefits to simulation end-users.

Empirical research is conducted in this study and it followed four distinct stages, namely, it proposed a hypothesis, identified methods to progressively evaluate the hypothesis, compiled the results obtained by applying the identified methods, and finally, evaluated the hypothesis based on these results and discussions. The research has led to the development of a grid middleware called WinGrid, and certain aspects of design research have been used during the development of this artefact.

This research has proposed the hypothesis that grid computing will benefit CSP-based simulation practice in industry. In order to evaluate this hypothesis, a literature review was first conducted to investigate how grid computing technologies could potentially support CSP-based simulations in industry. To this end, six higher level grid services were identified along with two forms of grid computing, namely, Public Resource Computing (PRC) in an enterprise

context, and Enterprise Desktop Grid Computing (EDGC). Furthermore, two specific grid computing middleware were chosen as representative middleware for either PRC or EDGC forms of grid computing. This was done in order to enable further investigation of the two different forms of grid computing in relation to CSP-based simulation in industry. BOINC was selected as a representative middleware for the PRC form of grid computing because it is presently the most popular PRC middleware, it is available free of charge, and finally because it allows users to create their own BOINC-based projects. Condor was selected as a representative middleware for the EDGC form of grid computing owing to its large deployment base and its free availability.

The COTS Simulation Package-Grid Computing (CSP-GC) framework was proposed to provide a logical structure for the evaluation of the hypothesis. This framework identified six grid-facilitated CSP-specific services. These services were in turn based on the higher level grid services that were identified previously from the literature review. The six services were parallel computing service, task farming service, workflow service, collaboration service distributed simulation service and web-based simulation service. BOINC and Condor were then evaluated in relation to the CSP-specific services.

The evaluation of BOINC and Condor has shown that some of the CSP-GC framework defined services could be potentially provided by these middleware. For example, both middleware may be able to offer task farming service and distributed simulation service. However, in the case of the latter, a distributed simulation middleware (HLA-RTI) would be required. It has been argued that Condor may also be able to potentially provide parallel simulation service and workflow service through the use of the Condor parallel universe execution environment and Condor DAGMan respectively. However, the examination of the middleware has also indicated that web-based simulation service (through the use of grid portals and web services) and collaboration service (through enabling search and download of CSP models, and integrated support for virtual meetings) were not currently supported by either of the two middleware.

The research then expressed the need for an “ideal” grid middleware that was specifically implemented to support CSP-based simulation in industry. It was argued that the ideal middleware would be the one which is supported on Windows, which uses only one communication channel, implements the “push” job scheduling mechanism, supports task-parallel task farming applications and would support Java-based user applications. The EDGC middleware that was implemented based on these “ideal” middleware requirements was called WinGrid.

This research then presented a discussion on WinGrid and the web services extension of WinGrid called WinGrid-WS. WinGrid was evaluated in respect to the six CSP-GC framework

identified services. The discussion on WinGrid-WS was limited to task farming service and web-based simulation service since this middleware was explicitly implemented to support these two services. It has been shown that WinGrid can potentially support task farming service, workflow service and distributed simulation service. WinGrid-WS, on the other hand, supports task farming service and web-based simulation service. The latter is supported through the use of grid portals.

To experimentally evaluate three CSP specific services, namely, task farming service, workflow service and distributed simulation service, five hypothetical and real-world case studies were conducted in this research. BOINC, Condor and WinGrid have been used in three different case studies to implement CSP-based task farming service. The BOINC case study used MCS CSP Excel; the Condor case study used two different MCS CSP Excel-based applications to implement the MMMD form of task farming; and the real-world Ford case study integrated WinGrid and a proprietary DES CSP Witness-based application called FIRST; The workflow service was evaluated using WinGrid and an MCS CSP Analytics-based application in the context of the real-world investment bank case study. The NBS case study has used WinGrid and DES CSP Simul8 Professional to evaluate the distributed simulation service.

The results of these case studies showed that some of the CSP-specific services can be provided through the use of grid middleware. The hypothesis presented in this thesis was therefore validated as it was shown that simulation practitioners can potentially derive some benefit from using these grid-facilitated CSP-specific services. The evaluation of the CSP-GC framework has also identified a new service – distributed simulation with task farming service. The original framework was finally modified to represent only those services that can be provided using existing PRC and EDGC grid computing middleware. These services are task farming service, workflow service, web-based simulation service through the use of grid portals, distributed simulation service and distributed simulation with task farming service. The modified CSP-GC framework also shows the technology (CSP-grid integration technology and the HLA-RTI) that would be required to implement these services.

Summing up, this research has proposed the CSP-GC framework that has outlined five CSP-specific services and has recognised the form of grid computing and specific grid middleware which could be used to provide some of these services for the benefit of CSP-based simulation practice in industry.

7.3 Aims and objectives revisited

The aim of this thesis was to investigate how *grid computing technologies might benefit CSP-based simulation practice in industry*. Towards the realization of this aim, four objectives were identified. Figure 53 shows the chapters in which the different objectives have been met.

<p>Chapter 1: Introduction</p> <p>Objective 1: Stated the hypothesis.</p>
<p>Chapter 2: Grid computing and simulation packages</p> <p>Objective 1: Reviewed the subject area of grid computing and identified what grid computing has to offer.</p> <p>Objective 2: Identified existing grid computing middleware.</p>
<p>Chapter 3: Proposing the CSP-GC framework</p> <p>Objective 2: Proposed the CSP-GC framework.</p> <p>Objective 2: Examined how BOINC and Condor can potentially support the CSP-GC framework.</p>
<p>Chapter 4: Development of desktop grids for Windows</p> <p>Objective 2: Examined how WinGrid and WinGrid-WS middleware can potentially support the CSP-GC framework.</p>
<p>Chapter 5: Case studies</p> <p>Objective 3: Experimentally tested the CSP-GC framework with BOINC, Condor and WinGrid.</p>
<p>Chapter 6: Revisiting the CSP-GC framework</p> <p>Objective 3: Experimentally tested the new CSP-GC service (distributed simulation with task farming) with WinGrid.</p> <p>Objective 4: Evaluated the CSP-GC framework and tested the hypothesis</p>
<p>Chapter 7: Summary and conclusion</p>

Figure 53: Chapters that meet the different objectives outlined in this thesis

- **Objective 1:** *State the hypothesis and identify what grid computing has to offer*

Chapter one of this thesis presented the hypothesis that “grid computing will benefit CSP-based simulation practice in industry”. A literature review of grid computing in chapter two identified higher level grid services that could potentially support the DES and the MCS CSPs.
- **Objective 2:** *Propose the CSP-GC framework and identify grid computing middleware that can potentially support the framework*

Chapter two identified existing grid computing middleware, namely PRC middleware BOINC and EDGC middleware Condor, which could potentially be used together with the CSPs. Chapter three proposed the CSP-GC framework and evaluated BOINC and Condor in relation to the CSP-specific services that were outlined in the CSP-GC framework. Similarly, chapter four examined WinGrid and WinGrid-WS in relation to the CSP-specific services. It was identified that these middleware could support some of the CSP-GC framework defined services.

- **Objective 3: Experimentally test the CSP-GC framework**
Using a total of six hypothetical and real-world case studies, chapters five and six presented experimental evaluation of some of the CSP-GC framework defined services using grid middleware and unmodified MCS and DES CSPs.
- **Objective 4: Evaluate CSP-GC framework and test the hypothesis**
Chapter six evaluated the CSP-GC framework based on the results of the case study experimentation and the grid-specific discussions presented in this thesis.

7.4 Contribution of this research

This research is arguably the first attempt to undertake a study of CSPs in the context of grid computing. The contribution of this research is the modified CSP-GC framework, presented in chapter six, which identifies five grid-facilitated CSP-specific services that can be potentially provided through the use of grid technologies. This framework further shows that the CSP-grid integration technology and the HLA-RTI distributed simulation middleware will have to be used to implement some of the CSP-specific services. The CSP-grid integration technology can be potentially used with any CSP that exposes package functionality and any grid middleware that supports the execution of Java programs. A HLA-RTI is only required to run a CSP-based distributed simulation over the grid.

A further contribution is the recognition of the form of grid computing, namely Public-Resource Computing (PRC) in an enterprise context and Enterprise Desktop Grid Computing (EDGC), which can be used to grid-enable existing CSPs. This research has shown that cluster-based grid computing is generally unsuitable for integration with Windows-based end-user applications like the CSPs. Using PRC and EDGC forms of grid computing for CSP-based simulation in industry can not only facilitate the execution of distributed models, speed up simulation experimentation, etc., but it can also maximize the utilization of hardware resources (PCs and network infrastructure) and software resources (CSPs) within an organization. The latter is achieved through making use of under utilized desktop computers and the software installed on them.

Yet another contribution is the identification of specific grid computing middleware, namely BOINC, Condor, WinGrid and WinGrid-WS, which can be used to interface with CSPs to provide some of the CSP-specific services identified by the modified CSP-GC framework. Of the four middleware that have been examined in this thesis, BOINC and Condor may be more suitable for use by simulation users, since they are available for download free of charge, include installation manuals and user guides, and are supported by user forums and training programs (for example, *Condor Week* is an annual training program conducted by the University of Wisconsin, Madison). WinGrid and WinGrid-WS middleware, on the other hand, are primarily research software and the intervention of the system developer will generally be required to implement new CSP-based solutions.

7.5 Further research

This research has investigated how simulation users could potentially benefit from the use of grid technologies at their workplace. The focus of this thesis was on end-users who were considered experts in modelling and simulation but were not expected to be IT specialists. However, the CSP-grid integration technology that has been proposed in this work requires some knowledge of Java and Visual Basic programming. Furthermore, the end-users will also need to know the middleware-specific mechanisms to create jobs, submit jobs, retrieve results, etc. Some of this knowledge could be acquired through self-study and imparted through training. However, for the wider adoption of grid technology for CSP-based simulation, it may be necessary to develop higher-level tools that would hide the complexity of the CSP-grid integration technology and middleware specific mechanisms, and provide end-users with easy to use graphical interfaces through which they could possibly integrate CSPs with grid middleware.

Two CSP-specific services identified by the modified CSP-GC framework relate to distributed simulation. Although it has been shown that grid computing could facilitate the execution of distributed models (through the use of HLA-RTI distributed simulation middleware), implementing a distributed simulation federation is not a trivial task using CSPs that do not have inbuilt support for it. More research is needed in the area of CSP-based distributed simulation, so that in future it will ideally be possible for end-users to implement distributed models using the CSPs themselves and then to execute the models over the grid.

Condor MW, Condor DAGMan and Condor parallel universe are specific Condor components which have been identified to potentially support CSP-specific task farming service, work flow service and parallel computation service respectively. Condor MW and Condor DAGMan could not be evaluated in context to task farming service and workflow service because the investment bank case study (section 5.7) has used WinGrid. Condor parallel universe execution environment could not be experimentally tested to examine the support for parallel computation service because the existing MCS and DES CSPs do not presently have MPI/PVM implementations. These are all future areas of research.

Future research could also involve extending WinGrid to support web services. This would allow the evaluation of collaboration service through facilitating search and download of CSP model components (section 3.3.4) and evaluation of web-based simulation service through use of web services (section 3.3.6). Future research in WinGrid could also involve implementing a WinGrid workflow component on top of the WinGrid Job Dispatcher (WJD). The application workflow logic can then be input into the WinGrid Workflow component and which will thereafter be responsible for submitting jobs to WJD based on the underlying workflow logic.

LIST OF REFERENCES

Alder, D. (2004). The Jacob project: a Java–COM bridge, Version 1.8, 1999-2004. Website <http://danadler.com/jacob/>. Last accessed 15th February 2007.

Alfieri, R., Cecchini, R., Ciaschini, V., Dell’Agnello, L., Frohner, A., Lo’rentey, K. and Spataro, F. (2005). From gridmap-file to VOMS: managing authorization in a grid environment. *Future Generation Computer Systems*, 21(4): 549–558.

Almond, J. and Snelling, D. (1999). UNICORE: uniform access to supercomputing as an element of electronic commerce. *Future Generation Computer Systems*, 15(5-6): 539-548.

Alstad, A. (2006). Grid system for performance transparency with COTS simulation packages. Masters thesis. School of Information Systems, Computing and Mathematics Brunel University, UK. Available online http://people.brunel.ac.uk/~cspggnm/MSc_anders.doc. Last accessed 3rd April 2007.

Anderson, D. P. (2004). BOINC: a system for public-resource computing and storage. In *Proceedings of the 5th International Workshop on Grid Computing*, pp.4-10. IEEE Computer Society, Washington, DC, USA.

Anderson, D. P., Cobb, J., Korpela, E., Lebofsky, M. and Werthimer, D. (2002). SETI@home: an experiment in public-resource computing. *Communications of the ACM*, 45(11): 56-61.

Anderson, D. P., Christensen, C. and Allen, B. (2006). Designing a runtime system for volunteer computing. In *Proceedings of the 2006 International Conference on High Performance Computing, Networking, Storage, and Analysis (Supercomputing, 2006)*. Article No. 126. ACM Press, New York, NY, USA.

Antonioletti, M., Atkinson, M., Baxter, R., Borley, A., Chue Hong, N. P., Collins, B., Hardman, N., Hume, A. C., Knox, A., Jackson, M., Krause, A., Laws, S., Magowan, J., Paton, N. W., Pearson, D., Sugden, T., Watson, P. and Westhead, M. (2005). The design and implementation of grid database services in OGSA-DAI. *Concurrency and Computation: Practice and Experience*, 17(2–4): 357–376.

Argonne National Laboratory. (2006). MPICH2 user’s guide, Version 1.0.5. Available online <http://www-unix.mcs.anl.gov/mpi/mpich/downloads/mpich2-doc-user.pdf>. Last accessed 20th March 2007.

Argonne National Laboratory. (2007). The message passing interface (MPI) standard. Website <http://www-unix.mcs.anl.gov/mpi>. Last accessed 9th February 2007.

Atkinson, M., DeRoure, D., Dunlop, A., Fox, G., Henderson, P., Hey, T., Paton, N., Newhouse, S., Parastatidis, S., Trefethen, A., Watson, P. and Webber, J. (2005). Web service grids: an evolutionary approach. *Concurrency and Computation: Practice and Experience*, 17(2-4): 377-389.

Aubanel, E. (2000). Introduction to MPI. Advanced computational research laboratory, University of New Brunswick, Canada. Available online [http://acr1.cs.unb.ca/php/training/mpi/aubanel-intro_to_mpi.ppt#256,1,Introduction to MPI](http://acr1.cs.unb.ca/php/training/mpi/aubanel-intro_to_mpi.ppt#256,1,Introduction%20to%20MPI). Last accessed 2nd March 2007.

Ayani, R. and Rajaei, H. (1992). Parallel simulation using conservative time windows. In *Proceedings of the 24th Winter Simulation Conference*, Swain J. J., Goldsman, D., Crain, R. C. and Wilson, J. R. (eds.), pp. 709-717. ACM Press, New York, NY, USA.

Baker, M., Buyya, R. and Laforenza, D. (2002). Grids and grid technologies for wide-area distributed computing. *Software - Practice and Experience*, 32(15): 1437-1466.

Barbera, R., Falzone, A. and Rodolico, A. (2003). The GENIUS Grid Portal. In *Proceedings of Computing in High Energy and Nuclear Physics*. Available online <http://web.oapd.inaf.it/wp10/portals/TUCT001.pdf>. Last accessed 29th April 2007.

Barney, B. (2006). Introduction to parallel computing. Available online http://www.llnl.gov/computing/tutorials/parallel_comp/. Last accessed 2nd March 2007

Basel Committee on Banking Supervision (1999). Principles for the management of credit risk. Available online <http://www.bis.org/publ/bcbs54.pdf>. Last accessed 23rd February 2007.

Basney, J. and Livny, M. (1999). Deploying a high throughput computing cluster. In Buyya, R. (ed.), *High Performance Cluster Computing, Volume 1* (chapter 5). NJ, USA: Prentice Hall PTR.

Basney, J., Raman, R. and Livny, M. (1999). High throughput monte carlo. In *Proceedings of the 9th SIAM Conference on Parallel Processing for Scientific Computing*. Society for Industrial and Applied Mathematics (SIAM), Philadelphia, PA, USA. Available online www.cs.wisc.edu/condor/doc/htmc-siam9.ps. Last accessed 4th April 2007.

Bayucan, A., Henderson, R. L., Lesiak, C., Mann, B., Proett, T. and Tweten, D. (1999). Portable batch system: external reference specification. MRJ technology solutions. Available online http://www.jlab.org/hpc/PBS/v2_2_ers.pdf. Last accessed 16th March 2007.

- Beckles, B., Se-Chang, S. and Kewley, J. (2005). Current methods for negotiating firewalls for the Condor system. In *Proceedings of the 4th UK e-Science All Hands Meeting*. Available online <http://www.cs.wisc.edu/condor/doc/CondorandFirewalls.pdf>. Last accessed 10th May 2007.
- Bell, D., Mustafee, N., Taylor, S. J. E., de Cesare, S. and Lycett, M. (2006). A web services component discovery and deployment architecture for simulation model reuse. In *Proceedings of the 2006 European Simulation Interoperability Workshop (EURO SIW)*. 06E-SIW-047. Simulation Interoperability Standards Organization, Orlando, Florida, USA.
- Beowulf.org. (2007). What makes a cluster a Beowulf? Website <http://www.beowulf.org/overview/index.html>. Last accessed 9th February 2007.
- Berlich, R., Kunze, M. and Schwarz, K. (2005). Grid computing in Europe: from research to deployment. In *Proceedings of the 2005 Australasian Workshop on Grid Computing and e-Research*, pp. 21-27. Australian Computer Society, Darlinghurst, Australia.
- Berman, F., Wolski, R., Casanova, H., Cirne, W., Dail, H., Faerman, M., Figueira, S., Hayes, J., Obertelli, G., Schopf, J., Shao, G., Smallen, S., Spring, N., Su, A. and Zagorodnov, D. (2003). Adaptive computing on the grid using AppLeS. *IEEE transactions on parallel and distributed systems*, 14(4): 369-382.
- Bernholdt, D., Bharathi, S., Brown, D., Chancio, K., Chen, M., Chervenak, A., Cinquini, L., Drach, B., Foster, I., Fox, P., Garcia, J., Kesselman, C., Markel, R., Middleton, D., Nefedova V., Pouchard, L., Shoshani, A., Sim, A., Strand, G. and Williams, D. (2005). The earth system grid: supporting the next generation of climate modelling research. *Proceedings of the IEEE*, 93(3): 485-495.
- Berry, D., Usmani, A., Torero, J., Tate, A., McLaughlin, S., Potter, S., Trew, A., Baxter, R., Bull, M. and Atkinson, M. (2005). FireGrid: integrated emergency response and fire safety engineering for the future built environment. In *Proceedings of the 2005 UK e-Science All Hands Meeting*, pp. 1034–1041. Available online <http://www.allhands.org.uk/2005/proceedings/papers/384.pdf>. Last accessed 29th April 2007.
- Bhaskar, R., Lee, H. S., Levas, A., Pétrakian, R., Tsai, F. and Tulsie, B. (1994). Analyzing and re-engineering business processes using simulation. In *Proceedings of the 26th Winter Simulation Conference*, Tew, J. D., Manivannan, S., Sadowski, D. A. and Seila, A. F. (eds.), pp. 1206-1213. Society for Computer Simulation International, San Diego, CA, USA.

Boer, C. A. (2005). Distributed simulation in industry. PhD thesis. Erasmus Research Institute of Management (ERIM), Erasmus University Rotterdam, The Netherlands. Available online <https://ep.eur.nl/handle/1765/6925>. Last accessed 4th April 2007.

Boer, C. A., Saanen, Y., Veeke, H. and Verbraeck, A. (2002). Simulation backbone Famas.MV2. Project 0.2 technical design (final report). Onderzoekschool voor Transport, Infrastructuur en Logistiek (TRAIL), Technical University of Delft, Delft, The Netherlands.

BOINC. (2007a). Desktop grid computing with BOINC. Website <http://boinc.berkeley.edu/dg.php>. Last accessed 17th February 2007.

BOINC. (2007b). Overview of BOINC. Website <http://boinc.berkeley.edu/intro.php>. Last accessed 17th February 2007.

BOINC (2007c). The BOINC application programming interface (API). Website <http://boinc.berkeley.edu/trac/wiki/BasicApi>. Last accessed 10th May 2007.

BOINC (2007d). BOINC network communication overview. Website <http://boinc.berkeley.edu/comm.php>. Last accessed 10th May 2007.

Borshchev, A., Karpov, Y. and Kharitonov, V. (2002) Distributed simulation of hybrid systems with AnyLogic and HLA. *Future Generation Computer Systems*, 18(6): 829–839.

Bortscheller, B. J. and Saulnier, E. T. (1992). Model reusability in a graphical simulation package. In *Proceedings of the 24th Conference on Winter Simulation*, Swain, J. J., Goldsman, D., Crain, R. C. and Wilson, J. R. (eds.), pp. 764-772. ACM Press, New York, NY, USA.

Brodlie, K., Wood, J., Duce, D. and Sagar, M. (2004). gViz: visualization and computational steering on the grid. In *Proceedings of the UK e-Science All Hands Meeting 2004*, pp. 54-60. Available online http://www.comp.leeds.ac.uk/vvr/gViz/publications/AHM04_wshop_paper.pdf. Last accessed 28th April 2007.

Brooke, J. M., Coveney, P. V., Harting, J., Jha, S., Pickles, S. M., Pinning, R. L. and Porter, A. R. (2003). Computational steering in RealityGrid. In *Proceedings of the UK e-Science All Hands Meeting 2003*, pp. 885-888. Available online <http://www.nesc.ac.uk/events/ahm2003/AHMCD/pdf/179.pdf>. Last accessed 28th April 2007.

Brooks, R., Robinson, S. and Lewis, C. (2001). Simulation and inventory control. *Operational Research Series*. Hampshire, UK: Palgrave.

Bryant, R. E. (1977). Simulation of packet communication architecture computer systems, MIT/LCS/TR-188, Massachusetts Institute of Technology, Cambridge, Massachusetts. Available online <http://www.lcs.mit.edu/publications/pubs/pdf/MIT-LCS-TR-188.pdf>. Last accessed 4th April 2007.

Burke, S., Campana, S., Peris, A. D., Donno, F., Lorenzo, P. M., Santinelli, R. and Sciaba, A. (2007). gLite 3 user guide, manuals series. Document identifier CERN-LCG-GDEIS-722398. Available online <https://edms.cern.ch/file/722398/gLite-3-UserGuide.pdf>. Last accessed 15th March 2007.

Buyya, R., Abramson, D. and Giddy, J. (2000). Nimrod/G: an architecture for a resource management and scheduling system in a global computational grid. In *Proceedings of the 4th international conference on High Performance Computing in the Asia-Pacific Region*, pp. 283-289. IEEE Computer Society, Washington, DC, USA.

Calder, B., Chien, A., Wang, J. and Yang, D. (2005). The entropia virtual machine for desktop grids. In *Proceedings of the 1st ACM/USENIX international conference on Virtual execution environments*, pp.186-196. ACM Press, New York, NY, USA.

Canabarro, E. and Duffie, D. (2003). Measuring and marking counterparty risk. In Tilman, L.M. (ed.), *Asset/Liability Management of Financial Institutions* (chapter 9). London, UK: Euromoney books. Available online http://www.stanford.edu/~duffie/Chapter_09.pdf. Last accessed 26th March 2007.

Casanova, H. (2002). Distributed computing research issues in grid computing. *ACM SIGACT News*, 33(3): 50-70. ACM Press, New York, NY, USA.

Chance, D. M. (2004). Monte carlo simulation, teaching note 96-03. Available online <http://www.bus.lsu.edu/academics/finance/faculty/dchance/Instructional/TN96-03.pdf>. Last accessed 8th March 2007.

Chandra, A., Trivedi, R. and Weissman, J. (2005). Hosting services on the grid: challenges and opportunities. University of Minnesota - Computer Science and Engineering Technical Report. Report number 05-026. Available online http://www.cs.umn.edu/research/technical_reports.php?page=report&report_id=05-026. Last accessed 7th May, 2007.

Chandy, K. M. and Misra, J. (1979). Distributed simulation: a case study in design and verification of distributed programs. *IEEE Transactions on Software Engineering*, 5(5): 440-452.

Chandy, K. M. and Misra, J. (1981). Asynchronous distributed simulation via a sequence of parallel computations. *Communications of the ACM*, 24(11): 198 - 206.

Cheriton, D. R. (1988). The V distributed system. *Communications of the ACM*, 31(3): 314-333.

Chetty, M. and Buyya, R. (2002). Weaving computational grids: how analogous are they with electrical grids? *Computing in Science and Engineering*, 4(4): 61-71.

Chien, A., Calder, B., Elbert, S. and Bhatia, K. (2003). Entropia: architecture and performance of an enterprise desktop grid system. *Journal of Parallel and Distributed Computing*, 63(5): 597-610.

Choi, S., Baik, M., Hwang, C., Gil, J. and Yu, H. (2004). Volunteer availability based fault tolerant scheduling mechanism in desktop grid computing environment. In *Proceedings of the 3rd IEEE International Symposium on Network Computing and Applications*, pp. 366-371. IEEE Computer Society, Washington, DC, USA.

Christensen, C., Aina, T. and Stainforth, D. (2005). The challenge of volunteer computing with lengthy climate model simulations. In *Proceedings of the First International Conference on e-Science and Grid Computing (E-SCIENCE '05)*, pp.8-15. IEEE Computer Society, Washington, DC, USA.

Condor. (2007). Condor high throughput computing: top five myths about Condor. Available online <http://www.cs.wisc.edu/Condor/myths.html>. Last accessed 16th February 2007.

Condor DAGMan. (2007). Condor DAGMan. Website <http://www.cs.wisc.edu/condor/dagman>. Last accessed 27th February 2007.

Condor MW. (2005). User's guide to MW: a guide to using MW. Available online <http://www.cs.wisc.edu/condor/mw/usersguide.pdf>. Last accessed 19th February 2007.

Condor MW. (2007). What is Condor MW? Website <http://www.cs.wisc.edu/condor/mw/index.html>. Last accessed 19th February 2007.

Condor Version 6.9.1 Manual. (2007a). Platform-specific information on Microsoft Windows, Condor 6.9.2 manual. Website http://www.cs.wisc.edu/condor/manual/v6.9/6_2Microsoft_Windows.html. Last accessed 27th February 2007.

Condor Version 6.9.1 Manual. (2007b). Road-map for running jobs, Condor 6.9.2 manual. Website http://www.cs.wisc.edu/condor/manual/v6.9/2_4Road_map_Running.html. Last accessed 27th February 2007.

Condor Version 6.9.1 Manual. (2007c). Command reference manual: condor_prio, Condor 6.9.2 manual. Website http://www.cs.wisc.edu/condor/manual/v6.9/condor_prio.html. Last accessed 4th March 2007.

Contingency Analysis. (2003). Credit risk glossary. Website http://www.riskglossary.com/link/credit_risk.htm. Last accessed 22nd February 2007.

Corbato, F. J. and Vyssotsky, V. A. (1965). Introduction and overview of the Multics system. In *Proceedings of the AFIPS Fall Joint Computer Conference*, pp. 185–196. IEEE Educational Activities Department, Piscataway, NJ, USA. Available online <http://www.multicians.org/fjcc1.html>. Last accessed 17th March 2007.

Credent Analytics. (2007). Credit risk management system - Credent Analytics 2.3 user guide. SunGard Corporation [<http://www3.sungard.com/financial/>].

Cygwin. (2007). What is cygwin? Website <http://cygwin.com/>. Last accessed 3rd March 2007.

Czajkowski, K., Fitzgerald, S., Foster, I. and Kesselman, C. (2001). Grid information services for distributed resource sharing. In *Proceedings of the 10th IEEE International Symposium on High Performance Distributed Computing*, pp. 181-194. IEEE Computer Society, Washington, DC, USA.

Dahmann, J. S., Fujimoto, R. M. and Weatherly, R. M. (1997). The department of defense high level architecture. In *Proceedings of the 29th Winter Simulation Conference*, Andradtir, S., Healy, K. J., Withers, D. H. and Nelson, B. L. (eds.), pp 142 – 149. ACM Press, New York, NY, USA.

Deelman, E., Blythe, J., Gil, Y., Kesselman, C., Mehta, G., Patil, S., Su, M., Vahi, K. and Livny, M. (2004). Pegasus: mapping scientific workflows onto the grid. In *Proceedings of Across Grids Conference 2004*, pp. 11-20. In Dikaiakos, M. (ed.), Lecture notes in Computer Science, volume 3165, Springer-Verlag, Germany.

Digipede Technologies. (2006). The digipede network. Website <http://www.digipede.net/products/digipede-network.html>. Last accessed 13th February 2007.

EDG WP6 Integration Team. (2003). Data grid installation guide. Document identifier: DataGrid-06-TED-0105-2-0. Available online <http://marianne.in2p3.fr/datagrid/documentation/EDG-Installation-Guide-2.0.pdf>. Last accessed 16th March 2007.

EGEE. (2007). Enabling grids for e-science project. Website <http://www.eu-egee.org/>. Last accessed 12th February 2007.

Ellisman, M. and Peltier, S. (2004). Medical data federation: the biomedical informatics research network. In Foster, I. and Kesselman, C. (eds.), *The Grid: Blueprint for a New Computing Infrastructure (2nd Edition)*, chapter 8. San Francisco, CA: Morgan Kaufmann.

Elmroth, E., Ding, C., Wu, Y. and Pruess, K. (1999). A parallel implementation of the TOUGH2 software package for large scale multiphase fluid and heat flow simulations. In *Proceedings of the 1999 Conference on Supercomputing*, article no. 52. ACM Press, New York, NY, USA.

Eltis, E. and Komolkin, A. V. (2004). Comparative analysis of PVM and MPI for the development of physical applications on parallel clusters. In *Proceedings of the 2004 Joint Advanced Student School (JASS 2004)*. Available online <http://www.mayr.informatik.tu-muenchen.de/konferenzen/Jass04/courses/2/Papers/Comparison.pdf>. Last accessed 12th March 2007.

EU-DataGrid. (2004). The data grid project. Website <http://eu-datagrid.web.cern.ch/eu-datagrid/>. Last accessed 12th February 2007.

Fischer, M. C., Adams, A. and Miller, G. (1994). Aggregate level simulation protocol (ALSP) - training for the future. In *Proceedings of the 1994 Military Operations Research Symposium*. Military Operations Research Society (MORS), USA. Available online http://ms.ie.org/alsp/biblio/mors_94_fischer/mors_94_fischer.html. Last accessed 15th February 2007.

Fishwick, P. A. (1997). Web-based simulation. In *Proceedings of the 29th Winter Simulation Conference*, Andradottir, S., Healy, K. J., Withers, D. H. and Nelson, B. L. (eds.), pp. 100-102. ACM Press, New York, NY, USA.

Flynn, M. J. (1966). Very high-speed computing systems. *Proceedings of the IEEE*, 54(12): 1901-1909.

Forsdick, H. C., Schantz, R. E. and Thomas, R. H. (1978). Operating systems for computer networks. *Computer*, 11(1): 48-57.

Foster, I. (1995). Designing and building parallel programs: concepts and tools for parallel software engineering. Reading, Mass.: Addison-Wesley.

Foster, I. (2002). What is the grid? A three point checklist. Grid Today, July 2002. Available online <http://www.gridtoday.com/02/0722/100136.html>. Last accessed 18th March 2007.

Foster, I. (2005). A globus primer (draft version). Available online <http://www.globus.org/toolkit/docs/4.0/key/>. Last accessed 13th February 2007.

Foster, I. (2006). Globus toolkit version 4: software for service-oriented systems. *Journal of Computer Science and Technology*, 21(4): 513-520.

Foster, I. and Kesselman, C. (1998). The grid: blueprint for a new computing infrastructure. San Francisco, CA: Morgan Kaufmann.

Foster, I. and Kesselman, C. (2004). Concepts and architecture. In Foster, I. and Kesselman, C. (eds.), *The Grid: Blueprint for a New Computing Infrastructure (2nd Edition)*, chapter 4. San Francisco, CA: Morgan Kaufmann.

Foster, I. and Lamnitchi, A. (2003). On death, taxes, and the convergence of peer-to-peer and grid computing. In *Proceedings of the 2nd International Workshop on Peer-to-Peer Systems (IPTPS'03)*, pp.118–128. In Kaashoek, F. and Stoica, I. (eds.), Lecture notes in Computer Science, volume 2735, Springer-Verlag, Germany.

Foster, I., Geisler, J., Nickless, B., Smith, W. and Tuecke, S. (1996). Software infrastructure for the I-WAY high-performance distributed computing experiment. In *Proceedings of the 5th IEEE Symposium on High Performance Distributed Computing (HPDC'96)*, pp. 562-571. IEEE Computer Society, Washington, DC, USA.

Foster, I., Kesselman, C. and Tuecke, S. (2001). The anatomy of the grid: enabling scalable virtual organizations. *International Journal of High Performance Computing Applications*, 15(3): 200-222.

Foster, I., Kesselman, C., Nick, J. M. and Tuecke, S. (2002). Grid services for distributed system integration. *IEEE Computer*, 35(6): 37-46.

Frey, J. (2002). Condor DAGMan: handling inter-job dependencies. Available online <http://www.bo.infn.it/calcolo/condor/dagman/>. Last accessed 27th February 2007.

Fujimoto, R. M. (1990). Parallel discrete event simulation. *Communications of the ACM*, 33(10): 30-53.

Fujimoto, R. M. (1999a). Parallel and distributed simulation. In *Proceedings of the 31st Winter Simulation Conference*, Farrington, P. A., Nembhard, H. B., Sturrock, D. T. and Evans, G. W. (eds.), pp. 122– 131. ACM Press, New York, NY, USA.

Fujimoto, R. M. (1999b). Parallel and distributed simulation systems. New York, NY: John Wiley & Sons.

Fujimoto, R. M. (2001). Parallel and distributed simulation systems. In *Proceedings of the 33rd Winter Simulation Conference*, Peters, B. A., Smith, J. S., Medeiros, D. J. and Rohrer, M. W. (eds.), pp. 147-157. IEEE Computer Society, Washington, DC, USA.

Fujimoto, R. M. (2003). Distributed simulation systems. In *Proceedings of the 35th Winter Simulation Conference*, Chick, S., Sánchez, P. J., Ferrin, D. and Morrice, D. J. (eds.), pp. 124-134. Winter Simulation Conference, USA.

Fujimoto, R. M. and Weatherly, R. M. (1996). Time management in the DoD high level architecture. In *Proceedings of the 10th Workshop on Parallel and Distributed Simulation Workshop*, pp. 60-67. IEEE Computer Society, Washington, DC, USA.

Gan, B. P., Liu, L., Jain, S., Turner, S. J., Cai, W. and Hsu, W. (2000). Distributed supply chain simulation across enterprise boundaries. In *Proceedings of the 32nd Winter Simulation Conference*, Joines, J. A., Barton, R. R., Kang, K. and Fishwick, P. A. (eds.), pp. 1245-1251. Society for Computer Simulation International, San Diego, CA, USA.

Gan, B. P., Lendermann, P., Low, M. Y. H., Turner, S. J., Wang, X. and Taylor, S. J. E. (2005). Interoperating Autosched AP using the high level architecture. In *Proceedings of the 37th Winter Simulation Conference*, Kuhl, M. E., Steiger, N. M., Armstrong, F. B. and Joines, J. A. (eds.), pp. 394-401. Winter Simulation Conference, USA.

Gani, S. and Picuri, P. (1995). The object revolution: how COM technology is changing the way we do business. *Computing & Control Engineering Journal*, 6(3): 108-112.

Garonne, V., Tsaregorodtsev, A. and Stokes-Rees, I. (2004). DIRAC: workload management system. In *Proceedings of Computing in High Energy and Nuclear Physics (CHEP2004)*. Available online <https://twiki.cern.ch/twiki/bin/viewfile/LHCb/DiracReview?rev=1.1;filename=diracWMS.pdf>. Last accessed 7th May 2007.

Garonne, V., Tsaregorodtsev, A. and Caron, E. (2005). A study of meta-scheduling architectures for high throughput computing: pull versus push. In *Proceedings of the 4th International Symposium on Parallel and Distributed Computing (ISPDC'05)*, pp. 226-233. IEEE Computer Society, Washington, DC, USA.

Geist, A., Beguelin, A., Dongarra, J., Jiang, W., Manchek, R. and Sunderam. V. (1994). PVM: parallel virtual machine: a users' guide and tutorial for networked parallel computing. Cambridge, MA: MIT Press. Available online <http://www.netlib.org/pvm3/book/pvm-book.html>. Last accessed 27th February 2007.

Gentzsch, W. (2004). Enterprise resource management: applications in research and industry. In Foster, I. and Kesselman, C. (eds.), *The Grid: Blueprint for a New Computing Infrastructure (2nd Edition)*, chapter 12. San Francisco, CA: Morgan Kaufmann.

Giesler, M. and Pohlmann, M. (2003). The anthropology of file sharing: consuming Napster as a gift. *Advances in Consumer Research*, volume 30, pp 273-279. Available online <http://www.acrwebsite.org/volumes/display.asp?id=8790>. Last accessed 4th March 2007.

Globus Alliance. (2005). GT4 administration guide. Available online <http://www.globus.org/toolkit/docs/4.0/admin/docbook/index.html>. Last accessed 15th March 2007.

Globus Alliance. (2007a). GT 4.0 common runtime components: key concepts. Website <http://www.globus.org/toolkit/docs/4.0/common/key/>. Last accessed 29th April 2007.

Globus Alliance. (2007b). Research papers from globus alliance members. Website <http://www.globus.org/alliance/publications/papers.php#Applications>. Last accessed 13th February 2007.

Goldchleger, A., Queiroz, C. A., Kon, F., Goldman, A. (2004). Running highly-coupled parallel applications in a computational grid. In *Proceedings of the 22nd Brazilian Symposium on Computer Networks (SBRC'2004)*. Available online <http://gsd.ime.usp.br/publications/InteGradeBSPSBRC04.pdf>. Last accessed 29th April 2007.

Good, N. S. and Krekelberg, A. (2003). Usability and privacy: a study of kaza P2P file-sharing. In *Proceedings of the SIGCHI conference on human factors in computing systems*, pp. 137-144. ACM Press, New York, NY, USA.

Goux, J.-P., Kulkarni, S., Linderoth, J. and Yoder, M. (2000). An enabling framework for master-worker applications on the Computational grid. In *Proceedings of the 9th International Symposium on High Performance Distributed Computing (HPDC'00)*, pp. 43-50. IEEE Computer Society, Washington, DC, USA.

Gray, N. D., Hotchkiss, J., LaForge, S., Shalit, A. and Weinberg, T. (1998). Modern languages and Microsoft's component object model. *Communications of the ACM*, 41(5): 55-65.

Grid3. (2007). An application grid laboratory for science. Website <http://www.ivdgl.org/grid2003/>. Last accessed 12th February 2007.

Grimshaw, A. S. and Wulf, W. A. (1996). Legion - a view from 50,000 feet. In *Proceedings of the 5th IEEE International Symposium on High Performance Distributed Computing (HPDC'96)*, pp. 89-100. IEEE Computer Society, Washington, DC, USA.

Hantz, F. and Guyennet, H. (2005). HiPoP: highly distributed platform of computing. In *Proceedings of the Joint International Conference on Autonomic and Autonomous Systems and International Conference on Networking and Services*, pp. 91-96. IEEE Computer Society, Washington, DC, USA.

Herzog, T. N. and Lord, G. (2002). Applications of monte carlo methods to finance and insurance. Winstead, Conn: ACTEX Publications. Available online <http://books.google.com>. Last accessed 11th March 2007.

Herzog, T. N. and Lord, G. (2003). Applications of simulation models in finance and insurance. In *Proceedings of the 35th Winter Simulation Conference*, Chick, S., Sánchez, P. J., Ferrin, D. and Morrice, D. J. (eds.), pp. 249-257. Winter Simulation Conference, USA.

Hey, T. and Trefethen A. E. (2002). The UK e-science core programme and the grid. *Future Generation Computer Systems*, 18(8): 1017-1031.

Heymann, E., Senar, M. A., Luque, E. and Livny, M. (2000). Adaptive scheduling for master-worker applications on the computational grid. In *Proceedings of the 1st International Workshop on Grid Computing*, pp. 214-227. In Buyya, R. and Baker, M. (eds.), Lecture Notes in Computer Science, volume 1971, Springer-Verlag, UK.

Hibino, H., Fukuda, Y., Yura, Y., Mitsuyuki, K. and Kaneda, K. (2002). Manufacturing adapter of distributed simulation systems using HLA. In *Proceedings of the 34th Winter Simulation Conference*, Yücesan, E., Chen, C. H., Snowdon, J. L. and Charnes, J. M. (eds.), pp. 1099-1107. Winter Simulation Conference, USA.

Hollocks, B. W. (2006). Forty years of discrete-event simulation - a personal reflection. *Journal of the Operational Research Society*, 57(12): 1383-1399.

Huang, Z., Song, G. and Zheng, Y. (2006). Proxy-based parallel visualization in a grid environment with PC clusters. In *Proceedings of the first International Multi-Symposiums on Computer and Computational Sciences (IMSCC'06)*, pp. 683 - 687. IEEE Computer Society Press, Los Alamitos, CA, USA.

IEEE 1516 (2000). IEEE standard for modelling and simulation (M&S) high level architecture (HLA). New York, NY: Institute of Electrical and Electronics Engineers.

IEEE 1516.0. (2000). IEEE standard for modelling and simulation (M&S) high level architecture (HLA) – rules. New York, NY: Institute of Electrical and Electronics Engineers.

IEEE 1516.1. (2000). IEEE standard for modelling and simulation (M&S) high level architecture (HLA) - federate interface specification (FIS). New York, NY: Institute of Electrical and Electronics Engineers.

IEEE 1516.2. (2000). IEEE standard for modelling and simulation (M&S) high level architecture (HLA) - object model template (OMT) specification. New York, NY: Institute of Electrical and Electronics Engineers.

IEEE 1516.3. (2003). IEEE standard for modelling and simulation (M&S) high level architecture (HLA) - federate development process (FEDEP). New York, NY: Institute of Electrical and Electronics Engineers.

Jackson, T., Austin, J., Fletcher M. and Jessop, M. (2003). Delivering a grid enabled distributed aircraft maintenance environment (DAME). In *Proceedings of the 2003 UK e-Science All Hands Meeting*, pp. 420-427. Available online <http://www.nesc.ac.uk/events/ahm2003/AHMCD/>. Last accessed 13th February 2007.

Jefferson, D. R. (1985). Virtual Time. *ACM Transactions on Programming Languages and Systems*, 7(3): 404 – 425.

Johnson, C. (2003). What is research in computing science? Teaching notes, Department of Computer Science, University of Glasgow. Available online http://www.dcs.gla.ac.uk/~johnson/teaching/research_skills/research.html. Last accessed 28th March 2007.

Johnson, G. D. (1999). Networked simulation with HLA and MODSIM III. In *Proceedings of the 31st Winter Simulation Conference*, Farrington, P. A., Nembhard, H. B., Sturrock, D. T. and Evans, G. W. (eds.), pp. 1065-1070. ACM Press, New York, NY, USA.

Johnston, W. E., Gannon, D. and Nitzberg, B. (1999). Grids as production computing environments: the engineering aspects of NASA's information power grid. In *Proceedings of the 8th International Symposium on High Performance Distributed Computing*, pp. 197 - 204. IEEE Computer Society, Washington, DC, USA.

Johnston, W. E., Bair, R., Foster, I., Geist, A., Kramer, W. and Simon, H. D. (2001). Science grid: enabling and deploying the SciDAC collaboratory software environment. U.S. Department of Energy Office of Science. Available online http://www.doesciencegrid.org/Grid/Grid/papers/DOE_Science_Grid_Collaboratory_Pilot_Proposal_03_14.nobudget.pdf. Last accessed 12th February 2007.

Kannan S., Roberts, M., Mayes, P., Brelsford, D. and Skovira, J. F. (2001). Workload management with LoadLeveler. IBM Redbooks. Available online <http://www.redbooks.ibm.com/abstracts/sg246038.html>. Last accessed 16th March 2007.

Karlsson, M. and Olsson, L. (2001). pRTI 1516 - rationale and design. In *Proceedings of the 2001 Fall Simulation Interoperability Workshop*. 01F-SIW-038. Simulation Interoperability Standards Organization, Orlando, Florida, USA.

Karonis, N. T., Toonen, B. and Foster, I. (2003). MPICH-G2: A grid-enabled implementation of the Message Passing Interface. *Journal of Parallel and Distributed Computing*, 63(5): 551-563.

Katsaliaki, K. (2007). Analysing the supply chain of blood in the U.K. using simulation. PhD thesis. Department of Management, University of Southampton, U.K.

Katsaliaki, K. and Brailsford, S. (2007). Using simulation to improve the blood supply chain, *Journal of the Operational Research Society*, 58(2): 219-227.

Kilgore, R. A. (2000). Silk, Java and object-oriented simulation. In *Proceedings of the 32nd Winter Simulation Conference*, Joines, J. A., Barton, R. R., Kang, K. and Fishwick, P. A. (eds.). Society for Computer Simulation International, San Diego, CA, USA.

Kiss, T. (2007). GEMICA / P-GRADE: a workflow-oriented portal and application hosting environment. Presentation: Induction to Grid Computing and the National Grid Service, Brunel University, 19th April 2007. Available online <http://indico.cern.ch/materialDisplay.py?contribId=12&sessionId=5&materialId=slides&confId=13902>. Last accessed 5th May 2007.

Kondo, D., Chien, A. and Casanova, H. (2004). Resource management for rapid application turnaround on enterprise desktop grids. In *Proceedings of the 2004 Conference on Supercomputing (SC'04)*, paper 17. IEEE Computer Society, Washington, DC, USA.

Kuhl F., Weatherly R. and Dahmann J. (1999). Creating computer simulation systems: an introduction to the high level architecture. Upper Saddle River, NJ: Prentice Hall PTR.

Kuljis, J. and Paul, R. J. (2000). A review of web based simulation: whither we wander? In *Proceedings of the 32nd Winter Simulation Conference*, Joines, J. A., Barton, R. R., Kang, K. and Fishwick, P. A. (eds.), pp.1872-1881. Society for Computer Simulation International, San Diego, CA, USA.

Kurose, J. F. and Ross, K. W. (2003). Computer networking: a top-down approach featuring the Internet (2nd edition). Reading, MA: Addison Wesley.

Ladbrook, J. and Januszczak, A. (2001). Ford's power train operations – changing the simulation environment. In *Proceedings of the 33rd Winter Simulation Conference*, Peters, B. A., Smith, J. S., Medeiros, D. J. and Rohrer, M. W. (eds.), pp.863-869. IEEE Computer Society, Washington, DC, USA.

Lamanna, M. (2004). The LHC computing grid project at CERN. *Nuclear Instruments and Methods in Physics Research (Section A: Accelerators, Spectrometers, Detectors and Associated Equipment)*, 534(1-2): 1-6.

Laughery, R. (1998). Computer simulation as tool for studying human-centered systems. In *Proceedings of the 30th Winter Simulation Conference*, Medeiros, D. J., Watson, E. F., Carson, J. S. and Manivannan, M. S. (eds.), pp. 61 - 66. IEEE Computer Society Press, Los Alamitos, CA, USA.

LCG. (2007a). LCG project overview. Website <http://lcg.web.cern.ch/LCG/overview.html>. Last accessed 12th February 2007.

LCG. (2007b). LCG middleware overview. Website <http://lcg.web.cern.ch/LCG/activities/middleware.html>. Last accessed 12th February 2007.

Lendermann, P., Low, M. Y. H., Gan, B. P., Julka, N., Peng, C. L., Lee, L. H., Taylor, S. J. E Turner, S. J., Cai, W., Wang, X., Hung, T., McGinnis, L. F. and Buckley, S. (2005). An integrated and adaptive decision-support framework for high-tech manufacturing and service networks. In *Proceedings of the 37th Winter Simulation Conference*, Kuhl, M. E., Steiger, N. M., Armstrong, F. B. and Joines, J. A. (eds.), pp. 2052-2062. Winter Simulation Conference, USA.

Levine, D. and Wirt, M. (2004). Interactivity with scalability: infrastructure for multiplayer games. In Foster, I. and Kesselman, C. (eds.), *The Grid: Blueprint for a New Computing Infrastructure (2nd Edition)*, chapter 13. San Francisco, CA: Morgan Kaufmann.

LHC@home. (2007). What is LHC@home? Website http://athome.web.cern.ch/athome/LHC_athome/whatis.html. Last accessed 17th February 2007.

Linden Research. (2007). Second Life. Website <http://secondlife.com/>. Last accessed 9th February 2007.

Litzkow, M., Livny, M. and Mutka, M. (1988). Condor - a hunter of idle workstations. In *Proceedings of the 8th International Conference of Distributed Computing Systems*, pp.104-111. IEEE Computer Society, Washington, DC, USA.

Litzkow, M., Tannenbaum, T., Basney, J. and Livny, M. (1997). Checkpoint and migration of UNIX processes in the Condor distributed processing system. University of Wisconsin-Madison Computer Sciences Technical Report 1346. Available online <http://www.cs.wisc.edu/Condor/doc/ckpt97.pdf>. Last accessed 16th February 2007.

Livny, M. and Beck, A. (1997). High throughput computing: an interview with Miron Livney. HPCWire, June 1997. Available online <http://www.cs.wisc.edu/Condor/HPCwire.1>. Last accessed 16th February 2007.

Lorenz, P., Schriber, T. J., Dorwarth, H. and Ritter, K. (1997). Towards a web based simulation environment. In *Proceedings of the 29th Winter Simulation Conference*, Andradottir, S., Healy, K. J., Withers, D. H. and Nelson, B. L. (eds.), pp.1338-1344. ACM Press, New York, NY, USA.

Low, M. Y. H., Lye, K. W., Lendermann, P., Turner, S. J., Chim, R. T. W and Leo, S. H. (2005). An agent-based approach for managing symbiotic simulation of semiconductor assembly and test operation. In *Proceedings of the 4th International Joint Conference on Autonomous Agents and Multiagent Systems (AAMAS'05)*, pp. 85-92. ACM Press, New York, NY, USA.

- Lowery, J. C. (1998). Getting started in simulation in healthcare. In *Proceedings of the 30th Winter Simulation Conference*, Medeiros, D. J., Watson, E. F., Carson, J. S. and Manivannan, M. S. (eds.), pp. 31-35. IEEE Computer Society Press, Los Alamitos, CA, USA.
- Luther, A., Buyya, R., Ranjan, R. and Venugopal, S. (2005). Alchemi: a .NET-based enterprise grid computing system. In *Proceedings of the 6th International Conference on Internet Computing (ICOMP'05)*, pp. 269-278. CSREA Press, USA. Available online http://gridbus.csse.unimelb.edu.au/papers/alchemi_icom05.pdf. Last accessed 4th April 2007.
- Lüthi, J. and Großmann, S. (2001). The resource sharing system: dynamic federate mapping for HLA-based distributed simulation. In *Proceedings of the 15th Workshop on Parallel and Distributed Simulation*, pp. 91-98. IEEE Computer Society, Washington, DC, USA.
- Mahmoud, Q. H. (2005). Service-Oriented Architecture (SOA) and web services: the road to Enterprise Application Integration (EAI). Available online <http://java.sun.com/developer/technicalArticles/WebServices/soa/>. Last accessed 4th May 2007.
- Marins, J. T. M, Santos J. F. and Saliby, E. (2004). Variance reduction techniques applied to monte carlo simulation of asian calls. In *Proceedings of the 2004 Business Association of Latin American Studies (BALAS) Conference*.
- Marr, C., Storey, C., Biles, W. E. and Kleijnen, J. P. C. (2000). A Java-based simulation manager for web-based simulation. In *Proceedings of the 32nd Winter Simulation Conference*, Joines, J. A., Barton, R. R., Kang, K. and Fishwick, P. A. (eds.), pp. 1815–1822. Society for Computer Simulation International, San Diego, CA, USA.
- Marr, D. T., Binns, F., Hill, D. L., Hinton, G., Koufaty, D. A., Miller, J. A. and Upton, M. (2002). Hyper-threading technology architecture and micro-architecture. *Intel Technology Journal*, 6(1): 4-15.
- Matsuoka, S., Shinjo, S., Aoyagi, M., Sekiguchi, S., Usami, H. and Miura, K. (2005). Japanese computational grid research project: NAREGI. *Proceedings of the IEEE*, 93(3): 522-533.
- Mccoy, R. A. and Deng, Y. (1999). Parallel particle simulations of thin-film deposition. *International Journal of High Performance Computing Applications*, 13(1): 16-32. Sage Publications, Thousand Oaks, CA, USA.

McLean, C. and Riddick, F. (2000). The IMS mission architecture for distributed manufacturing simulation. In *Proceedings of the 32nd Winter Simulation Conference*, Joines, J. A., Barton, R. R., Kang, K. and Fishwick, P. A. (eds.), pp. 1539-1548. Society for Computer Simulation International, San Diego, CA, USA.

Microsoft Support. (2007). How to automate Microsoft Excel from Visual Basic (knowledge base article KB219151, version 5). Available online <http://support.microsoft.com/kb/219151>. Last accessed 6th April, 2007.

Microsoft WCCS. (2007). Windows compute cluster server 2003 product overview. Website <http://www.microsoft.com/windowsserver2003/ccs/overview.aspx>. Last accessed 15th March 2007.

Miller, D. C. and Thorpe, J. A. (1995). SIMNET: the advent of simulator networking. *Proceedings of the IEEE*, 83(8): 1114-1123.

Mustafee, N. (2004). Performance evaluation of interoperability methods for distributed simulation. MSc. thesis. Department of Information Systems, Computing and Mathematics, Brunel University, UK.

Mustafee, N. and Taylor, S. J. E. (2006a). Investigating distributed simulation with COTS simulation packages: experiences with Simul8 and the HLA. In *Proceedings of the 2006 Operational Research Society Simulation Workshop (SW06)*, Garnett, J., Brailsford, S., Robinson, S. and Taylor, S. (eds.), pp. 33-42. Operational Research Society, Birmingham, UK.

Mustafee, N. and Taylor, S. J. E. (2006b). Using a desktop grid to support simulation modelling. In *Proceedings of the 28th Information Technology Interfaces Conference (ITI 2006)*, Stiffler, V.L. and Dobric, V. H. (eds.), pp. 557-562. IEEE Computer Society, Washington, DC, USA.

Mustafee, N., Alstad, A., Larsen, B., Taylor, S. J. E. and Ladbrook, J. (2006a). Grid-enabling FIRST: speeding up simulation applications using WinGrid. In *Proceedings of the 10th International Symposium on Distributed Simulation and Real-Time Applications (DSRT 2006)*, Alba, E., Turner, S. J., Roberts, D. and Taylor, S. J. E. (eds.), pp. 157-164. IEEE Computer Society, Washington, DC, USA.

- Mustafee, N., Taylor, S. J. E., Katsaliaki, K. and Brailsford, S. (2006b). Distributed simulation with COTS simulation packages: a case study in health care supply chain simulation. In *Proceedings of the 37th Winter Simulation Conference*, Perrone, L. F., Wieland, F. P., Liu, J., Lawson, B. G., Nicol, D. M. and Fujimoto, R. M. (eds.), pp. 1136-1142. Winter Simulation Conference, USA.
- Mutalik, P. P., Knight, L. R., Blanton, J. L. and Wainwright, R. L. (1992). Solving combinatorial optimization problems using parallel simulated annealing and parallel genetic algorithms. In *Proceedings of the 1992 ACM/SIGAPP symposium on Applied computing*, pp. 1031- 1038. ACM Press, New York, NY, USA.
- Mutka, M. W. (1992) . Estimating capacity for sharing in a privately owned workstation environment. *IEEE Transactions on Software Engineering*, 18(4): 319-328.
- National e-Science Centre. (2001). Defining e-Science. Website <http://www.nesc.ac.uk/nesc/define.html>. Last accessed 12th February 2007.
- Natrajan, A., Nguyen-Tuong, A., Humphrey, M. A., Herrick, M., Clarke, B. P. and Grimshaw, A. S. (2002). The Legion Grid Portal. *Concurrency and computation: practice and experience*, 14(13-15): 1365-1394.
- Nelson, B. L. (1987). Variance reduction for simulation practitioners. In *Proceedings of the 19th Winter Simulation Conference*, Thesen, A., Grant, H. and Kelton, W. D. (eds.), pp. 43-51. ACM Press, New York, NY, USA.
- Németh, C., Dózsa, G., Lovas, R. and Kacsuk, P. (2004). The P-GRADE Grid Portal. In *Proceedings of the International Conference on Computational Science and its Applications (ICCSA 2004)*, pp. 10-19. In Laganá et al. (eds.), Lecture notes in Computer Science, volume 3044, Springer-Verlag, Germany.
- NHS Blood and Transplant. (2006). NHS Blood and Transplant (NHSBT). Website <http://www.nhsbt.nhs.uk>. Last accessed 14th February 2007.
- Nicol, D. and Heidelberger, P. (1996). Parallel execution for serial simulators. *ACM Transactions on Modelling and Computer Simulation*, 6(3): 210-242.
- NLR (2007). National LambdaRail network: infrastructure. Website <http://www.nlr.net/services/infrastructure.php>. Last accessed 13th February 2007.
- Novotny, J. (2002). The grid portal development kit. *Concurrency and Computation: Practice and Experience*, 14(13-15): 1129-1144.

Oinn, T., Addis, M., Ferris, J., Marvin, D., Senger, M., Greenwood, M., Carver, T., Glover, K., Pocock, M. R., Wipat, A. and Li, P. (2004). Taverna: a tool for the composition and enactment of bioinformatics workflows. *Bioinformatics*, 20(17): 3045-3054.

OMII. (2006a). Installation and setup guide for the OMII middleware, release 3.2.0. Available online http://www.omii.ac.uk/docs/3.2.0/installation_guide/omii_3_installation_and_setup_guide.htm. Last accessed 15th March 2007.

OMII. (2006b). User guide for the OMII middleware, release 3.2.0. Accessible online http://www.omii.ac.uk/docs/3.2.0/user_guide/omii_user_guide.htm. Last accessed 15th March 2007.

Open Grid Forum. (2007). Overview of open grid forum (OGF). Website http://www.ogf.org/About/abt_overview.php. Last accessed 15th March 2007.

Open Science Grid. (2007). What is the open science grid (OSG)? Website <http://www.opensciencegrid.org/>. Last accessed 12th February 2007.

Page, E. H. and Nance, R. E. (1994). Parallel discrete event simulation: a modelling methodological perspective. In *Proceedings of the 8th Workshop on Parallel and Distributed Simulation*, pp. 88-93. ACM Press, New York, NY, USA.

Page, E. H. and Smith, R. (1998). Introduction to military training simulation: a guide for discrete event simulationists. In *Proceedings of the 30th Winter Simulation Conference*, Medeiros, D. J., Watson, E. F., Carson, J. S. and Manivannan, M. S. (eds.), pp. 53 - 60. IEEE Computer Society Press, Los Alamitos, CA, USA.

Page, E. H., Buss, A., Fishwick, P. A., Healy, K. J., Nance, R. E. and Paul, R. J. (2000). Web-based simulation: revolution or evolution? *ACM Transactions on Modelling and Computer Simulation*, 10(1): 3-17.

Pande, V. (2007). About Folding@Home. Website <http://folding.stanford.edu/>. Last accessed 17th February 2007.

Parabon computation. (2007). Compute against cancer. Website <http://www.computeagainstcancer.org/learnMore.jsp>. Last accessed 17th February 2007.

Paul, R. J., and Taylor, S. J. E. (2002). What use is model reuse: is there a crook at the end of the rainbow? In *Proceedings of the 34th Winter Simulation Conference*, Yücesan, E., Chen, C. H., Snowdon, J. L. and Charnes, J. M. (eds.), pp. 648-652. Winter Simulation Conference, USA.

Perez, J. A. L. (2005). BOINC architecture and basic principles, CERN presentation. Website <https://twiki.cern.ch/twiki/pub/LHCAAtHome/LinksAndDocs/boincciemat06.pdf>. Last accessed 17th February 2007.

Peris, A. D., Lorenzo, P. M., Donno, F., Sciaba, A., Campana, S. and Santinelli, R. (2005). LCG-2 user guide, manuals series. Document identifier: CERN-LCG-GDEIS-454439. Available online <https://edms.cern.ch/file/454439//LCG-2-UserGuide.pdf>. Last accessed 15th March 2007.

Pidd, M. (2002). Simulation software and model reuse: a polemic. In *Proceedings of the 34th Winter Simulation Conference*, Yücesan, E., Chen, C. H., Snowdon, J. L. and Charnes, J. M. (eds.), pp. 772-775. Winter Simulation Conference, USA.

Pidd, M. (2004a). Computer simulation in management science (5th edition). Chichester, UK: John Wiley & Sons.

Pidd, M. (2004b). Simulation worldviews: so what? In *Proceedings of the 36th Winter Simulation Conference*, Ingalls, R. G., Rossetti, M. D., Smith, J. S. and Peters, B. A. (eds.), pp. 288-292. Winter Simulation Conference, USA.

Pidd, M. and Carvalho, M. A. (2006). Simulation software: not the same yesterday, today or forever. *Journal of Simulation*, 1(1): 7-20.

Pidd, M., Oses, N. and Brooks, R. J. (1999). Component-based simulation on the web. In *Proceedings of the 31st Winter Simulation Conference*, Farrington, P. A., Nembhard, H. B., Sturrock, D. T. and Evans, G. W. (eds.), pp. 1438-1444. ACM Press, New York, NY, USA.

Quinn, K., Turner, V. and Yang, J. (2005). The next evolution in enterprise computing: the convergence of multicore x86 processing and 64-bit operating systems. IDC Whitepaper no. 05C4442. Available online http://multicore.amd.com/Resources/IDC_Convergence_en.pdf. Last accessed 13th April 2007.

Rajaei, H., Ayani, R. and Thorelli, L. (1993). The local time warp approach to parallel simulation. In *Proceedings of the 7th Workshop on Parallel and Distributed Simulation*, pp. 119 – 126. ACM Press, New York, NY, USA.

Reed, D. A. (2003). Grids, the teragrid and beyond. *IEEE Computer*, 36(1): 62-68. IEEE Computer Society Press, Los Alamitos, CA, USA.

Reynolds, P. F. (1988). A spectrum of options for parallel simulation. In *Proceedings of the 20th Winter Simulation Conference*, Abrams, M., Haigh, P. and Comfort, J. (eds.), pp. 325 – 332. ACM Press, New York, NY, USA.

Robinson, E. and DeWitt, D. J. (2007). Turning cluster management into data management: a system overview. In *Proceedings of 3rd Biennial Conference on Innovative Data Systems Research (CIDR)*. Available online <http://www-db.cs.wisc.edu/cidr/cidr2007/papers/cidr07p14.pdf>. Last accessed 10th May 2007.

Robinson, S. (2005a). Discrete-event simulation: from the pioneers to the present, what next? *Journal of the Operational Research Society*, 56 (6): 619-629.

Robinson, S. (2005b). Distributed simulation and simulation practice. *Simulation*, 81(5): 5-13.

Robinson, S. and Pidd, M. (1998). Provider and customer expectations of successful simulation projects. *Journal of the Operational Research Society*, 49(3): 200-209.

Robinson, S., Nance, R. E., Paul, R. J., Pidd, M. and Taylor, S. J. E. (2004). Simulation model reuse: definitions, benefits and obstacles. *Simulation Modelling Practice and Theory*, 12(7-8): 479-494.

Ryde M. (2005). A high usability transparency framework for model interoperability using COTS distributed simulation. PhD thesis. School of Information Systems, Computing and Mathematics, Brunel University, UK.

Saiz, P., Aphecetche, L., Buncic, P., Piskac, R., Revsbech, J.-E. and Sego, V. (2003). AliEn—ALICE environment on the grid. *Nuclear Instruments and Methods in Physics Research (Section A: Accelerators, Spectrometers, Detectors and Associated Equipment)*, 502(2-3): 437-440.

Saliby, E. (1997). Descriptive sampling: an improvement over latin hypercube sampling. In *Proceedings of the 29th Winter Simulation Conference*, Andradottir, S., Healy, K. J., Withers, D. H. and Nelson, B. L. (eds.), pp.230-233. ACM Press, New York, NY, USA.

- Saroiu, S., Gummadi, P. K. and Gribble, S. D. (2002). A measurement study of peer-to-peer file sharing systems. In *Proceedings of the Multimedia Computing and Networking (MMCN)*. Available online <http://www.cs.toronto.edu/~stefan/publications/mmcn/2002/mmcn.html>. Last accessed 18th March 2007.
- Schopf, J. M. and Nitzberg, B. (2002). Grids: the top ten questions. *Scientific Programming*, 10(2): 103-111.
- Segal, B. (2000). Grid computing: the European data grid project. In *Proceedings of the 2000 IEEE Nuclear Science Symposium and Medical Imaging Conference*, pp. 15–20. IEEE Computer Society, Washington, DC, USA. Available online http://edg-wp2.web.cern.ch/edg-wp2/docs/NSS_Paper.pdf. Last accessed 4th April 2007.
- Shannon, R. E. (1998). Introduction to the art and science of simulation. In *Proceedings of the 30th Winter Simulation Conference*, Medeiros, D. J., Watson, E. F., Carson, J. S. and Manivannan, M. S. (eds.), pp. 7-14. IEEE Computer Society Press, Los Alamitos, CA, USA.
- Simul8 Corporation (2002). Simul8 COM professional edition documentation. Available online <http://people.brunel.ac.uk/~cspgngnm/Simul8COMDocumentation.pdf>. Last accessed 6th April 2007.
- Smarr, L. and Catlett, C. E. (1992). Metacomputing. *Communications of the ACM*, 35(6): 44-52.
- Spencer, B., Finholt, T. A., Foster, I., Kesselman, C., Beldica, C., Futrelle, J., Gullapalli, S., Hubbard, P., Liming, L., Marcusiu, D., Pearlman, L., Severance, C. and Yang, G. (2004). NEESgrid: A distributed collaboratory for advanced earthquake engineering experiment and simulation. In *Proceedings of the 13th World Conference on Earthquake Engineering*, paper No. 1674. Available online <http://www.globus.org/alliance/publications/papers/13worldconferenceonEarthquakeEngineering-rad8A451.pdf>. Last accessed 4th April 2007.
- Stainforth, D., Kettleborough, J., Allen, M., Collins, M., Heaps, A. and Murphy, J. (2002). Distributed computing for public interest climate modelling research. *Computing in Science and Engineering*, 4(3): 82-89.
- Stevens, R. and Futures Lab Group. (2004). Group-oriented collaboration: the access grid collaboration system. In Foster, I. and Kesselman, C. (eds.), *The Grid: Blueprint for a New Computing Infrastructure (2nd Edition)*, chapter 15. San Francisco, CA: Morgan Kaufmann.

Strassburger, S., Schulze, T., Klein, U. and Henriksen, J. O. (1998). Internet-based simulation using off-the-shelf simulation tools and HLA. In *Proceedings of the 30th Winter Simulation Conference*, Medeiros, D. J., Watson, E. F., Carson, J. S. and Manivannan, M. S. (eds.), pp. 1669-1676. IEEE Computer Society Press, Los Alamitos, CA, USA.

Sudra, R., Taylor, S. J. E. and Janahan, T. (2000). Distributed supply chain simulation in grids. In *Proceedings of the 32nd Winter Simulation Conference*, Joines, J. A., Barton, R. R., Kang, K. and Fishwick, P. A. (eds.), pp. 356-361. Society for Computer Simulation International, San Diego, CA, USA.

Sun Microsystems Limited. (2000). Java native interface (JNI). Website <http://java.sun.com/j2se/1.3/docs/guide/jni/>. Last accessed 15th February 2007.

Sun, Q., Daswani, N. and Garcia-Molina, H. (2006). Maximizing remote work in flooding-based peer-to-peer systems. *Computer Networks*, 50(10): 1583-1598.

Swain, J. J. (2003). Simulation reloaded: sixth biennial survey of discrete-event software tools. *OR/MS Today*, 30(4): 46-57. Institute for Operations Research and the Management Sciences (INFORMS), USA. Available online <http://www.lionhrtpub.com/orms/orms-8-03/frsurvey.html>. Last accessed 4th April 2007.

Swain J. J. (2005). Gaming reality: biennial survey of discrete-event simulation software tools. *OR/MS Today (December 2005)*. Institute for Operations Research and the Management Sciences (INFORMS), USA. Available online <http://www.lionhrtpub.com/orms/orms-12-05/frsurvey.html>. Last accessed 4th April 2007.

Swain J. J. (2007). INFORMS simulation software survey. *OR/MS Today*. Institute for Operations Research and the Management Sciences (INFORMS), USA. Available online <http://www.lionhrtpub.com/orms/surveys/Simulation/Simulation.html>. Last accessed 4th April 2007.

Systemflow Simulations. (2006). Systemflow 3D animator (S3DA). Website <http://www.systemflow.com/s3d.php>. Last accessed 12th March 2007.

Tanenbaum, A. S., Renesse, R. V., Staveren, H. V., Sharp, G. J. and Mullender, S. J. (1990). Experiences with the amoeba distributed operating system. *Communications of the ACM*, 33(12): 46-63.

Taufer, M., An,C., Kerstens, A. and Brooks, C. L. (2006). Predictor@home: a protein structure prediction supercomputer based on global computing. *IEEE Transactions on Parallel and Distributed Systems*, 17(8): 786-796.

Taylor, S. J. E. (2000). Groupware and the simulation consultant. In *Proceedings of the 32nd Winter Simulation Conference*, Joines, J. A., Barton, R. R., Kang, K. and Fishwick, P. A. (eds.), pp.83-89. Society for Computer Simulation International, San Diego, CA, USA.

Taylor, S. J. E. and Robinson, S. (2006). So where to next? A survey of the future for discrete-event simulation. *Journal of Simulation*, 1(1): 1-6.

Taylor, S. J. E., Saville, J. and Sudra, R. (1999). Developing interest management techniques in distributed interactive simulation using java. In *Proceedings of the 31st Winter Simulation Conference*, Farrington, P. A., Nembhard, H. B., Sturrock, D. T. and Evans, G. W. (eds.), pp. 518 – 523. ACM Press, New York, NY, USA.

Taylor, S. J. E., Sudra, R., Janahan, T., Tan, G. and Ladbrook, J. (2001). Towards COTS distributed simulation using grids. In *Proceedings of the 33rd Winter Simulation Conference*, Smith, J. S., Medeiros, D. J. and Rohrer, M. W. (eds.), pp. 1372-1379. IEEE Computer Society, Washington, DC, USA.

Taylor, S. J. E., Sudra, R., Janahan, T., Tan, G. and Ladbrook, J. (2002). GRIDS-SCF: an infrastructure for distributed supply chain simulation. *Simulation*, 78(5): 312-320.

Taylor, S. J. E, Sharpe, J. and Ladbrook, J. (2003). Time management issues in COTS distributed simulation: a case study. In *Proceedings of the 35th Winter Simulation Conference*, Chick, S., Sánchez, P. J., Ferrin, D. and Morrice, D. J. (eds.), pp.838-846. Winter Simulation Conference, USA.

Taylor, S. J. E., Bohli, L., Wang, X., Turner, S. J and Ladbrook, J. (2005a). Investigating distributed simulation at the ford motor company. In *Proceedings of the 9th International Symposium on Distributed Simulation and Real-Time Applications (DSRT 2005)*, pp. 139-147. IEEE Computer Society, Washington, DC, USA.

Taylor, S. J. E, Turner, S. J., Mustafee, N., Ahlander, H. and Ayani, R. (2005b). COTS distributed simulation: a comparison of CMB and HLA interoperability approaches to type I interoperability reference model problems. *Simulation*, 81(1): 33–43.

Taylor, S. J. E, Turner, S. J., Low, M. Y. H., Wang, X., Strassburger, S. and Ladbrook, J. (2006a). Developing interoperability standards for distributed simulation and COTS simulation packages with the CSPI PDG. In *Proceedings of the 37th Winter Simulation Conference*, Perrone, L. F., Wieland, F. P., Liu, J., Lawson, B. G., Nicol, D. M. and Fujimoto, R. M. (eds.), pp. 1101-1110. Winter Simulation Conference, USA.

Taylor, S. J. E., Wang, X., Turner, S. J. and Low, M. Y. H. (2006b). Integrating heterogeneous distributed COTS discrete-event simulation packages: an emerging standards-based approach. *IEEE Transactions on Systems, Man and Cybernetics: Part A*, 36(1): 109-122.

Tewoldeberhan, T. W., Verbraeck, A., Valentin, E. C. and Bardonnnet, G. (2002). An evaluation and selection methodology for discrete-event simulation software. In *Proceedings of the 34th Winter Simulation Conference*, Yücesan, E., Chen, C. H., Snowdon, J. L. and Charnes, J. M. (eds.), pp. 67-75. Winter Simulation Conference, USA.

Thain, D., Tannenbaum, T. and Livny, M. (2004). Distributed computing in practice: the Condor experience. *Concurrency and Computation: Practice and Experience*, 17(2-4): 323-356.

United Devices. (2007). Grid MP: The technology for enterprise application virtualization. Website <http://www.ud.com/products/gridmp.php>. Last accessed 18th March 2007.

US Department of Defense Modelling and Simulation Office. (1999). High level architecture run-time infrastructure RTI 1.3-next generation programmer's guide. US Department of Defence Modelling and Simulation Office, USA.

Vaishnavi, V. and Kuechler, W. (2006). Design research in information systems. Available online <http://www.isworld.org/Researchdesign/drisISworld.htm>. Last accessed 28th March 2007.

Veeke, H., Saanen, Y., Rengelink, W., Verbraeck, A. and Ham, R. (2002). Simulation backbone Famas.MV2. Project 0.2 (final report). Onderzoekschool voor Transport, Infrastructuur en Logistiek (TRAIL), Technical University of Delft, Delft, The Netherlands.

Virtual Data Toolkit. (2007). What is in VDT 1.6.1 (supporting platforms)? Website <http://vdt.cs.wisc.edu/releases/1.6.1/contents.html>. Last accessed 16th March 2007.

Walli, S. R. (1995). The posix family of standards. *StandardView*, 3(1): 11-17.

- Wang, X., Turner, S. J., Low, M. Y. H. and Gan, B. P. (2004). Optimistic synchronization in HLA based distributed simulation. In *Proceedings of the 18th Workshop on Parallel and Distributed Simulation*, pp. 123-130. ACM Press, New York, NY, USA.
- Wang, X., Turner, S. J., Low, M. Y. H and Taylor, S. J. E. (2006). COTS simulation package (CSP) interoperability – a solution to synchronous entity passing. In *Proceedings of the 20th Workshop on Principles of Advanced and Distributed Simulation*, pp. 201-210. IEEE Computer Society, Washington, DC, USA.
- Welch, V., Siebenlist, F., Foster, I., Bresnahan, J., Czajkowski, K., Gawor, J., Kesselman, C., Meder, S., Pearlman, L. and Tuecke, S. (2003). Security for grid services. In *Proceedings of the 12th IEEE International Symposium on High Performance Distributed Computing (HPDC'03)*, pp.48-57. IEEE Computer Society, Washington, DC, USA.
- Whitman, L., Huff, B. and Palaniswamy, S. (1998). Commercial simulation over the web. In *Proceedings of the 30th Winter Simulation Conference*, Medeiros, D. J., Watson, E. F., Carson, J. S. and Manivannan, M. S. (eds.), pp.335-339. IEEE Computer Society Press, Los Alamitos, CA, USA.
- Woltman, G. (2007). GIMPS: how it works. Website <http://www.mersenne.org/works.htm>. Last accessed 3rd March 2007.
- World Wide Web Consortium (W3C). (2000). Simple object access protocol (SOAP) 1.1. W3C working group note 8 May 2000. Available online <http://www.w3.org/TR/2000/NOTE-SOAP-20000508/>. Last accessed 16th March 2007.
- World Wide Web Consortium (W3C). (2004). Web service architecture. W3C working group note 11 February 2004. Available online <http://www.w3.org/TR/ws-arch/>. Last accessed 15th March 2007.
- Yang, X., Chohan, D., Wang, X. D. and Allan, R. (2005). A web portal for the national grid service. In *Proceedings of the 2005 UK e-Science All Hands Meeting*, pp. 1156–1162. Available online <http://epubs.cclrc.ac.uk/bitstream/1084/paper05C.pdf>. Last accessed 4th April 2007.
- Yau, V. (1999). Automating parallel simulation Using parallel time streams. *ACM Transactions on Modeling and Computer Simulation (TOMACS)*, 9(2): 171- 201.

- Yu, J. and Buyya, R. (2004). A novel architecture for realizing grid workflow using tuple spaces. In *Proceedings of the 5th International Workshop on Grid Computing (Grid 2004)*, pp. 119- 128. IEEE Computer Society, Washington, DC, USA.
- Yu, J. and Buyya, R. (2006). A taxonomy of workflow management systems for grid computing. *Journal of Grid Computing*, 3(3-4): 171-200.
- Yücesan, E., Chen, C. H. and Lee, I. (1998). Web-based simulation experiments. In *Proceedings of the 30th Winter Simulation Conference*, Medeiros, D. J., Watson, E. F., Carson, J. S. and Manivannan, M. S. (eds.), pp. 1649-1654. IEEE Computer Society Press, Los Alamitos, CA, USA.
- Zhang, J. (2006). Investigation of the use of BOINC in organizations. MSc. thesis. School of Information Systems, Computing and Mathematics, Brunel University, UK. Available online http://people.brunel.ac.uk/~cspgnm/MSc_Jingri.doc. Last accessed 3rd April 2007.
- Zhou, S. (1992). LSF: Load sharing in large-scale heterogeneous distributed systems. In *Proceedings of the 1992 Workshop on Cluster Computing*. Supercomputing Computations Research Institute, Florida State University, Florida, USA.
- Zimmers, E. W. and Brinker, T. W. (1978). The application of computer simulation techniques to industrial packaging lines. In *Proceedings of the 10th Winter Simulation Conference*, Highland H. J (ed.), pp. 721-725. IEEE Computer Society Press, Los Alamitos, CA, USA.

APPENDIX A: Vendor URLs

Appendix A.1: Vendor URLs – parallel computing support

Table 44: Vendor URLs – support for parallel computing

Software	Vendor	URL	Date Accessed
"@Risk Industrial"	Palisade Corporation	http://www.palisade.com/risk/	10th February 2007
TreeAge Pro	TreeAge Software, Inc.	http://www.treeage.com/products/proNew.html	9th February 2007

Appendix A.2: Vendor URLs – task farming support**Table 45: Vendor URLs – task farming support in CSPs**

Software	Vendor	URL	Date Accessed
GoldSim Monte Carlo	GoldSim Technology Group	http://www.goldsim.com/Content.asp?PageID=43	9th February 2007
SIMPROCESS	CACI Products Company	http://www.simsript.com/products/simprocess31.cfm	9th February 2007
Simul8 Professional and Standard Editions	Simul8 Corporation	http://www.simul8.com/support/newsletter/Parallel_Processing.htm	3rd May 2007
Vanguard Studio (DecisionPro)	Vanguard Software Corporation	http://www.vanguardsw.com/products/add-ins/grid-computing/	11th February 2007

Appendix A.3: Vendor URLs – data source access support**Table 46: Vendor URLs – data source access support in CSPs**

Software	Vendor	URL	Date Accessed
AnyLogic 6.0	XJ Technologies	http://www.xjtek.com/anylogic/features/	12th March 2007
Arena	Rockwell Automation	http://www.arenasimulation.com/products/feature_matrix.asp	12th March 2007
Enterprise Dynamics Studio	Incontrol Enterprise Dynamics	http://incontrol.nl/?to=features	12th March 2007
GoldSim Monte Carlo	GoldSim Technology Group	http://www.goldsim.com/Content.asp?PageID=258	12th March 2007
Simprocess	CACI Products Company	http://www.simprocess.com/pdf/SOA-SimulationOnDemand-Simprocess.pdf	11th February 2007
Vanguard Studio (DecisionPro)	Vanguard Software Corporation	http://www.vanguardsw.com/products/add-ins/web-services/	11th February 2007
WITNESS 2006	Lanner Group	http://www.lanner.com/en/simulation_professionals/witness_suite.php	12th March 2007

Appendix A.4: Vendor URLs – CSPs that expose package functionality

Table 47: Vendor URLs – CSPs that expose package functionality

Software	Vendor	URL	Date Accessed
AgenaRisk Enterprise Edition	AgenaRisk	http://www.agenarisk.com/newsletters/	9th February 2007
Simprocess	CACI Products Company	http://www.simprocess.com/products/simprocessKFTP.cfm	9th February 2007
Simcad Pro	CreateASoft, Inc.	http://www.createasoft.com/processImprovementSimulator/leanProcessSimulationSoftware/SimcadProProcessSimulator7.2.html	9th February 2007
Crystal Ball Professional and Premium Editions	Decisioneering	http://www.crystalball.com/cbpro/devkit.html	9th February 2007
GoldSim	GoldSim Technology Group	http://www.goldsim.com/Content.asp?PageID=474	9th February 2007
Extend Industry, Extend OR and Extend Suite	Imagine That, Inc.	http://www.imaginethatinc.com/sols_advantage.html	9th February 2007
Enterprise Dynamics Studio	Incontrol Enterprise Dynamics	http://incontrol.nl/?to=product_falcon	9th February 2007
Analytica	Lumina Decision Systems, Inc	http://www.lumina.com/ana/newtoana3.1.htm	9th February 2007
Witness	Lanner	http://www.lanner.com/en/simulation_professionals/simulation_developer_kit.php	11th February 2007
@RiskProfessional	Palisade Corporation	http://www.palisade-europe.com/risk/	10th February 2007
Enterprise Dynamics	Production Modelling Corporation	http://www.pmc Corp.com/ed/index.shtm	10th February 2007
ProModel	ProModel Corporation	http://www.promodel.com/products/promodel/features.asp	10th February 2007
Arena	Rockwell Automation	http://www.arenasimulation.com/products/feature_matrix.asp	10th February 2007
Simul8 Standard and Professional Editions	Simul8 Corp	http://www.simul8.com/products/features/index.htm	10th February 2007
eM-Plant	UGS	http://www.ugs.com/products/tecnomatix/docs/fs_tecnomatix_em_plant.pdf	10th February 2007
AnyLogic	XJ Technologies	http://www.xjtek.com/anylogic/features/	11th February 2007

Appendix A.5: Vendor URLs – support for reusable modelling components**Table 48: Vendor URLs – reusable model components support in CSPs**

Software	Vendor	URL	Date Accessed
Crystal Ball Standard and Professional Editions	Decisioneering	http://www.crystalball.com/crystal_ball/index.html	9th February 2007
Extend Industry, Extend OR and Extend Suite	Imagine That, Inc.	http://www.imaginethatinc.com/sols_advantage.html	9th February 2007
Extend Industry, Extend OR and Extend Suite	Imagine That, Inc.	http://www.imaginethatinc.com/prods_modules.html	9th February 2007
Micro Saint Sharp Version 2.1	Micro Analysis & Design	http://www.maad.com/index.pl/micro_saint	10th February 2007
Visual Simulation Environment (VSE)	Orca Computer, Inc.	http://www.orcacomputer.com/vse/VSEBrochure/VSEBrochureSet.html	10th February 2007
Arena	Rockwell Automation	http://www.arenasimulation.com/products/professional_edition.asp	10th February 2007
eM-Plant	UGS	http://www.ugs.com/products/tecnomatix/docs/fs_tecnomatix_em_plant.pdf	10th February 2007
Vanguard Studio (DecisionPro)	Vanguard Software Corporation	http://www.vanguardsw.com/products/application-server/	11th February 2007
Vanguard Studio (DecisionPro)	Vanguard Software Corporation	http://www.vanguardsw.com/products/vanguard-studio/	11th February 2007
AnyLogic	XJ Technologies	http://www.xjtek.com/anylogic/why-purchase/	13th March 2007

Appendix A.6: Vendor URLs – support for sharing model**Table 49: Vendor URLs – support for sharing models in CSPs**

Software	Vendor	URL	Date Accessed
AnyLogic	XJ Technologies	http://www.xjtek.com/anylogic/beta6/features/	11th February 2007

Appendix A.7: Vendor URLs – distributed simulation support**Table 50: Vendor URLs - distributed simulation support in CSPs**

Software	Vendor	URL	Date Accessed
Arena	Rockwell Automation	http://www.arenasimulation.com/products/feature_matrix.asp	10th February 2007
AutoMod	Brooks Software	http://www.brookssoftware.com/download/27_disc_amo_d_1106.pdf	9th February 2007
Simprocess	CACI Products Company	http://www.caci.com/asl/simprocess_func_tech.shtml	9th February 2007

Appendix A.8: Vendor URLs – support for web-based simulation**Table 51: Vendor URLs – support for web-based simulation**

Software	Vendor	URL	Date Accessed
Quantitative Methods Software (QMS)	QuantMethods	http://www.quantmethods.com/FAQ.html	11th February 2007
MineSim™	Systemflow Simulations, Inc.	http://www.systemflow.com/minesim/index.html	11th February 2007
Vanguard Studio (DecisionPro)	Vanguard Software Corporation	http://www.vanguardsw.com/products/application-server/	11th February 2007
AnyLogic	XJ Technologies	http://www.xjtek.com/anylogic/features/	11th February 2007
AgenaRisk Enterprise Edition	AgenaRisk	http://www.agenarisk.com/newsletters/	9th February 2007
Witness	Lanner	http://www.lanner.com/en/simulation_professionals/witness_server.php	11th February 2007
Analytica	Lumina Decision Systems, Inc	http://www.lumina.com/ana/ADE.htm	11th February 2007
Vanguard Studio (DecisionPro)	Vanguard Software Corporation	http://www.vanguardsw.com/products/add-ins/web-services/	11th February 2007
Simprocess	CACI Products Company	http://www.simprocess.com/pdf/SOA-SimulationOnDemand-Simprocess.pdf	11th February 2007

APPENDIX B: NBS case study - further discussion

This appendix is intended to be read in conjunction with section 5.8 of this thesis. The NBS case study has investigated the CSP-specific distributed simulation service using DES CSP Simul8 Professional, High Level Architecture-Run Time Infrastructure (HLA-RTI) middleware for distributed simulation and enterprise desktop grid middleware WinGrid. The integration technology that has been used to integrate the three separate programs is referred to as the *WinGrid-DMSO_HLA_RTI-Simul8* integration architecture. It builds on the WinGrid-CSP integration architecture which is presented in section 4.4 of this thesis.

1. CSP Controller Middleware (CCM) architecture

The software component that has been developed to implement the *WinGrid-DMSO_HLA_RTI-Simul8* architecture is referred to as the *CSP Controller Middleware (CCM)*. The CCM has two separate implementations for the HLA-defined *Time Advance Request (TAR)* and *Next Event Request (NER)* mechanisms, which are used to request advancement of simulation time from the HLA-RTI. These implementations of CCM are referred to as CCM-TAR and CCM-NER respectively. The CCM has two distinct components, namely *Simul8 adapter* and *DMSO HLA-RTI adapter*, which interact with DES CSP Simul8 Professional and the DMSO HLA-RTI respectively. The architecture of the CSP Controller Middleware is shown in figure 54.

The Simul8 adapter defines methods like *OpenSim(modelFile)*, *RunSimulation(time)*, *getBloodOrdersFromHospital(hospital)* and *introduceEntitiesToHospital(hospital, bloodUnit)* that are invoked by the DMSO RTI adapter to open a Simul8 *modelFile*, run the model to the *time* specified, get blood orders from *hospital* and to introduce entities into the *hospital* respectively. These methods encapsulate both the application logic and the Simul8 COM method calls. For example, method *getBloodOrdersFromHospital(hospital)* has application logic that reads *hospital* order details being output by Simul8 into an Excel file and method *introduceEntitiesToHospital(hospital, bloodUnit)* invokes Simul8 COM method *ExecVL* to set various *bloodUnit* parameters into the running *hospital* model and to schedule events. The Simul8 adapter also calls methods defined in the DMSO RTI adapter like *tellSimulationTimeEnd(time)* and *sendOrderToNBS(hospital, bloodOrder)* to convey to the DMSO RTI adapter that Simul8 has completed processing a model till a defined “safe” *time* (see discussion below) and to transfer the *bloodOrder* collected from the *hospital*. The DMSO RTI adapter methods contain application logic and invoke HLA defined service calls. For example, the method *tellSimulationTimeEnd(time)* has application logic which sets the logical time of the federation to the *time* returned by the method call and *sendOrderToNBS(hospital, bloodOrder)* invokes HLA defined method *sendInteraction* to pass the *bloodOrder* details from respective *hospital* federates to the NBS PTI federate in the form of HLA interactions. It is

worthwhile here to mention that it is the RTI adapter that has separate federate logic for NER and TAR implementations (referred subsequently as CCM-NER and CCM-TAR).

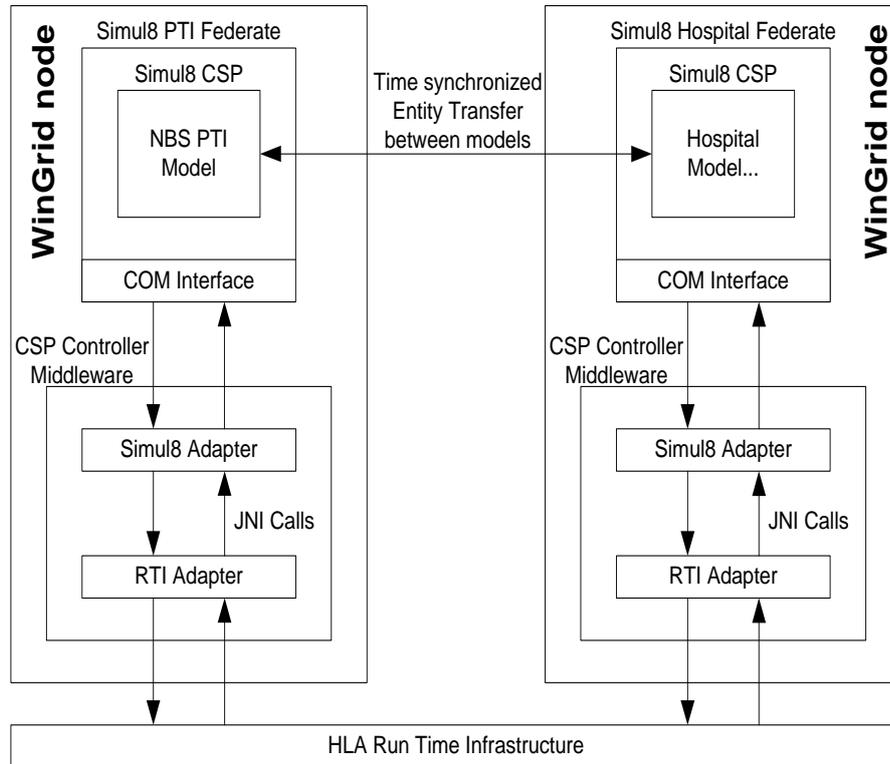
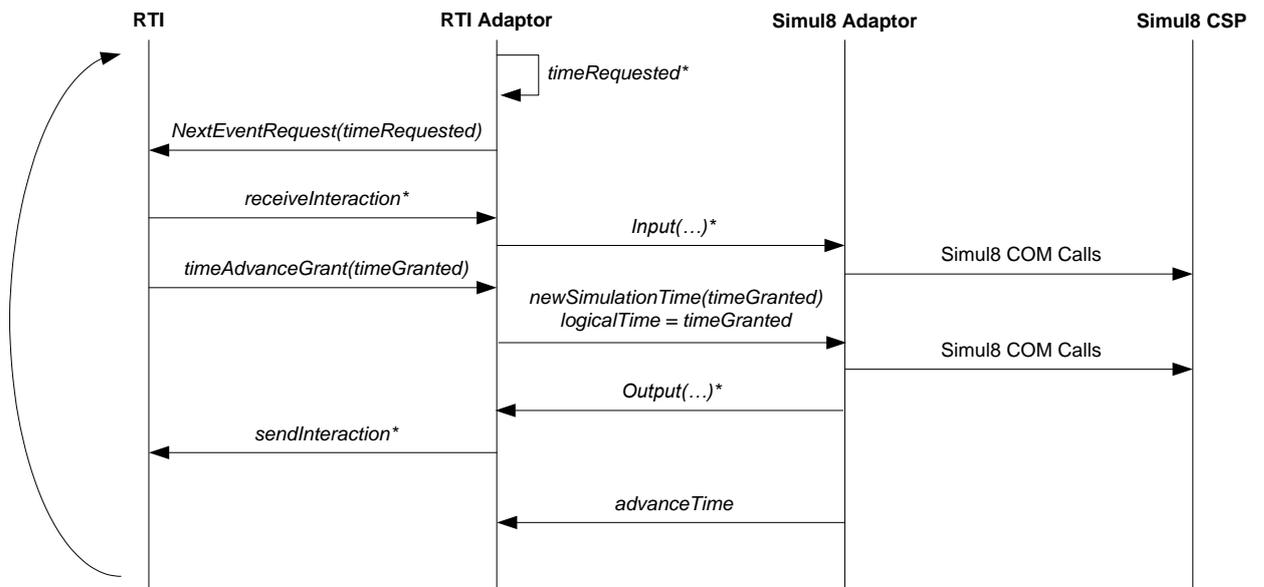


Figure 54: CSP Controller Middleware (CCM) architecture



NOTE: $timeRequested = logicalTime + 60$ (if, $logicalTime = timePreviouslyRequested$) OR $timeRequested = timePreviouslyRequested$ (if, $logicalTime < timePreviouslyRequested$)

Figure 55: CCM-Next Event Request (NER) protocol

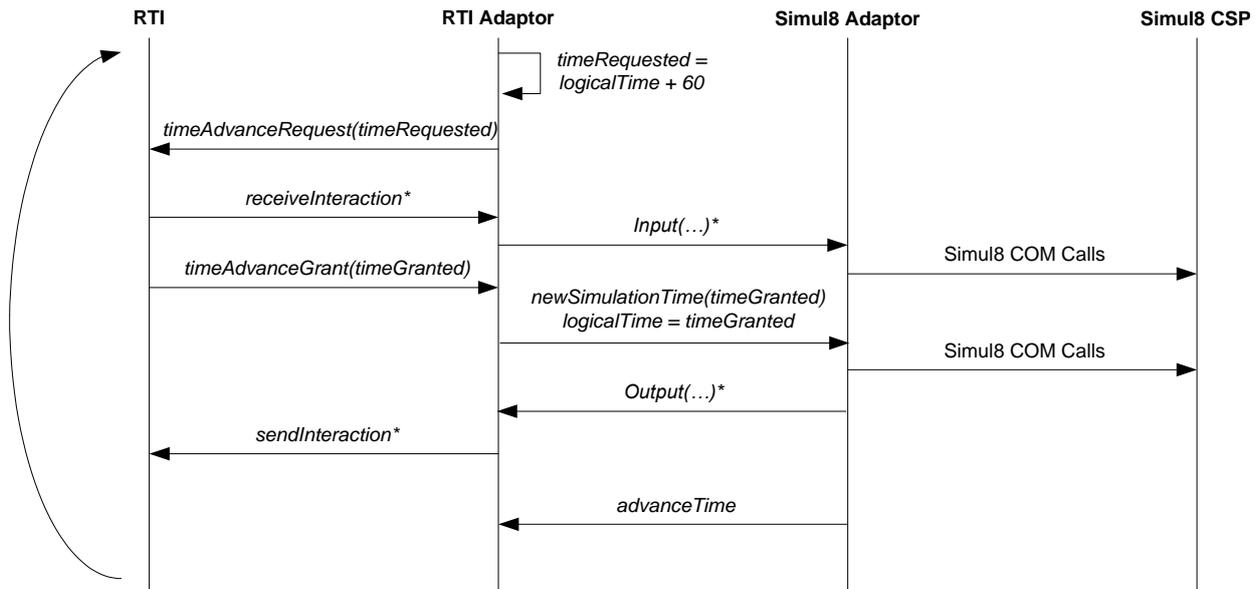


Figure 56: CCM-Time Advance Request (TAR) protocol

2. CCM-NER and CCM-TAR protocols

To introduce the CCM-NER and CCM-TAR protocols, discussions on HLA NER and HLA TAR time advance mechanisms are first presented. Both NER and TAR service calls, defined by the HLA standard and implemented by the HLA-RTI middleware, are invoked with a time component which represents the logical time the federate wishes to move to. Depending on whether NER or TAR is called by the simulating federate, the time granted to it by the RTI can be different. NER will grant the federate a time that is either less than or equal to the requested time depending on whether external events are present and if so, then their timestamps. If an external event exists for the federate with timestamp less than the requested time then the time granted by RTI will be equal to the timestamp of the external event. If no external events exist or an external event with timestamp equal to the requested time is received, then the RTI will grant the federate the requested time. TAR, on the other hand, will grant the simulation federate a time that is exactly equal to the time requested by a federate. The message exchange protocol followed by the CCM-TAR and CCM-NER variants of CCM are shown in figures 55 and 56 respectively.

CCM-NER invokes the HLA defined NER method call (*nextEventRequest[timeRequested]*) and CCM-TAR invokes the HLA defined TAR method call (*timeAdvanceRequest[timeRequested]*). Both these service calls have a time argument (*timeRequested*) that specifies the simulation time to which the federate wants to move to. The CCM-NER requests a time from the RTI that is equal to its current logical time + 60 (*timeRequested=logicaltime+60*) or a time that is equal to its previously requested time

($timeRequested = timePreviouslyRequested$) depending on whether the RTI had granted the $timeRequested$ by the federate in the preceding NER call or it had granted a time less than the $timeRequested$. CCM-TAR, on the other hand, requests a time from the RTI that is always equal to its current logical time + 60 ($timeRequested = logicalTime + 60$). The NBS PTI centre and the hospitals exchange information at every 60 units of simulation time and therefore both CCM-NER (in case $timeRequested$ had been granted in preceding HLA NER call) and CCM-TAR request a time equal to the current logical time of the federate + 60 simulation units. The difference with regards to $timeRequested$ by CCM-NER and CCM-TAR protocols is because they implement two different HLA synchronization strategies, viz. NER and TAR.

In case of both CCM-TAR and CCM-NER, the new time granted to the federate by the RTI is conveyed using HLA TIME ADVANCE GRANT callback ($timeAdvanceGrant[timeGranted]$). This callback, invoked by the RTI on the federate RTI adapter, carries the time ($timeGranted$) that has been granted by the RTI and is a guarantee that there will be no external events from the rest of the federation before this time. This new “safe” time is conveyed by the RTI adapter to the Simul8 adapter ($newSimulationTime[timeGranted]$) and the simulating federate processes the Simul8 model to this time. This may, in turn, generate other internal or external events. Subsequently, the logical time of the federate becomes equal to this new time ($logicalTime = timeGranted$) and the process of requesting time advancement using NER or TAR starts all over again.

This discussion now looks at how external events are sent across federates in the NBS simulation. HLA interactions are used to achieve this. Interactions are an HLA defined transport mechanism for intra-federation communication (i.e., communication between the running models that together form the distributed simulation). When a federate generates an external event the Simul8 adapter of CCM conveys this to the DMSO RTI adapter, which in turn invokes the HLA defined service SEND INTERACTION ($sendInteraction^*$). Each interaction contains a time stamp and associated data. These interactions are sent to the RTI to be delivered to the respective federates in the causally correct order. On the receiving end, the RTI delivers the interactions to the DMSO RTI adapter through the RTI callback RECEIVE INTERACTION ($receiveInteraction^*$). The DMSO RTI adapter of the CCM then forwards the received data to the Simul8 adapter for introduction into the model. The data being exchanged in the federation relate to blood orders and deliveries. In both $sendInteraction^*$ and $receiveInteraction^*$, the superscript “*” indicates that multiple interactions can be sent or received.

3. Experiments

To investigate the performance of NBS standalone simulation with (1) NBS distributed simulation over WinGrid using NER time management service (implemented by CCM-NER) and, (2) NBS distributed simulation over WinGrid using TAR time management service

(implemented by CCM-TAR), four different scenarios were designed. Each scenario was represented by one NBS PTI centre serving one, two, three or four hospitals respectively. The name of the scenario reflects the number of hospitals that the NBS PTI caters for. For example, scenario *2Hospital* implies that 2 hospitals are being served by one NBS PTI centre. In case of distributed NBS simulation, scenario *2Hospital* implies three separate Simul8 models, each modelling either the NBS PTI centre, Hospital1 or Hospital2 and running on three separate WinGrid nodes. In case of standalone NBS simulation, scenario *2Hospital* suggests that a single Simul8 model, running on a single PC, has modelled the behaviour of the NBS PTI centre and two hospitals.

The results of the experiments have already presented in section 5.8.7 of this thesis. Graph one shows the time taken to execute the standalone and the distributed versions of the NBS simulation. Graph two shows the monthly execution time of the NBS standalone and distributed simulations.

4. Discussion

From the results the following observations can be made:

- (A) For scenarios *1Hospital* and *2Hospital* the standalone NBS simulation executes faster than its distributed counterparts and for scenarios *3Hospital* and *4Hospital* the distributed versions out perform the conventional simulation.
- (B) Comparing the performance of the distributed versions we see that for each consecutive month of the year and for each of the four scenarios (except months 2 and 7 in scenario *4Hospital*), the simulation using TAR time management executes between 3.5-23.9% faster than its NER counterpart (see table 52 below).
- (C) The average performance gain by using TAR over NER for scenarios *1Hospital*, *2Hospital*, *3Hospital* and *4Hospital* is approximately 13.7%, 21%, 19% and 6% respectively.

Table 52: Percentage performance increase of TAR over NER

Performance gain of TAR over NER (%)	Scenario <i>1Hospital</i>	Scenario <i>2Hospital</i>	Scenario <i>3Hospital</i>	Scenario <i>4Hospital</i>
1 month	16.84	23.01	20.19	5.62
2 months	13.12	21.13	17.68	-7.89
3 months	12.83	21.88	19.28	6.46
4 months	15.19	22.48	18.97	11.73
5 months	14.33	20.92	19.81	8.56
6 months	13.12	19.40	17.07	6.82
7 months	11.76	20.40	18.98	-2.28
8 months	13.59	20.32	18.99	8.98
9 months	15.17	21.35	18.22	15.28
10 months	14.15	21.61	17.69	8.87
11 months	13.86	20.21	23.86	7.35
12 months	10.94	19.72	17.40	3.54
Average performance gain (%)	13.74	21.04	19.01	6.09

The implications of these observations are now considered.

A. Comparing Standalone and Distributed Implementations

By applying the principles of distributed simulation and the HLA, the time taken to execute the NBS simulation is reduced significantly when the model becomes larger. When compared with the conventional NBS model, both the distributed versions recorded a negative performance improvement for scenarios *1Hospital* and *2Hospital*. However, as more complicated models were introduced in scenarios *3Hospital* and *4Hospital* the distributed models executed faster compared to their standalone counterparts. The results (section 5.8.7) show that the conventional model with one hospital takes approximately 14 minutes to run for a whole simulated year. The run time rises to 78 minutes when the model runs with two hospitals and to approximately 17.5 hours with three hospitals. The addition of the fourth hospital increases the execution time to 35.8 hours. The NER version of the distributed model with one NBS supply centre and one hospital runs in approximately 8.4 hours, with two hospitals in 9.8 hours, with three hospitals in 12.7 hours and with four hospitals in 16.5 hours. The execution time for the TAR version of the distributed model is 7.2, 7.8, 10.3 and 15.5 hours for the *1Hospital*, *2Hospital*, *3Hospital* and *4Hospital* scenarios respectively.

These findings indicate that for the conventional method an expansion in model size will be accompanied by an increase in the total runtime. On the other hand, for the distributed methods an increase in the number of hospitals (and therefore of computers) will be followed by a much smaller increase in total runtime. Therefore, if more than two hospitals are added to any model, the distributed method would be a better platform in which to develop and run the simulation experiments. Overall, the distinctive trend that the two methods follow concerning runtimes seems to be continuous; in other words, the more hospitals that are added to the model, the more the differences in the runtimes between the two methods favour the distributed approach. The increase in runtime appears to be primarily due to a large event list caused by a combination of the volume of entities and the “counting down” of the shelf life of blood products in minutes. The large event list in turn possibly causes swapping between RAM and virtual memory which further causes long runtimes. The results suggest that the distributed approach allows the processing and memory demands made by large event lists to be shared over several computers. Note that eliminating the “counting down” model feature with a different approach to blood product shelf life would most likely give an increase in performance. However, this would invalidate the model.

It may be argued that a machine with more processing power and with more RAM (compared to the 1.73GHz processor and 1GB RAM laptops that were used for the NBS experiments) could execute the standalone *3Hospital* and *4Hospital* scenarios of NBS model much faster, such that it outperforms its distributed *3Hospital* and *4Hospital* counterparts. Thus the negative performance improvement recorded by using the distributed models, as against using the conventional standalone models, for scenario *1Hospital* and *2Hospital* may also

occur in scenarios *3Hospital* and *4Hospital* through the use of better hardware. This would possibly make the distributed simulation infeasible.

Although there is some merit to this line of reasoning, two specific arguments are presented to show the feasibility of using the distributed approach.

First argument: Having more CPUs and more memory does always equate to faster performance. This is especially true in case of machines having multiple CPUs (Dual-Core and Quad-Core processors) or machines that have CPUs with Hyper-Threading Technology (HTT) enabled. HTT is a new CPU technology and more elaboration is necessary for further discussion later in this section. HTT makes a single physical processor appear as two logical processors, wherein the physical execution resources are shared and the architecture state is duplicated for the two logical processors (Marr et al., 2002). The operating system treats a hyper-threaded CPU as two processors instead of one and a program can schedule processes or threads on both the logical processors and the CPU will execute them simultaneously, as if there were two physical processors present in the system.

One important factor that determines that a program executes faster on a higher configuration machine is that the program itself has been implemented to make the best possible use of all the available hardware in the system. Thus, it differs according to package implementation. To test whether Simul8 gains from an even higher configuration machine, the *4Hospital* scenario was experimented on a standalone PC having 2GB RAM and 3.2GHz Hyper-threaded Pentium 4 CPU. The time taken to run the simulation was around 38 hours (the time taken to execute the *4Hospital* scenario on a laptop having 1GB RAM and 1.73GHz Intel Celeron processor was around 35.8 hours). The same *4Hospital* model was run on an even higher configuration machine to examine whether Simul8 would gain from using a computer with two dual core 2.8GHz processor (i.e., four processors) with 12GB RAM. In this case, the time taken to execute the simulation took even longer (approx. 42 hours).

Thus, the execution time was not reduced by using more hardware. One of the reasons for this is that most of the processing in Simul8 takes place on one main thread that makes use of one “logical” processor (in case HTT is enabled) or one “physical” processor (in case of Dual-Core and Quad-Core machines). Thus, it can be argued that for a CSP to utilize additional hardware effectively, the CSP vendor may have to modify the program itself. A distributed approach to CSP simulation may alleviate the need for such technology-specific changes.

Multiple processors in a system are a reality that program developers may have to face sooner than later for the following reason. Moore’s law states that the number of transistors on a chip, or transistor density, doubles every 24 months. However, as transistor size

decreases and transistor density and computing power increases, the heat generated by the chip becomes a major problem and multi-core processors become important (Quinn et al., 2005). Consequently, the major chip manufacturers are now looking at doubling CPU performance by increasing the number of CPU cores, as against doubling the clock-speed of a single CPU. Until the time a CSP is implemented to utilize multiple CPU-cores, distributed simulation of very large and complex models may remain feasible. Furthermore, the performance gains which can be expected by implementing multiple-processor friendly CSPs need to be investigated. Issues such as the division of the execution of a single instance of the simulation executive onto two processors, distributing the event list over multiple CPUs, etc. can be difficult and may require some synchronization of its own. As is the case with distributed simulation, to achieve this synchronization some overheads may be generated. Thus, whether standalone, multiple-processor CSP implementation outperforms distributed, single-processor CSP implementation, or vice-versa, is a question which requires further investigation.

Second argument: The second argument on the feasibility of distributed simulation for modelling large CSP-based supply chain models is that it can provide an alternative to single computer CSP simulation, in cases where the model to be simulated is so large and complex that its execution cannot be completed in acceptable time even on the fastest machine available for commercial purchase. In such cases, self-federating an existing CSP simulation by dividing the model between multiple computers can help reduce run time.

B. Comparing NER and TAR

The distributed simulation using TAR time management service call performs better because the discrete-event NBS simulation is modelled to exchange information at constant intervals of simulation time (the NBS PTI centre and the hospitals exchange information at every 60 units of simulation time). Thus, it is possible to treat the NBS simulation as a time-stepped simulation in the distributed sense and use TAR to request RTI for a time advance equal to current logical time + 60 units of simulation time.

Using NER time management introduces the overhead of an extra *NextEventRequest* service call being made by a federate (and the resultant invocation of *TimeAdvanceGrant* callback by the RTI) whenever an interaction is received. Figures 55 and 56 outline the protocols followed by NER and TAR versions of the CSP controller middleware (CCM) respectively.

The CCM-NER protocol represented in figure 55 shows that when a time-constrained federate (a federate that receives timestamped messages from other federates) and time-regulating federate (a federate that sends timestamped messages to other federates) is in time granted state (see figure 57 below), the DMSO RTI Adapter of the CCM requests time

advance (*timeRequested*) equal to either, (1) its *logicaltime* + 60, or (2) its previous time request (*timePreviouslyRequested*). (1) is used if the federate had received a *timeAdvanceGrant* equal to *timeRequested* during the preceding time advancing state. In short, if *timeGranted* = *timePreviouslyRequested* then *timeRequested* for the next NER call will be *logicaltime* + 60. This happens when no time stamped order (TSO) interactions are received by the federate during the time advancing stage. However, if an interaction is received then *timeGranted* by RTI will be equal to the timestamp of the interaction and *timeGranted* will be less than *timePreviouslyRequested*. As the simulation executes in equal timesteps, viz, 60, 120, 180, therefore *timeRequested* for the next NER call will be *timePreviouslyRequested* (but which was not granted by RTI). Since the *logicaltime* of the federate will be equal to *timeGranted* by RTI through the *timeAdvanceGrant* callback, we can also say that (a) if *logicaltime* = *timePreviouslyRequested* then *timeRequested* for the next NER call will be *logicaltime* + 60, and (b) *logicaltime* < *timePreviouslyRequested* then *timeRequested* for the next NER call will be *timePreviouslyRequested*.

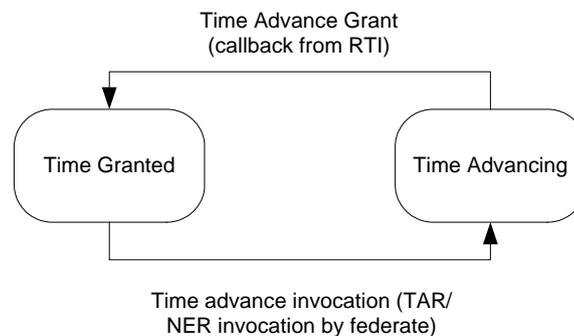


Figure 57: Time Management States of a Federate (adapted from Kuhl et al., 1999)

As previously discussed, the CCM-TAR protocol represented in figure 56 is different because the DMSO RTI Adapter of the CCM always requests a time equal to its *logicaltime* + 60 when invoking the next TAR request, irrespective of whether the federate has received an interaction in the preceding time advancing state. In this case the *timeGranted* returned by RTI through the *timeAdvanceGrant* callback will always be equal to *timePreviouslyRequested*. Any TSO interactions are delivered to the federate before the *timeAdvanceGrant* callback. Thus, using TAR time management mechanism in the NBS distributed simulation saves one redundant message exchange between the federate and the RTI whenever the federate receives an interaction.

C. Analyzing Performance Gains Achieved by Using TAR over NER

To further examine the performance gain achieved by using TAR over NER and to investigate its gradual drop (from approx. 21% in scenario *2Hospital* to approx. 6% in scenario *4Hospital*), a discussion relating to the interactions being sent across the NBS federation is presented below. As has been said earlier, the discrete-event NBS model can be perceived

as a time-stepped simulation because the exchange of information between federates take place every 60 units of simulation time. The orders generated in the hospitals between two distinct time steps (say, 60 and 120) are kept in buffer and only released to the NBS PTI model in the subsequent time step (120 in this case). Similar is the case with NBS PTI model. The successfully match blood units are kept ready for delivery but not released to the hospitals until the next time step. In HLA-based simulation, a time-regulating federate in a time granted state can send interactions with any timestamp at least equal to its logical time + its *lookahead*. A lookahead value, expressed in terms of simulation time units, places a restriction on the time-regulating federate; if the federate is at a logical time t and has a lookahead value l , the RTI will not allow it to send timestamped messages with time less than $t+l$ (Kuhl et al., 1999). The NBS models operate with a look ahead of 1 unit of simulation time. Thus, at time 120 the hospitals send interactions to NBS PTI with a time stamp of 121. These interactions carry order information specifying the requirement of blood. Similarly, the NBS PTI delivers interactions to the different hospitals at time 121 to inform the respective hospitals of the quantity of blood delivered along with a host of attributes.

The timestamp of the interactions received by a federate in time advancing stage are important. To find out why, the previous example is extended and it is supposed that at logical time 120, *hospital1*, *hospital2* and *hospital3* send requests for blood. The timestamp of the interactions being sent to NBS PTI will be 121. The NBS PTI receives all the interactions in the time advancing stage when it requests the RTI to advance its simulation time to 180. The messages that the federate exchanges with RTI to reach logical time 180 will depend upon the time management service being used.

1. TAR: RTI delivers all three TSO interactions through *receiveInteraction* callback and then grants time 180 through *timeAdvanceGrant* callback. The logical time of the federate is therefore 180.
2. NER: RTI delivers the three TSO interactions to the NBS PTI federate using *receiveInteraction* callback. The RTI will then grant time 121 through *timeAdvanceGrant* and the federate will reach time granted state. The federate will then request time 180 from the RTI and in this occasion the time advance will be granted to 180. This is because communication between federates can only take place at constant intervals of time. At time 120, the set of orders were already released by the hospitals with a timestamp 121. If orders are generated between 120 and 180 they would be released when the hospitals are in the time granted state at logical time 180. The timestamp of the interaction for the next set of orders will be 181.

The above discussion shows that a NER federate in the NBS simulation generates a maximum of one extra pair of federate-RTI communication (when compared to a TAR federate) for every 60 units of simulation time, irrespective of the number of interactions it

receives. In the example above, the NBS PTI federate received three interactions with timestamp 121 but generated only one extra NER call and received subsequent callback. The NBS simulation was run for 524160 simulated minutes. Therefore, the total number of extra federate-RTI communication that could be generated is 8736 ($524160 / 60$) for each NER federate. The actual number is much less since orders are not placed every hour by the hospitals and the NBS PTI delivers blood at pre-defined times (except for emergency cases).

From the discussions above it seems likely that the drop of average performance gain by using TAR over NER (from approx. 21% in case of scenario *2Hospital* to approx. 19% in scenario *3Hospital* and again to approx. 6% in scenario *4Hospital*) cannot be attributed to an increased number of extra federate-RTI communications taking place as the number of hospitals are increased. As discussed above, when the number of hospitals increase from 3 to 4, for example, the NBS PTI federate may receive a maximum of 4 interactions (one from each hospital placing an order). However, since the time stamps of the interactions received will be the same therefore the NER generates only one extra pair of federate-RTI communication in the form of one NER call and the subsequent callback received from RTI.

It seems likely that the drop in performance is because the NBS PTI model grows more complicated as it starts serving more hospitals. The process of finding a match between hospital orders and present blood stocks itself is complicated. As the number of hospitals increase this process has to be repeated for orders for each hospital. The time gained by applying TAR time management mechanism is primarily because of the reduction of messages between federates. But as the NBS PTI model becomes more complex it takes longer to execute it and this slowly erodes the time gained through reduction of messages brought about through the application of TAR. A solution to this could be to divide the NBS PTI centre into two or more separate models. However, this would require revalidation of the model.

5. Conclusion

Using multiple sets of experiment results it has been shown that a Simul8-DMSO RTI distributed simulation will run faster than its standalone counterpart when the model has reached sufficient size. Thus, for the NBS model, distributed simulation appears to offer a viable alternative to conventional simulation by sharing the processing and memory requirements of the simulation across multiple computers. Since two specific software applications have been used for this study (namely, Simul8 and DMSO RTI 1.3NG), it is difficult to generalize the findings to encompass the entire range of CSPs and RTIs available today.

It has been further argued that the selection of an appropriate conservative time advance mechanism (NER or TAR) in HLA-based distributed simulation should be made not only

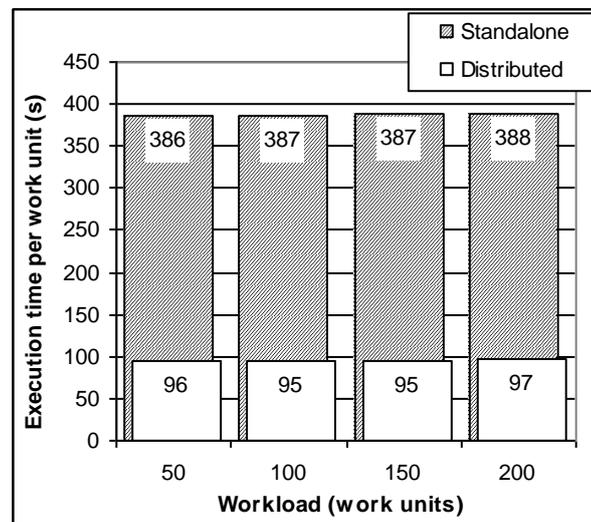
based on the internal characteristics of the simulation, but consideration should also be given to the characteristics of the message flow between models. As has been shown in the case of NBS distributed simulation, a HLA federation comprising of DES federates (i.e., each federate simulates a discrete-event model), designed to exchange messages only at constant intervals of time, can be considered as a time-stepped simulation in the distributed sense. Thus, using TAR time management service call is more appropriate in this case as compared to using NER.

APPENDIX C: BOINC case study - experiments and results

This appendix is intended to be read in conjunction with section 5.4 of this thesis. The BOINC case study investigates the BOINC middleware in relation to CSP-specific SMMD task farming service. The Range Accrual Swap application (section 5.4.2) based on MCS CSP Excel is used together with BOINC middleware to experimentally evaluate whether the CSP-grid solution is implementable in practice.

To experiment with BOINC and the RAS application, 200 work units were created on the BOINC server side by running a java program which invoked the BOINC *create_work* command (section 5.4.3). The experiment consisted of timing the execution of 50, 100, 150 and 200 work units of the Excel-based RAS financial model. The time taken to execute the distributed BOINC implementation over eight computers was compared to the standalone execution of the RAS model on a laptop equipped with a 1.73GHz Intel Celeron processor and 1GB RAM.

The results are summarized in graph 8 below. It shows execution time per work unit, averaged over five separate runs of the experiment. However, this graph only includes experiments using the four BOINC clients running over the laptops for reasons outlined below.



Graph 8: RAS application results

The graph shows that the speedup is approximately linear compared to standalone execution for the range of workloads that were tested. This was expected for several reasons: client computers were entirely dedicated to running the simulation; work units carried little data due to the nature of the simulation; the BOINC client pre-fetched new work units from the server so that it may continue uninterrupted. Under these circumstances BOINC imposed very little overhead.

Pre-fetching of new work units by the BOINC client has both a positive and negative impact on the operation of the system. By setting the work unit request interval sufficiently short, the client ensures that it has work units in hand before the work unit currently being executed has completed. However, when client computers of differing performance specifications are used on the same application, a phenomenon was observed that has been termed as “job hoarding”.

Essentially job hoarding occurs because the BOINC system currently provides no fine control over how many work units are pre-fetched by each client, and thus “fast” clients and “slow” clients both pre-fetch multiple work units. If the work units are relatively large-grained, the fast clients may complete execution of all their work units before the slow clients have finished processing the first of their work units.

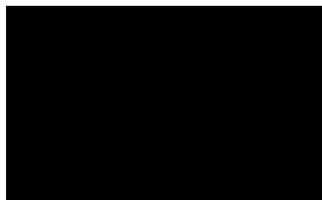
In the BOINC-RAS experiments, the faster laptops completed around 95% of the total workload and became idle before the first work units had been completed by the slower desktop computers. At this point the desktop machines were each hoarding further work units which the laptops could not access, and the initial results showed a total distributed execution time far in excess to the time taken to execute the standalone RAS application over a high specification laptop. Thus, measurements from only the four laptops were taken until the hoarding effect could be investigated in more detail.

APPENDIX D: WinGrid user documentation (version 1.0)

WinGrid 0.2 User Documentation

**WinGrid-Excel™-Analytics™ Integration for Speeding up IRS and
RBF Simulations at [REDACTED]**

Document Version 1.0



Author and Systems Developer:

Navonil Mustafee
Research Student
Centre for Applied Simulation Modelling (CASM)
School of Information Systems, Computing & Mathematics
Brunel University, Uxbridge, Middlesex UB8 3PH

Index:

1. WINGRID – THE DESKTOP GRID FOR WINDOWS	267
1.1. WinGrid and Master-Workers Model	
1.2. WinGrid Components	
1.3. WinGrid integration with Microsoft Excel™ and Analytics™	
2. RUNNING WINGRID ON YOUR PC	268
2.1. WinGrid Software Dependencies	
2.2. Drive Mapping	
2.3. Register Dynamic Link Library (DLL)	
2.4. Execute Batch File	
2.5. Mapping Drive, Installing JACOB + JDIC, Setting CLASSPATH, Registering DLL and Including a new Computer as part of WinGrid Computation Infrastructure	
3. WTC – THE WINGRID THIN CLIENT	277
3.1. WinGrid Thin Client (WTC) Arguments	
3.2. WTC_DEBUG Batch File	
3.3. WTC Batch File	
3.4. WinGrid Thin Client (WTC) Status	
3.5. WinGrid Thin Client (WTC) Menu	
4. WJD – The WINGRID JOB DISPATCHER	282
4.1. Configuring Parameter Files	
4.2. Configuring WinGrid Job Dispatcher (WJD) Parameter File	
4.3. Configuring WinGrid Job Dispatcher (WJD) Application Specific Parameter File	
4.4. WinGrid Job Dispatcher (WJD) Log File Directory	
4.5. WinGrid Job Dispatcher (WJD) Execution	
4.6. WinGrid Job Dispatcher (WJD) Execution Completion	
5. PERFORMANCE RESULT COLLECTION	296
6. WTC AND WJD ERRORS	298
6.1. WinGrid Thin Client (WTC) Errors	
6.2. WinGrid Job Dispatcher (WJD) Errors	
7. MISCELLANEOUS	303
7.1. Acknowledgements	
7.2. Contact Information	

1 WINGRID – THE DESKTOP GRID FOR WINDOWS

WinGrid is a program that utilizes multiple PCs over a network to perform computation intensive jobs. The time taken to execute these jobs is inversely proportional to the number of computers running the WinGrid software. The software creates a computation infrastructure by pooling together multiple workstations (nodes) and using the processor of each such workstation to execute a part of the job. Coordination among the different nodes is maintained through exchange of protocol messages.

1.1 WinGrid and Master-Workers Model

WinGrid implements the push-model of the Master-Workers distributed computing architecture. In this architecture you have one Master program (think of this as your line manager) running on one single computer that continuously monitors multiple Worker programs (yourself and your colleagues) running on separate computers. All jobs to be executed are with the Master program. You can think of jobs as computationally intensive tasks like, say, adding 100 million randomly generated numbers. The Master program shares this workload among several Worker programs, sending each a subset of the calculations to perform. Each Worker program performs its part of the computation and sends the Master the result. Finally, the Master program has to assimilate all the results returned to it by the Workers to present the final figure.

The computer that runs the Master process (process and program mean the same in our context) can also execute some of the jobs. This it does by starting the Worker process alongside the Master process. The Master process in WinGrid sends only one job to each Worker for processing at any one time (your ideal line manager!). The Master process can be started on any computer, but please remember that only one computer can run this process at any given time. The Worker process, of course, will have to be run on multiple computers for the distributed-run to execute faster than its sequential counterpart.

1.2 WinGrid Components

The Master process of WinGrid is called the *WinGrid Job Dispatcher (WJD)* and the Worker process is called the *WinGrid Thin Client (WTC)*. The more the number of WTCs in the network (read as, the more the number of computers in the network which have the WinGrid Thin Client program running) the faster will be the execution of jobs.

1.3 WinGrid Integration with Microsoft Excel™ and Analytics™

WinGrid is a multi-purpose program and can be used with different applications. At XXX, WinGrid is used with Microsoft Excel™ spreadsheets. This spreadsheet, in turn, invokes Analytics™ for computation of complex risk calculations. The WinGrid software integrates with Microsoft Excel™ on both the Master side and the Worker side using integrated program code. We call this integrated code the *WinGrid Master Application* and the *WinGrid Worker Application* respectively.

The WJD (WinGrid Master Process) calls the *WinGrid Master Application* to gather details of the jobs that have to be processed. The WTC (WinGrid Worker process), on the other hand, calls the *WinGrid Worker Application* to process the jobs which are sent by the WJD. For our XXX application jobs are different currencies (GBP, USD, INR etc) that serve as inputs for calculations of *Interest Rate Swaps (IRS)* and the *Risky Bond Forwards (RBF)*.

A basic architecture of WJD and WTC and how they communicate with each other is presented in Figure 1 below.

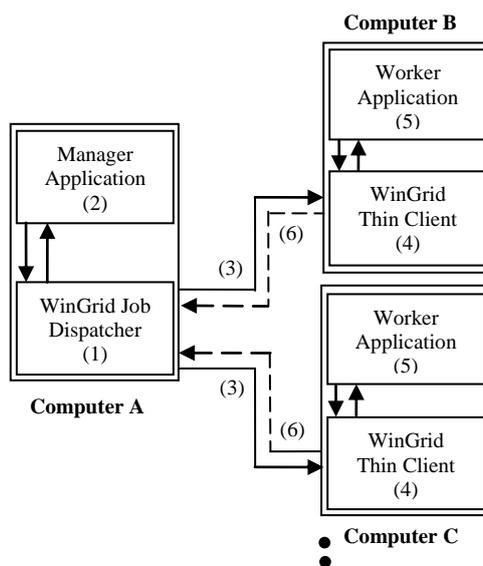


Figure 1: WinGrid Architecture

The user submits a job by running the WJD process (1) on Computer A. The WJD then interacts with the integrated WinGrid Master Application (2) to extract job details. WJD then divides the work into individual work units and sends them for processing (3) to the WTC processes (4) running on Computers B and C. The dots represent that the system can handle more than 2 computers. The WTC pass this work to their WinGrid Worker Application for processing (5) and returns the result to the WJD (6). The results of all the sub jobs are collated and communicated back to the WinGrid Master Application for presentation to the user.

Running only the WJD process on a computer is not compute intensive. This is because most of the processing is done by the WTCs by invoking Analytics™ through Excel™. However, after the WJD has received all the results from the WTCs the WJD has to assimilate these individual results together. During this stage (which roughly takes 1% or less of the total computation time) the WJD will require some CPU time and the computer may appear to become less responsive.

2. RUNNING WINGRID ON YOUR PC

To run WinGrid on your desktop please check whether you have the required software installed on you PC (section 2.1), map the drive (section 2.2), register a dynamic link library (section 2.3) and execute the batch file (section 2.4).

- Section 2.1: Checking software dependencies and installing software is only a one time process.
- Section 2.2: You have to map your drive to the shared location of WinGrid files. This is only a one time process.
- Section 2.3: Registering the DLL is a one time process.
- Section 2.4: You have to execute a batch file whenever you want to run the WJD and WTC processes. A batch file contains a series of commands that is intended to be executed by the Operating System. There are two separate batch files to invoke for WJD and WTC respectively.

2.1 WinGrid Software Dependencies

The term WinGrid software dependencies mean the programs and/or software libraries that have to be installed on your computer for successfully running WinGrid.

2.1.1. Java Runtime Environment (for both WJD and WTC processes)

WinGrid is written in Java programming language. Unlike a program written in C or C++, a Java program is dependent on another program called the *Java Runtime Environment (JRE)* for execution. To find out whether you have JRE installed on your desktop follow the following steps:

- (1) Go to *Start* → *Run*
- (2) On the Run dialogue box enter *cmd*. This will open up a program called *Command Prompt*.
- (3) Now type *java* at the *C:\>* prompt and press enter (The prompt can be other than *C:\>*).

If JRE is present then it will output the following message:

```
C:\Documents and Settings\Admin>java
Usage: java [-options] class [args...]
           (to execute a class)
   or  java [-options] -jar jarfile [args...]
           (to execute a jar file)

where options include:
   -client    to select the "client" VM
   -server    to select the "server" VM
   -hotspot   is a synonym for the "client" VM [deprecated]
   The default VM is client.
```

Screenshot 1: Output from java command

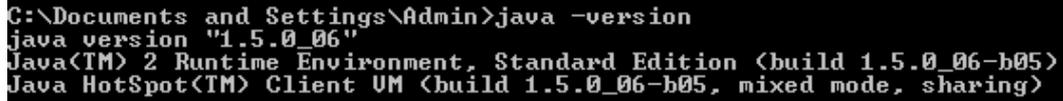
If JRE is not present then you will be echoed back with the following message:
'java' is not recognized as an internal or external command, operable program or batch file

In this case you will need to contact the computer support to install the JRE program on your computer.

The version of JRE required to run WinGrid is *version 1.5.0* or later. If you have JRE installed then you can find the version number by entering the following command in the Command Prompt program.

```
C:\> Java -version
```

You are prompted back with a version number similar to the following:



```
C:\Documents and Settings\Admin>java -version
java version "1.5.0_06"
Java(TM) 2 Runtime Environment, Standard Edition (build 1.5.0_06-b05)
Java HotSpot(TM) Client VM (build 1.5.0_06-b05, mixed mode, sharing)
```

Screenshot 2: Output from java -version command

The version number returned in the example above indicates that WinGrid can run on the machine as the version of JRE installed is higher than 1.5 (it is actually build 0_02). In case you are echoed back a version less than 1.5 then please contact the computer support to get necessary updates.

Further information: JRE can be downloaded from the following URL: <http://www.java.com/en/download/index.jsp>. This software can be downloaded for free.

2.1.2 JACOB library (for both WJD and WTC processes)

You need a JAR file (Java Archive File) called *jacob.jar* (version 1.9 or later). If *jacob.jar* is present on your machine then the location of this file should appear in the CLASSPATH variable. The CLASSPATH variable is used by JRE (see above) to find certain JAVA libraries required for executing an application. Follow these steps to see the value of your CLASSPATH variable.

- (1) Go to *Start* → *Setting* → *Control Panel* → *System*
- (2) Click on *Advanced* Tab and then click the *Environment Variables* button.
- (3) This opens up another window called *Environment Variables* which has two specific list boxes – namely, *user variables* and *system variables*.
- (4) Scroll down both these list boxes and look for a variable called CLASSPATH.
- (5) If you find the CLASSPATH variable the double click it to check its value.
- (6) The *Edit User Variable* Window that opens up has two text boxes called *variable name* and *variable value* respectively. You must now check the variable value text box and try to locate whether you have a path (the location of a file with respect to directories) that ends with *jacob.jar*. If yes, then you have the required JACOB library.

Please contact the computer support in case you do not have the CLASSPATH variable set or do not have the path to *jacob.jar* in you CLASSPATH.

Further Information: Refer to section 2.5.

2.1.3 JDIC library (for both WJD and WTC processes)

You will need *jdic.jar* and *jdic.dll* to run WinGrid. You can find whether you have *jdic.jar* by following steps 1 to 6 outlined in section 2.1.2 above. The only difference now is that you are looking for *jdic.jar* in place of *jacob.jar* in your CLASSPATH variable.

You will find *jdic.dll* in your *C:\Windows\System32* directory.

Please contact the computer support in case you do not have the CLASSPATH variable set or do not have the path to *jdic.jar* in your CLASSPATH or *Jdic.dll* is missing from your *C:\Windows\System32* directory.

Further Information: Refer to section 2.5.

2.1.4 Microsoft Excel™ (for both WJD and WTC processes)

The WinGrid Master Application (WMA) interacts with Microsoft Excel™ to extract job details. The file that WMA reads is called the *WinGridApplicationSpecific.properties.xls* file. The WJD also uses Microsoft Excel™ to extract certain parameters required for creating the WinGrid computing infrastructure (for example, parameters such as the addresses of the computers running WTC are read from *WinGridJobDispatcher.properties.xls* file). It thus follows that the computer which will run WJD (and the integrated WMA code) should be installed with Microsoft Excel™.

Excel™ must also be installed on the computers running WTC because the WinGrid Worker Application (WWA) interacts with Microsoft Excel™ to start processing the work units.

2.1.5 Analytics™ (for WTC process only)

The WinGrid Worker Application (WWA) interacts with Analytics™, via Microsoft Excel™, to process jobs sent to it by the WTD. Thus the computer on which WTC will run should be installed with both Microsoft Excel™ and Analytics™. You can find whether Analytics™ is installed in your machine from the start menu.

2.2 Drive Mapping

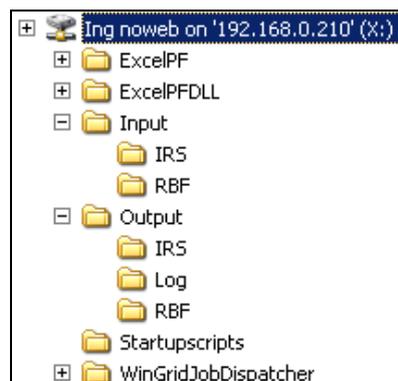
All the WinGrid software is stored in a shared directory. Each computer has to map this shared directory as **X:** drive and check the “Reconnect at Logon” checkbox (see the screenshot 3). Using a shared directory helps to rapidly deploy newer versions of the software because the updated files are required to be placed only in one shared location. The XXX shared directory to map X: is:

```
\\SPGBFAP20004\SharedData1\Group  
Data\GRGB001702\Projects\Brunel_Grid_Computing\WinGridCommonFiles
```



Screenshot 3: Map Network Drive dialogue box

On successful mapping the following directory structure will appear in Windows Explorer.



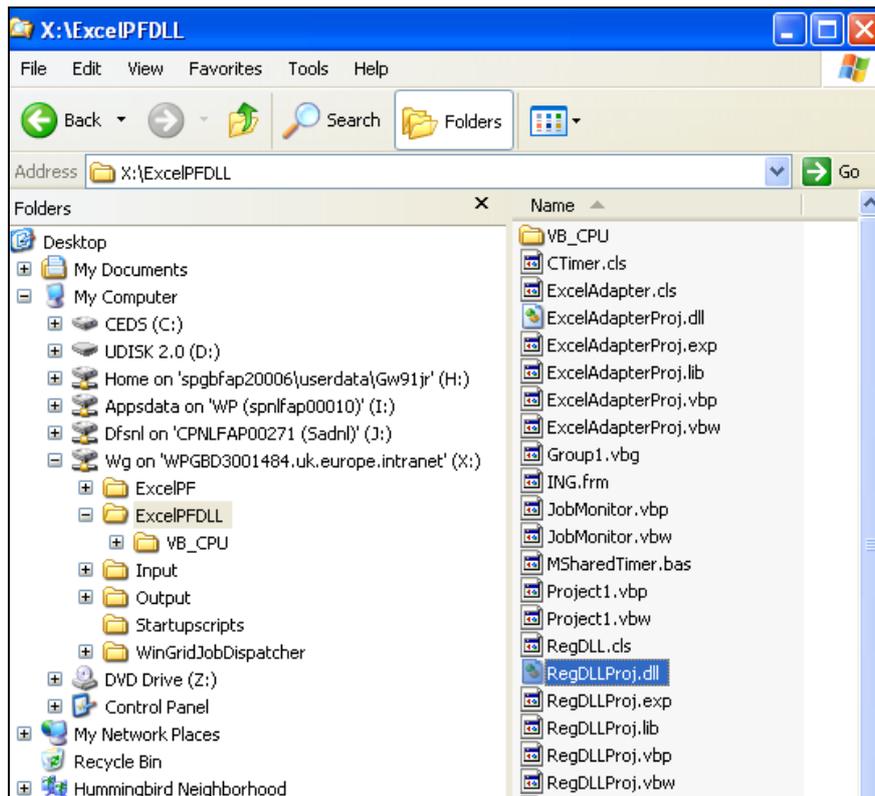
Screenshot 4: WinGrid directory structure

Screenshot 4 shows that X: is mapped to a shared folder called “Ing noweb” that is present in computer 192.168.0.210. In your case the share name will be different. But once mapped to X: you will get to see the same directory structure.

The directories *ExcelPF*, *ExcelPFDLL* and *WinGridJobDispatcher* contain program code and you need not interact with them. The folder *Input* and its subfolders *IRS* and *RBF* contain input files for WinGrid. The folder *Output* and its subfolders *IRS*, *Log* and *RBF* contain the output files generated by WinGrid. The folder *Startupscripts* contain the batch files required to start the program.

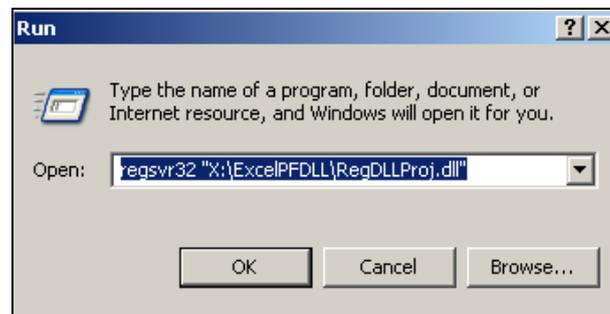
2.3 Register Dynamic Link Library (DLL)

A Dynamic Link Library (DLL) is an external program code that is invoked by an application dynamically at runtime. WinGrid is dependent on *RegDLLProj.dll*. This DLL has to be registered using a command called *regsvr32*. This is only a one time process. As shown in screenshot 5, the *RegDLLProj.dll* file that we have to register will appear under *X:\ExcelPFDLL*.



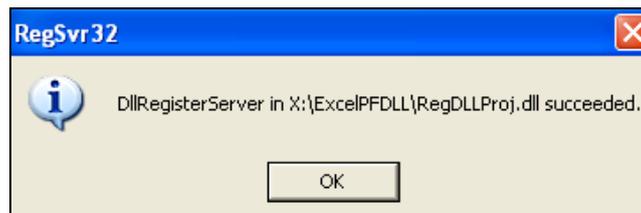
Screenshot 5: The RegDLLProj.dll is located in folder X:\ExcelPFDLL folder

In order to register this dll we have to navigate to *Start* → *Run*. This pops up a small window called *Run* and we have to enter the command *regsvr32 "X:\ExcelPFDLL\RegDLLProj.dll"* in the Open text box (screenshot 6).



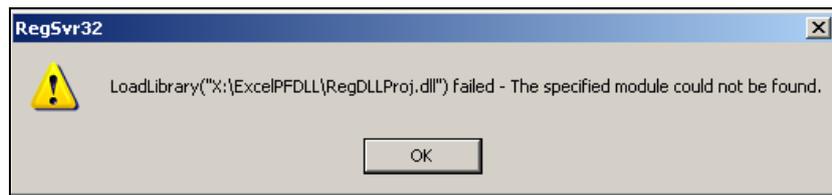
Screenshot 6: Registering the RegDLLProj.dll

If the DLL registration is successful then we get the following message box (screenshot 7).



Screenshot 7: Successful registration of RegDLLProj.dll

On failure, the unwelcoming message is (screenshot 8):



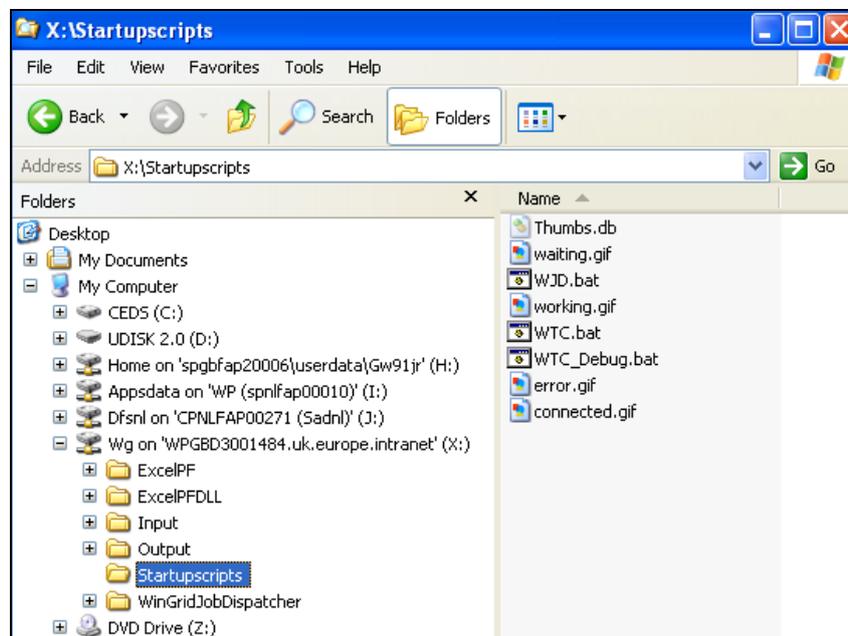
Screenshot 8: Unsuccessful registration of RegDLLProj.dll

If you are unable to register the DLL successfully please check whether the X: drive is mapped correctly and you are able to spot *RegDLLProj.dll* under *X:\ExcelPFDLL*.

Further Information: To un-register the already registered *RegDLLProj.dll* please use the command `regsvr32 -u "X:\ExcelPFDLL\RegDLLProj.dll"`. This you need to do if you decide that a PC previously a part of WinGrid infrastructure will not be required for processing anymore.

2.4 Execute Batch File

The batch files to start the WTC and the WJD processes are in the folder *X:\Startupscripts* (screenshot 9). In addition to the batch files (ending with .bat extension) there will be some graphics files (ending with .gif extension). Please ignore the .gif files. Clicking on a batch file may open a “Security Warning” prompting you to click the *Run* button to continue (screenshot 10).



Screenshot 9: The startup scripts to execute the programs are located in X:\Startupscripts



Screenshot 10: The Open File – Security Warning window

The *WJD.bat* will start the WinGrid Job Dispatcher (Master) process. This has to be started in only one computer. You need not worry about starting the WinGrid Thin Client (Worker) process from this location because in the current XXX configuration WTC processes is already started after you logon (this happens because this batch file is present in your startup folder “*C:\Documents and Settings\XXXXXX\Start Menu\Programs\Startup*”, where XXXXXX. is your profile name.

If you have deleted the WTC batch file from your startup folder list (you can safely do so!) then you need to start the WTC processes by clicking either *WTC.bat* or *WTC_Debug.bat* in *X:\Startupscripts*.

In case you have not deleted the WTC batch file present in your profiles folder but you have terminated the WTC program after logging in, then you need to restart WTC by clicking,
Start → *All Programs* → *Startup* → *WTC.bat* / *WTC_Debug.bat*, depending on which batch file is currently present.

As per the current configuration the startup folder invokes the *WTC_Debug.bat* file. You can replace *WTC_Debug.bat* with *WTC.bat* in the startup folder by deleting the former and adding the latter to “*C:\Documents and Settings\XXXXXX\Start Menu\Programs\Startup*”, where XXXXXX is your profile name.

But please remember:

- (1) Both *WTC.bat* and *WTC_Debug.bat* cannot be present at startup.
- (2) Do not include *WJD.bat* at startup.
- (3) Do not delete *WTC.bat*, *WTC_Debug.bat* or *WJD.bat* from *X:\Startupscripts*. This will make WinGrid non-functional until these files are replaced. You are allowed to delete, replace or add the WTC batch files in your profiles startup only.

About WJD.bat

Unlike its WTC counterparts the WJD does not have a separate WJD window. It only has a command window that cannot be ionized into the Windows System Tray. Terminating this command window will end the WJD program.

About *WTC.bat* and *WTC_Debug.bat*

The difference between *WTC.bat* and *WTC_Debug.bat* is how the program is started using JRE. When both these files are clicked two separate windows open up: (1) a command window, and (2) a white *WinGrid Thin Client* window. In case of *WTC.bat* the command window can be terminated and the WTC window can be iconified by clicking the minimize button (the program then appears only as an icon in the Windows System Tray). However, in case of *WTC_Debug.bat* the command window cannot be terminated as it will end the WinGrid program. It will also not be possible to iconify this command window (unlike the WTC window). This means that you will have an additional window open in your taskbar. More details on WTC can be found in section 3.

So what is the purpose of this additional command window incase of *WJD_Debug.bat* that cannot be terminated? As the name of the batch file suggests, this window is for debug purposes. It outputs a lot of messages which shows the current information exchanges taking place. It has to be reiterated that there is no separate debug version of WinGrid and that both *WTC.bat* and *WTC_Debug.bat*

2.5 Mapping Drive, Installing JACOB + JDIC, Setting CLASSPATH, Registering DLL and Including a new Computer as part of WinGrid Computation Infrastructure

This section of the document is maintained by **Robert Watson** (XXX).

Background Information

- All files needed to carry out this installation are stored on the network in ...\\Projects\\Brunel_Grid_Computing\\Software.
- Additional rights may have to be granted by IT in order to be able to do some of this work.
- All files are being stored on the Credit_Risk_Measurement area of the network.

Choose and set up a master PC

- The master PC is the one from which the control program runs. It is used to monitor the status of all the machines in the Grid. Being master PC is not a computational burden. The master PC may also simultaneously be a client PC. The master PC is whichever one runs the required batch file, WJD.bat. In the X:\ drive under Startupscripts.
- On all PCs (including the master) do the following.

Install client software

- Create a folder called *C:\Credit_Risk_Measurement\BrunelGRID*.

- Map the shared drive on the network. The same drive letter must be used on all client PCs. This will have to be done by the person who will be logged on to the machine.

Map \\SPGBFAP20004\SharedData1\Group
Data\GRGB001702\Projects\Brunel_Grid_Computing\WinGridCommonFiles as X:\.

- Copy the folder Software from ...\Projects\Brunel_Grid_Computing to C:\Credit_Risk_Measurement\BrunelGRID.
- Create a system environment variable called CLASSPATH. Copy into it the contents of C:\Credit_Risk_Measurement\BrunelGRID\Software\ClassPathValue.txt.
- Copy Jacob.dll from C:\Credit_Risk_Measurement\BrunelGRID\Software\jacob_1.9 to C:\WINDOWS\system32.
- Register RegDLLProj.dll by running the command given in C:\Credit_Risk_Measurement\BrunelGRID\Software\RegsvrCommand.txt.
- Copy X:\WTC_Debug.bat to User profile startup (C:\Documents and Settings\XXXXX\Start Menu\Programs\Startup). (where XXXX is the user name.)
- Add the name of the new client PC (and port number “60000” and an alias) to X:\Input\WinGridJobDispatcher.properties.xls.

Usage

- On the machine which will be the master, run X:\Startupscripts\WJD.bat.
- On each machine which will be a client (possibly including the machine which is the master), run Start – All Programs – Startup – WTC.bat.
- There also exists a debug version WTC_debug.bat.

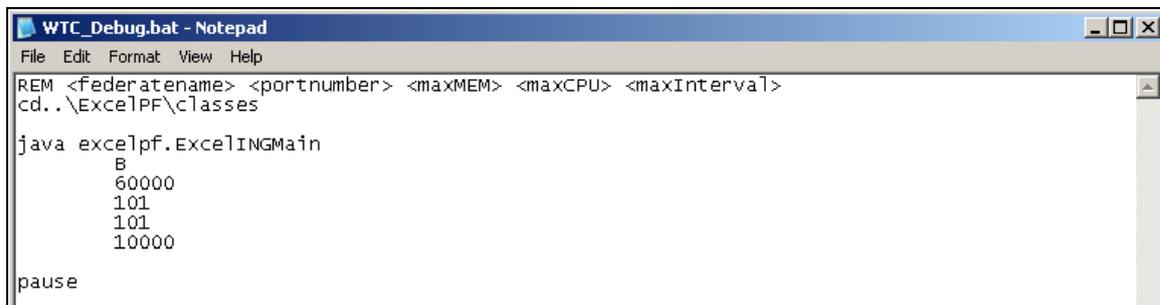
3. WTC – THE WINGRID THIN CLIENT

The WinGrid Thin Client (WTC) is responsible for listening to the WinGrid Job Dispatcher (WJD) for incoming job requests, accepting the job (or rejecting jobs in case of failures), passing the job for processing to its Worker Application (see figure 1) and returning the results of the job to the WJD. The WTC process should ideally be run on multiple computers. As discussed in section 2.4 above, the WTC process can be started by either executing the WTC_debug.bat or WTC.bat file.

3.1 WinGrid Thin Client (WTC) Arguments

Batch files are standard text files that contain commands that are to be processed by the Operating System. Let us take the example of WTC_Debug.bat. As seen from

screenshot 11 below, once this file is opened in notepad we see commands like REM, CD, JAVA, PAUSE etc (your batch file may contain additional commands). These commands are known to the Command Processor program of the Operating System.



```

WTC_Debug.bat - Notepad
File Edit Format View Help
REM <federatename> <portnumber> <maxMEM> <maxCPU> <maxInterval>
cd..\ExcelPF\classes

java excelpf.ExcelINGMain
    B
    60000
    101
    101
    10000

pause
  
```

Screenshot 11: The *WTC_Debug.bat* file opened in notepad

REM is a command that stands for REMARKS. The command processor on encountering a REM command will ignore whatever is written after it. The REM command is used in this batch file to inform the user of the arguments to the WTC program (think of arguments as variables in an equation). CD is a command to change directory to the path mentioned (*..\ExcelPF\classes* => go back one directory and then enter directory *ExcelPF\classes*). JAVA is the command to start the JRE and *excelpf.ExcelINGMain* is the name of the program. Let us now discuss the arguments to *excelpf.ExcelINGMain*. Please refer to the REM command also to see how the argument matches the placeholders defined by REM.

B → <federatename>

60000 → <portnumber>

101 → <maxMEM>

101 → <maxCPU>

10000 → <maxInterval>

Let us now discuss what these argument placeholders are:

<federatename> → The logical name of the computer running the WTC. Here we have assigned the name B. This argument is unimportant but should be present in the argument list.

<portnumber> → A port can be defined as an entry point for communication between two computers. For example, a house has only one address but may have multiple entrance doors. Similarly, a computer can have only one address (IPADDRESS) but can have multiple channels (ports) over which it can communicate. Each such channel will have a unique number which is commonly referred to as the port number. An application will use one port number for all its communication requirements. For example, all Internet communication is through port 8080. In our case all WinGrid communication will be through port 60000. [Note: we have taken a very simplistic view of inter-computer communication in this document. In reality a computer can have multiple Internet Protocol Addresses and

one application may utilize multiple ports to satisfy its communication requirements. However, WinGrid only uses port 60000 presently].

<maxMEM> → There is support in WinGrid to transparently start and stop job processing based on “**current memory load**” of the computer running the WTC. WTC program continuously monitors the current memory load of the system to decide whether jobs should be accepted or rejected, and also whether jobs under processing should be stopped. *This is an optional feature. It has also not been thoroughly tested (as of January 2006) and may have a few bugs.* The **<maxMEM>** value in the argument list is the memory load over which the processing would stop. Currently we see that **<maxMEM>** value is 101. This means that this feature is currently disabled as memory load can never be above 100. A value equal to or less than 100 will enable this feature.

<maxCPU> → There is support in WinGrid to transparently start and stop job processing based on “**current processor load**” of the computer running the WTC. This is possible because WTC continuously monitors the CPU load. *Like <maxMEM> this too is an optional feature and has not been thoroughly tested (As of January 2006).* The usage of **<maxCPU>** value is similar to **<maxMEM>**.

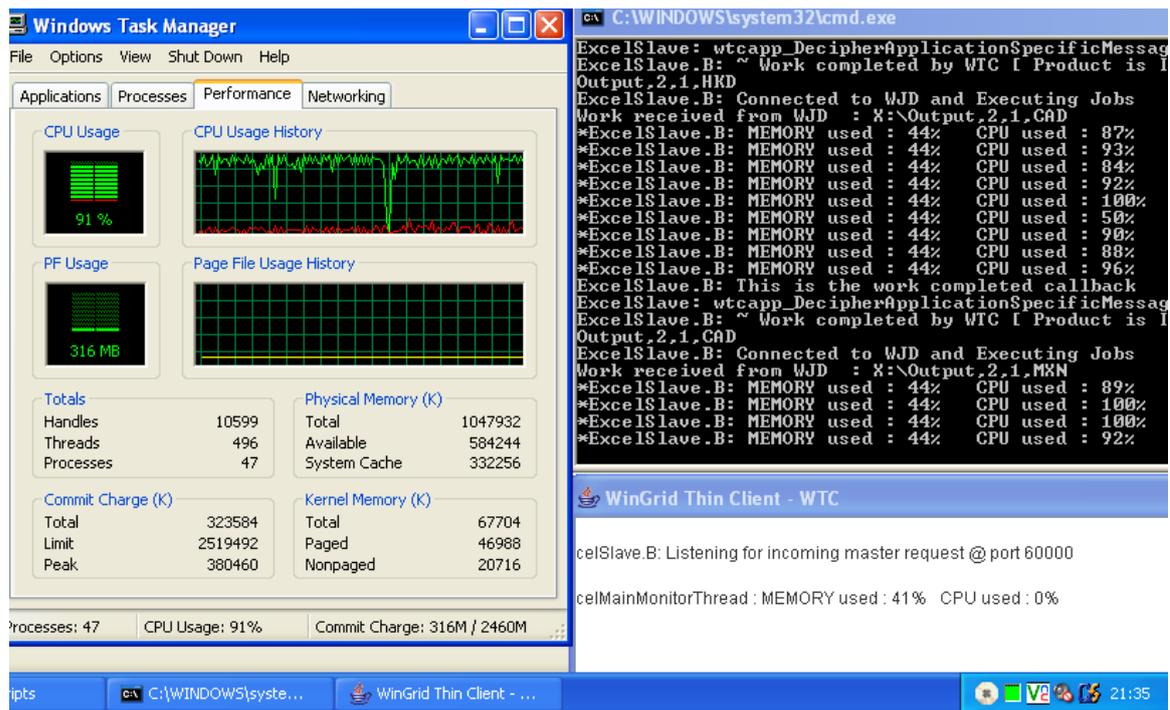
<maxInterval> → This argument is related to **<maxMEM>** and **<maxCPU>** arguments. As has been said earlier, the WTC continuously monitors the memory and CPU load of the computer running WTC. The time interval (in milliseconds) between two such measurements is defined by the place holder **<maxInterval>**. Thus a value 10000 means that the CPU load and the memory load will be measured every 10 seconds.

3.2 WTC_DEBUG Batch File

WTC_Debug.bat file has been discussed previously in section 2.4. This part of the document provides screenshots of the WTC process once the user clicks on *WTC_Debug.bat*.

Screenshot 12 shows that the WTC process has started and is utilizing around 90% CPU for processing the job. Running the *WTC_Debug.bat* opens up two separate windows, viz., the command window (in black) and the WTC window (in white). Furthermore you can see a “disc” icon in the Windows System Tray (more on this in section 3.4).

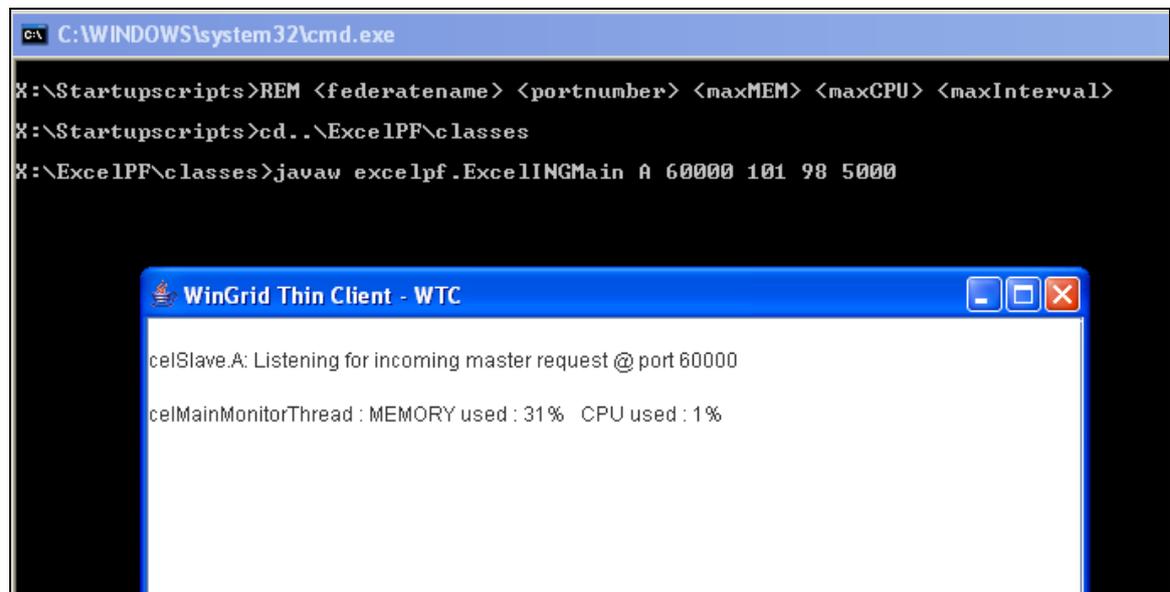
The command window outputs a lot of data while WTC process is running. This information is useful for (1) better user understanding of the system [A system that is understood better by the users will have a greater adoption rate], and (2) for debugging when things go wrong. It is not possible to iconize the command window and terminating it will stop the WTC program. The user has to keep this command window minimized at all time, implying an “extra” window in taskbar. The white WTC window, on the other hand, can be iconified so that it exists only in the System Tray.



Screenshot 12: WTC process is started using *WTC_Debug.bat* and you see both the command window (above right) and the WTC window (below right). None of these windows can be closed

3.3 WTC Batch File

Starting the WTC process by executing the *WTC.bat* file will enable the user to close the black command window without affecting the execution of WTC. The white WTC can be iconized to the Windows System Tray.



Screenshot 13: WTC process is started using *WTC.bat* and you see both the command window (in the background) and the WTC window (in the foreground). The command window can be closed

As is seen in screenshot 13, running the *WTC.bat* process will open two different windows. However, in this case, the user can safely close the black command window and the WTC program will still run.

3.4 WinGrid Thin Client (WTC) Status

Irrespective of the batch file used to invoke the WTC program, once the WTC process starts there will appear an icon in the Windows System Tray (the right hand portion of the task bar). This WinGrid icon is not static and will change depending on the state of WTC.



Screenshot 14: WinGrid icon in Windows System Tray is circled in red



: WTC is running but not connected to WJD (still waiting for connection- icon is WHITE).



: WTC is running and is connected to WJD (waiting for jobs- icon is GREEN).



: WTC is running and is connected to WJD but an error has occurred in WTC (icon is RED).

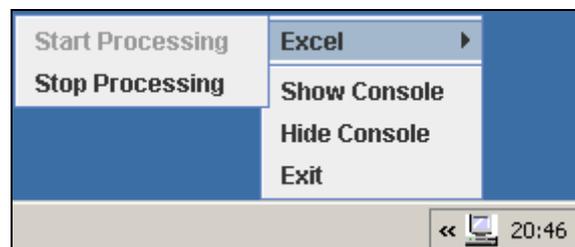


: WTC is running and is processing a job sent by WJD (this icon actually revolves).

Figure 2: Possible states of the WTC clients are represented by icons

3.5 WinGrid Thin Client (WTC) Menu

The WTC icons have a menu associated with them. The user can access this menu through right-click over the WinGrid icon on the System Tray. The menu, shown in screenshot 15, can be accessed irrespective of the state the WTC is in.



Screenshot 15: WTC menu can be accessed when the user right-clicks over the WinGrid icon

We will now describe the functionality provided by the WinGrid menu.

[Exit] → Terminates WTC in 5 seconds.

[Hide Console] → This will hide the open WinGrid WTC window

[Show Console] → This will display the WinGrid WTC window on the Task Bar (screenshot 16). A click on the minimized WTC window will now maximize it. [Show console] is also required when the user has minimized the WinGrid Thin Client window using the minimize option (found on the upper right-hand side of the WTC window) and now wants to make the window visible again.



Screenshot 16: WTC window minimized in the task bar

[Stop Processing] → Clicking on [Stop Processing] will immediately terminate any jobs running in WTC. This incomplete job will be pushed back to the WJD queue. Unlike the [Exit] option the WTC will still be running.

[Start Processing] → The user needs to click on [Start Processing] in order to make available his computer for processing again. This is only needed if the user had previously stopped processing jobs by selecting the [Stop Processing] option.

4. WJD – The WINGRID JOB DISPATCHER

The WJD process is started by the WJD.bat file. Opening the file in notepad we see the three arguments required by the WJD program. Argument one is the location of the *WinGridJobDispatcher.properties.xls* file (section 4.2), the second argument is the location of *WinGridApplicationSpecific.properties.xls* file (section 4.3) and the third argument is the directory that will contain the log files (section 4.4). The Excel™ files mentioned in the first two arguments are parameter files which the WJD reads before it starts dispatching work.

```

WJD.bat - Notepad
File Edit Format View Help
cd. .\wingridjobdispatcher\classes

java wingridjobdispatcher.core.wingridjobdispatcherMain
    "X:\Input\wingridjobdispatcher.properties.xls"
    "X:\Input\wingridApplicationSpecific.properties.xls"
    "X:\Output\Log"

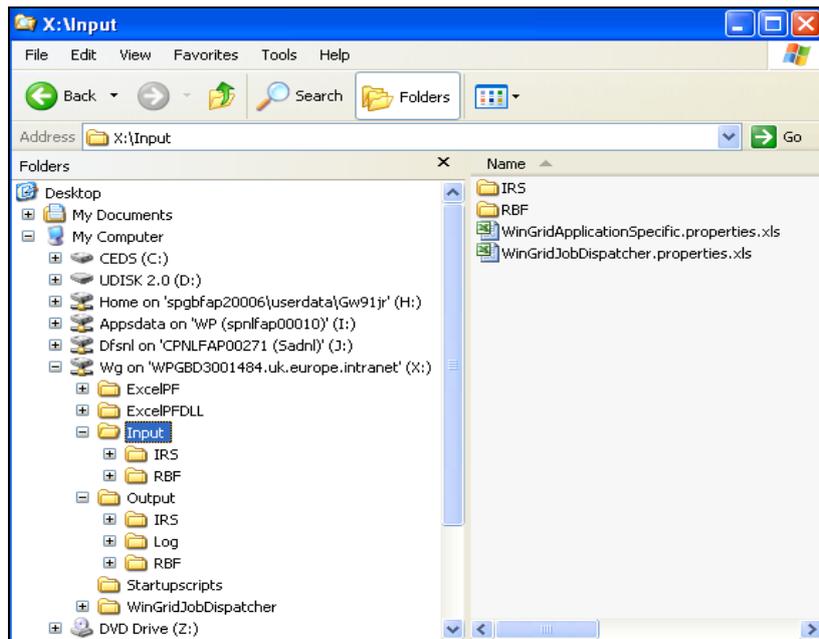
pause

```

Screenshot 17: The WJD.bat file opened in notepad

4.1 Configuring Parameter Files

The WinGrid parameter files are *WinGridJobDispatcher.properties.xls* and *WinGridApplicationSpecific.properties.xls*. These can be found under directory *X:\Input*. As shown in screenshot 18, the Input folder also contains two other folders, namely IRS and RBF. These folders contain the files that will be used by WTC to process jobs sent by WJD.



Screenshot 18: X:\Input is the location of the WinGrid parameter files

4.2 Configuring WinGrid Job Dispatcher (WJD) Parameter File

The WJD parameter file (*WinGridJobDispatcher.properties.xls*) contains a list of computers which the WJD will attempt to communicate with (screenshot 20). These computers should ideally have WTC running so that the WJD can reach it and dispatch work immediately. This file (screenshot 19) contains the logical name of the PC (computer name), the port number over which WTC is hearing for incoming WJD requests (port number) and an alias name (computer alias) for the computer. Any new computer that is assigned for WTC processing should have a corresponding entry in this file.

	A	B	C
1	Computer Names	Port Numbers	Computer Alias
2	WPGBD3001529.UK.EUROPE.INTRANET	60000	RAH_PC
3	WPGBD3001481.UK.EUROPE.INTRANET	60000	SAM_PC
4	WPGBD3001447.UK.EUROPE.INTRANET	60000	ROB_PC
5	WPGBD3001630.UK.EUROPE.INTRANET	60000	WJN_PC
6	WPGBD3001545.UK.EUROPE.INTRANET	60000	JTN_PC
7	WPGBD3001484.UK.EUROPE.INTRANET	60000	SPRPC1
8	WPGBD3001497.UK.EUROPE.INTRANET	60000	SPRPC2
9	WPGBD3001855.UK.EUROPE.INTRANET	60000	JTN_LC
10	WPGBD3001484.UK.EUROPE.INTRANET	60000	NAV_PC

Screenshot 19: The WJD Parameter File (*WinGridJobDispatcher.properties.xls*)

```

C:\WINDOWS\system32\cmd.exe
[ CAD SPRPC1 WorkDone ] [ ----- InQueue ] [ ----- InQueue ]
[ CHF SPRPC2 WorkDone ] [ ----- InQueue ] [ ----- InQueue ]
[ CZK SPRPC1 WorkDone ] [ ----- InQueue ] [ ----- InQueue ]
[ DKK SPRPC2 WorkDone ] [ ----- InQueue ] [ ----- InQueue ]
[ EUR SPRPC1 WorkDone ] [ ----- InQueue ] [ ----- InQueue ]
[ GBP SPRPC2 WorkDone ] [ ----- InQueue ] [ ----- InQueue ]
[ HKD SPRPC1 WorkDone ] [ ----- InQueue ] [ ----- InQueue ]
[ HUF SPRPC2 WorkDone ] [ ----- InQueue ] [ ----- InQueue ]
[ INR SPRPC1 WorkDone ] [ ----- InQueue ] [ ----- InQueue ]
[ JPY SPRPC2 WorkDone ] [ ----- InQueue ] [ ----- InQueue ]
[ KRW SPRPC1 WorkDone ] [ ----- InQueue ] [ ----- InQueue ]
[ MXN SPRPC2 WorkDone ] [ ----- InQueue ] [ ----- InQueue ]
[ NOK SPRPC1 WorkDone ] [ ----- InQueue ] [ ----- InQueue ]
[ NZD SPRPC2 Running ] [ ----- InQueue ] [ ----- InQueue ]
[ PHP SPRPC1 Running ] [ ----- InQueue ] [ ----- InQueue ]
[ PLN ----- InQueue ] [ ----- InQueue ] [ ----- InQueue ]
[ SEK ----- InQueue ] [ ----- InQueue ] [ ----- InQueue ]
[ SGD ----- InQueue ] [ ----- InQueue ] [ ----- InQueue ]
[ SKK ----- InQueue ] [ ----- InQueue ] [ ----- InQueue ]
[ TWD ----- InQueue ] [ ----- InQueue ] [ ----- InQueue ]
[ USD ----- InQueue ] [ ----- InQueue ] [ ----- InQueue ]
[ ZAR ----- InQueue ] [ ----- InQueue ] [ ----- InQueue ]

0-Jan-2007 12:48:27 wingridjobdispatcher.util.WinGridJobDispatcherLogger logInformationMessage
NFO: Work being sent by WJD to SPRPC1 [WPGBD3001484.UK.EUROPE.INTRANET] @ [60000] -> X:\Output_2,1,PHP
TM_PC => More work remaining. Trying to connect to JTM_PC [WPGBD3001545.UK.EUROPE.INTRANET] @ [60000] !!!!!
OB_PC => More work remaining. Trying to connect to ROB_PC [WPGBD3001447.UK.EUROPE.INTRANET] @ [60000] !!!!!
AH_PC => More work remaining. Trying to connect to RAH_PC [WPGBD3001529.UK.EUROPE.INTRANET] @ [60000] !!!!!
JM_PC => More work remaining. Trying to connect to JWM_PC [WPGBD3001630.UK.EUROPE.INTRANET] @ [60000] !!!!!
AM_PC => More work remaining. Trying to connect to SAM_PC [WPGBD3001481.UK.EUROPE.INTRANET] @ [60000] !!!!!
0-Jan-2007 12:48:34 wingridjobdispatcher.util.WinGridJobDispatcherLogger logSevereMessage
SEVERE: WJD 10-01-2007 12:48:34 : UnknownHostException. [WPGBD3001855.UK.EUROPE.INTRANET] @ [60000]
10-Jan-2007 12:48:44 wingridjobdispatcher.util.WinGridJobDispatcherLogger logSevereMessage
SEVERE: WJD 10-01-2007 12:48:44 : UnknownHostException. [WPGBD3001855.UK.EUROPE.INTRANET] @ [60000]

```

Screenshot 20: WJD is trying to connect to WTCs based on connection information in *WinGridJobDispatcher.properties.xls* file

4.3 Configuring WinGrid Job Dispatcher (WJD) Application Specific Parameter File

The WJD application specific parameter file (*WinGridApplicationSpecific.properties.xls*) contains application specific parameters that are required for our specific XXX application. As shown in screenshot 21, it requires five different inputs, viz., the name of the output directory, the name of the product to simulate (IRS or RBF), the operation to perform (create table, create profiles or both), the filename to simulate and, finally, whether the master has crashed. All this information is in worksheet called “General”. The WJD Application specific parameter file also has a further two worksheets, namely “RBF” and “IRS”, which have data specific to RBF and IRS simulations respectively.

4.3.1 General Worksheet of WinGrid Application Specific Parameter File

We will now look at the parameters in the “General WorkSheet”.

- **Output Directory:** The purpose of the output directory is to store temporary information generated during the simulation. This folder must be present in the X: drive and be accessible to all WTC clients. Each client extensively reads and writes to this directory. Currently the shared directory for output is *X:\Output*.
- **Name of the Project:** It can have a value of either 1 (for RBF) or 2 (for IRS).
- **Name of the Operation:** A value 1 indicates that both profiles + tables will be created (for either RBS or IRS depending on the value of the previous field), value 2 is for profiles only and value 3 is for creating tables only. *Options 2 and 3 have not been rigorously tested because in most cases Option 1 is all that is needed.*

- Filename to Simulate:** This is the name of the file that will be used to generate the profiles. There are two different files which are used for creating IRS and RBF profiles respectively. These files are kept in two different directories, viz., *X:\Input\IRS* and *X:\Input\RBF* respectively. In practice however, it is the same file with the same name but kept under two directories.

	A	B
1	Shared directory for output file	X:\Output
2		
3	Name of the product 1-> RBF	2
4	Name of the product 2-> IRS	
5	Operation 1->Create profiles + tables Operation 2->Create profiles only Operation 3->Create tables only	1
6		
7	Filename to simulate	X:\Input\IRS\Generate_Profiles.xls
8		
9	Master Crash 1->No Master Crash 2->Yes	1

Screenshot 21: The “General” worksheet of the WJD Application Specific Parameter File (*WinGridApplicationSpecific.properties.xls*)

Very Important:

If [Name of the Project] is 1 → Then [Filename to Simulate] should point to *X:\Input\RBF*

If [Name of the Project] is 2 → Then [Filename to Simulate] should point to *X:\Input\IRS*

- Master Crash:** This field can have a value of either 1 or 2. If WJD had crashed during previous execution then set the value of this variable to 2. If the WJD had completed successful processing during the previous run, then set the value to 1.

A new run of WJD with a master crash value of 1 will mean that the previously generated files (kept in shared directory for output file: *X:\Output*) will be deleted (see screenshot 22) and the WTC’s will create new files. However, if WJD is run with a master crash value of 2 then the previous files in the output directory will not be deleted. Thus, the computation will restart from a previous state.

```

C:\WINDOWS\system32\cmd.exe
Now Deleting file : Interest Rate Swap Pay Floating Currency DKK Tenor 324.xls
Now Deleting file : Interest Rate Swap Pay Floating Currency DKK Tenor 336.xls
Now Deleting file : Interest Rate Swap Pay Floating Currency DKK Tenor 348.xls
Now Deleting file : Interest Rate Swap Pay Floating Currency DKK Tenor 36.xls
Now Deleting file : Interest Rate Swap Pay Floating Currency DKK Tenor 360.xls
Now Deleting file : Interest Rate Swap Pay Floating Currency DKK Tenor 48.xls
Now Deleting file : Interest Rate Swap Pay Floating Currency DKK Tenor 6.xls
Now Deleting file : Interest Rate Swap Pay Floating Currency DKK Tenor 60.xls
Now Deleting file : Interest Rate Swap Pay Floating Currency DKK Tenor 72.xls
Now Deleting file : Interest Rate Swap Pay Floating Currency DKK Tenor 84.xls
Now Deleting file : Interest Rate Swap Pay Floating Currency DKK Tenor 96.xls
Now Deleting file : Interest Rate Swap Pay Floating Currency EUR Tenor 108.xls
Now Deleting file : Interest Rate Swap Pay Floating Currency EUR Tenor 12.xls
Now Deleting file : Interest Rate Swap Pay Floating Currency EUR Tenor 120.xls
Now Deleting file : Interest Rate Swap Pay Floating Currency EUR Tenor 132.xls
Now Deleting file : Interest Rate Swap Pay Floating Currency EUR Tenor 144.xls
Now Deleting file : Interest Rate Swap Pay Floating Currency EUR Tenor 156.xls
Now Deleting file : Interest Rate Swap Pay Floating Currency EUR Tenor 168.xls
Now Deleting file : Interest Rate Swap Pay Floating Currency EUR Tenor 180.xls
Now Deleting file : Interest Rate Swap Pay Floating Currency EUR Tenor 192.xls
Now Deleting file : Interest Rate Swap Pay Floating Currency EUR Tenor 204.xls
Now Deleting file : Interest Rate Swap Pay Floating Currency EUR Tenor 216.xls
Now Deleting file : Interest Rate Swap Pay Floating Currency EUR Tenor 228.xls
Now Deleting file : Interest Rate Swap Pay Floating Currency EUR Tenor 24.xls
Now Deleting file : Interest Rate Swap Pay Floating Currency EUR Tenor 240.xls
Now Deleting file : Interest Rate Swap Pay Floating Currency EUR Tenor 252.xls
Now Deleting file : Interest Rate Swap Pay Floating Currency EUR Tenor 264.xls
Now Deleting file : Interest Rate Swap Pay Floating Currency EUR Tenor 276.xls
Now Deleting file : Interest Rate Swap Pay Floating Currency EUR Tenor 288.xls
Now Deleting file : Interest Rate Swap Pay Floating Currency EUR Tenor 300.xls
Now Deleting file : Interest Rate Swap Pay Floating Currency EUR Tenor 312.xls
Now Deleting file : Interest Rate Swap Pay Floating Currency EUR Tenor 324.xls
Now Deleting file : Interest Rate Swap Pay Floating Currency EUR Tenor 336.xls
Now Deleting file : Interest Rate Swap Pay Floating Currency EUR Tenor 348.xls
Now Deleting file : Interest Rate Swap Pay Floating Currency EUR Tenor 36.xls
Now Deleting file : Interest Rate Swap Pay Floating Currency EUR Tenor 360.xls
Now Deleting file : Interest Rate Swap Pay Floating Currency EUR Tenor 420.xls
Now Deleting file : Interest Rate Swap Pay Floating Currency EUR Tenor 48.xls
Now Deleting file : Interest Rate Swap Pay Floating Currency EUR Tenor 480.xls
Now Deleting file : Interest Rate Swap Pay Floating Currency EUR Tenor 540.xls
Now Deleting file : Interest Rate Swap Pay Floating Currency EUR Tenor 6.xls
Now Deleting file : Interest Rate Swap Pay Floating Currency EUR Tenor 60.xls

```

Screenshot 22: Temporary output files (generated by WTCs) will be deleted by WJD if the Master Crash value is set to 1 in the WJD application specific parameter file

4.3.2 IRS and RBF Worksheets of WinGrid Application Specific Parameter File

We will now focus our discussion on the “RBF” and “IRS” worksheets in the WinGrid application specific parameters file. Each of these worksheets has a list of currencies. Each currency is a separate unit of computation (job). The WJD reads this list (it reads RBF worksheet if the *name of product* in “general” worksheet is 1, and IRS if the *name of product* is 2) and allocates job (one currency name + other parameters) to each connected WTC.

The WTCs will process the jobs it receives from WJD and return results (in our case it is only a job completion message because results that are computed by WTC are stored in the shared output folder that the WJD can also access). After one currency has been successfully computed the WJD will send the next currency in its queue. This process repeats until all the currencies have completed the three different phases of processing.

	A	B	C	D	E
1	AUD				
2	CAD				
3	CHF				
4	CZK				
5	DKK				
6	EUR				
7	GBP				
8	HKD				
9	HUF				
10	INR				
11	JPY				

Screenshot 23: The “IRS” worksheet of the WJD Application Specific Parameter File (*WinGridApplicationSpecific.properties.xls*)

4.3.3 Three Phases of IRS and RBF Simulation

There are three phases to both IRS and RBF simulations.

Phase 1: Create Profiles

Phase 2: Create EPE tables from the output files generated by WTCs

Phase 3: Create PFE tables from the output files generated by WTCs

The output files required by WTCs to create EPE and PFE tables for phase 2 and phase 3 of the simulations are generated by WTCs themselves in phase 1. Thus there exists a dependency between jobs. WJD has implemented an algorithm which only allows second stage processing when the first stage processing is complete, and allows third stage processing only if the second stage processing is complete.

For both IRS and RBF simulations, after the WTC processing has ended the WJD will create two master files – one for EPE and another for PFE – with the objective of presenting collective results to the user. The WJD does this by transferring data from the temporary Excel™ files (created by each WTC client during phase 2 and phase 3 of the simulation) to the Master EPE and PFE files. Thus, if there are 8 WTCs then the WJD will have to combine results generated by each of the WTCs. And it has to do it twice - once for master EPE table generation and again for the master PFE table generation.

The files required by both WTCs and the WJD for executing IRS and RBF simulations can be found under *X:\Input\IRS* and *X:\Input\RBF* directories respectively. As shown in screenshot 24 we need a total of 5 different files for both IRS and RBF simulation (3 required by WTC and 2 by WJD). The files on which the different IRS and RBF phases are dependent are listed below.

WTC requirement for IRS (see directory *X:\Input\IRS*):

- **Phase 1:** Generate_Profiles.xls
- **Phase 2:** Generate_Profiles_[IRS_Create_Table_EPE].xls
- **Phase 3:** Generate_Profiles_[IRS_Create_Table_PFE].xls

WTC requirement for RBF (see directory *X:\Input\RBF*):

- **Phase 1:** Generate_Profiles.xls
- **Phase 2:** Generate_Profiles_[RBF_Create_Table_EPE].xls
- **Phase 3:** Generate_Profiles_[RBF_Create_Table_PFE].xls

The files required by WJD for both IRS and RBF simulations are:

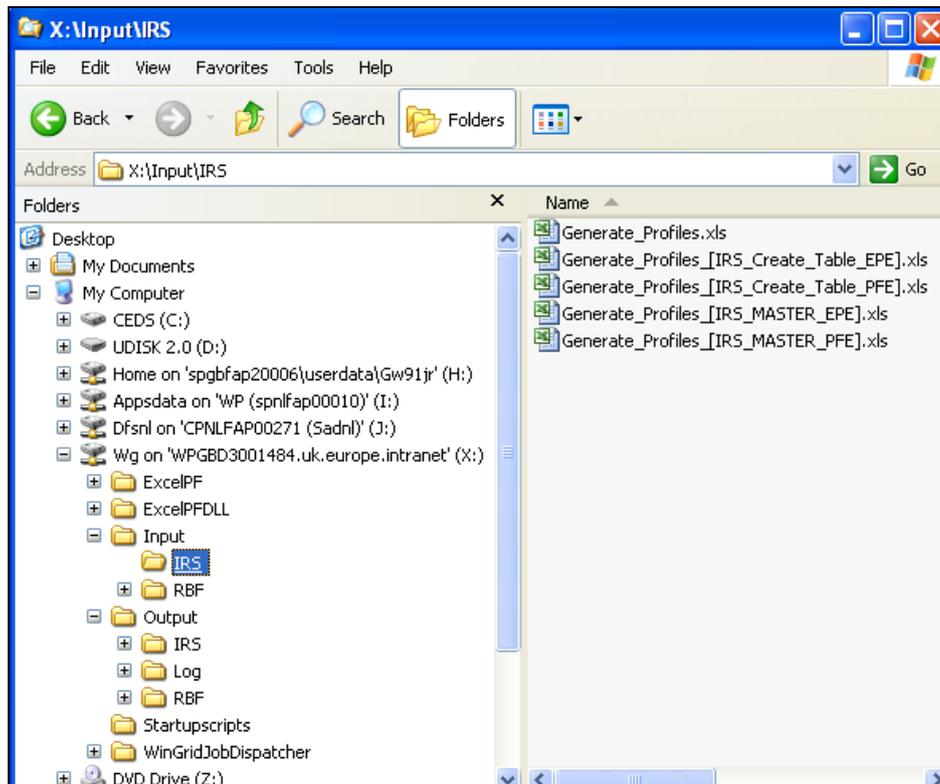
WJD requirement for IRS (see directory *X:\Input\IRS*):

- Generate_Profiles_[IRS_MASTER_EPE].xls
- Generate_Profiles_[IRS_MASTER_PFE].xls

WJD requirement for RBF (see directory *X:\Input\RBF*):

- Generate_Profiles_[RBF_MASTER_EPE].xls
- Generate_Profiles_[RBF_MASTER_PFE].xls

Further information on how these files are used during processing can be found under section 4.6 (WinGrid Job Dispatcher (WJD) Execution Completion).

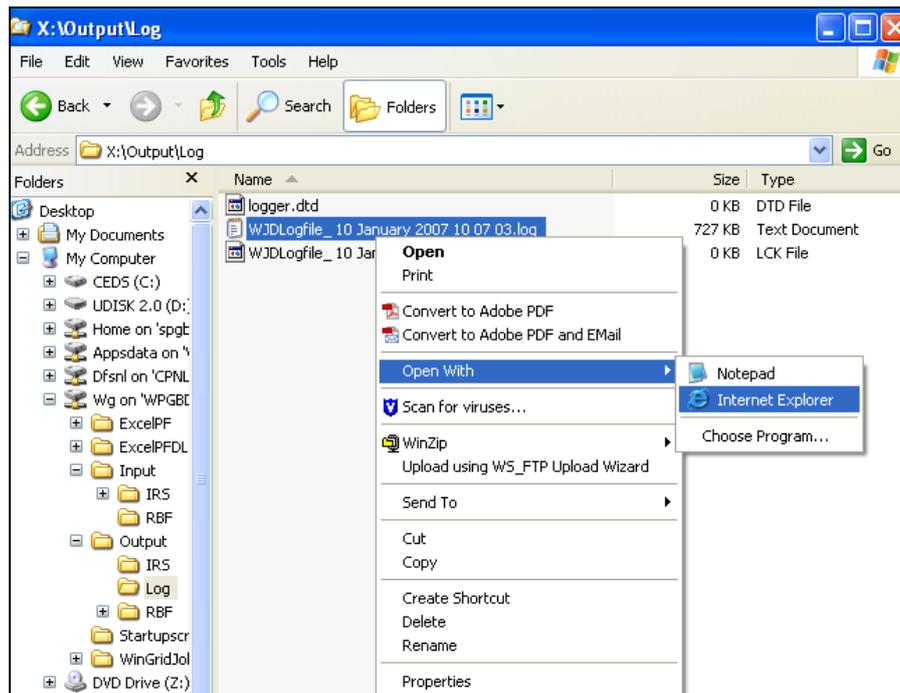


Screenshot 24: The files required for IRS and RBF simulations can be found under `X:\Input\IRS` and `X:\Input\RBF` directories respectively

4.4 WinGrid Job Dispatcher (WJD) Log File Directory

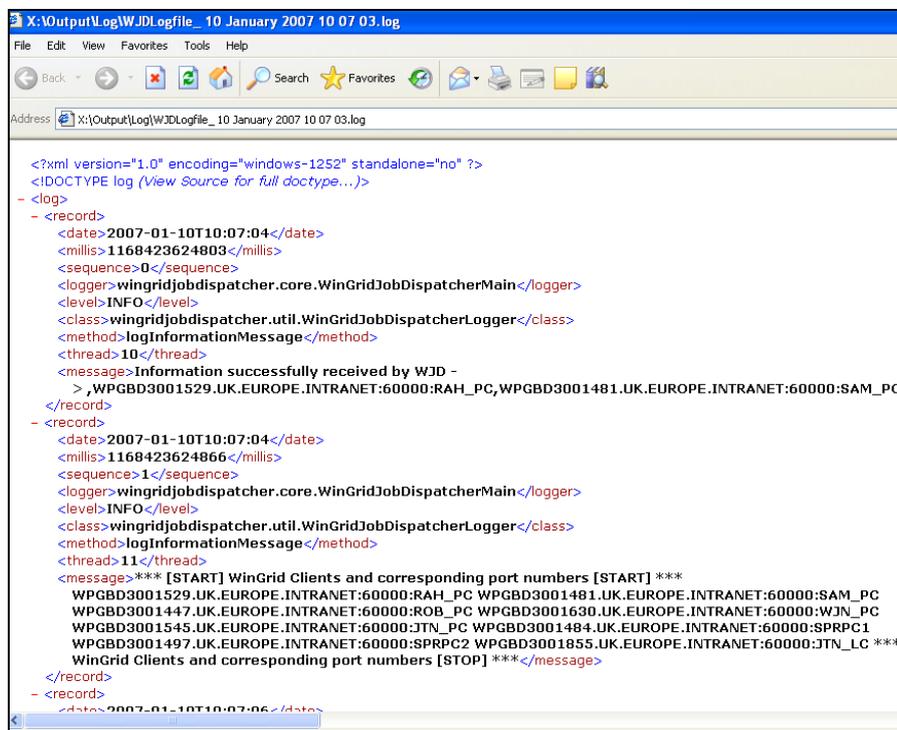
The third argument to the `WJD.bat` file (screenshot 17) is the location of the directory that will contain the log files. These log files are generated automatically during the WinGrid-based simulation runs and record a variety of information that can be used for debugging. The default location of this WJD log file directory is in `X:\Output\Log`.

The log files generated by WJD have filenames that include the system date and the system time (e.g.: `WJD_Logfile_10_January_2007_10_07_03.log`). This allows the previously generated log files to exist along side the newer logs. The log information is written in XML (meaning there are well-defined tags that qualify the information written) and it can be opened using either Notepad or Internet Explorer program, the latter being the preferred option because it can render XML data. However, for Internet Explorer to open this log file, another file called `logger.dtd` must exist in the log folder. If it does not then Internet Explorer will throw you an error. In this case simply create a new file with the name `logger.dtd`. This newly created file should be empty (size 0KB).



Screenshot 25: Opening the log file using Internet Explorer

As show in screenshot 25, the log file can be opened through right click on the file name and then selecting option *Open With* and then selecting Internet Explorer. This will display the log file information parsed according to XML tag hierarchy (screenshot 26).



Screenshot 26: The Log file information displayed in Internet Explorer

4.5 WinGrid Job Dispatcher (WJD) Execution

WJD process is started when the user executes the *WJD.bat* batchfile in folder *X:\Starupscripts*. As noted earlier in section 2.4, the user may be prompted by a “Security Warning” window to confirm whether the software should be run. Please click *Run* to begin execution of the WJD process. Only one computer in the WinGrid infrastructure will be allowed to run the WJD process at any one time. There are checks in the system which enforces this.

Depending on the “Master Crash” parameter set in the WJD application specific properties file *WinGridApplicationSpecific.properties.xls* file (see screenshot 21), the WJD may either delete the contents of *X:\Output\RBF* or *X:\Output\IRS* (depending on whether the system is configured for RBF or IRS simulation) or may decide to skip the deletion process. It will then try to establish connection with WTC clients by reading connection information from *WinGridJobDispatcher.properties.xls* file (screenshot 19).

The screenshot shows a Windows command prompt window titled "C:\WINDOWS\system32\cmd.exe" displaying the output of the WinGrid Job Dispatcher (WJD) process. The output is a table of jobs with columns for job ID, WTC, status, and progress. The jobs are listed in a grid format, with columns for job ID, WTC, status, and progress. The output is as follows:

```

10-Jan-2007 12:12:46 wingridjobdispatcher.util.WinGridJobDispatcherLogger logInf
INFO: STAGE 1 (PROFILES) STAGE 2 (PEP) STAGE 3 (PEP)
-verb CHD SPRPC2 WorkDone | | SPRPC1 WorkDone | | SPRPC1 WorkDone
-versi CAD SPRPC2 WorkDone | | SPRPC2 WorkDone | | SPRPC2 WorkDone
-verb CHF SPRPC1 WorkDone | | SPRPC2 WorkDone | | SPRPC1 WorkDone
-verb CZK SPRPC2 WorkDone | | SPRPC1 WorkDone | | SPRPC2 WorkDone
showv DKV SPRPC1 WorkDone | | SPRPC2 WorkDone | | SPRPC1 WorkDone
-jve- EUR SPRPC2 WorkDone | | SPRPC1 WorkDone | | SPRPC2 WorkDone
-? - GBP SPRPC1 WorkDone | | SPRPC2 WorkDone | | SPRPC1 WorkDone
-ea|< HUF SPRPC2 WorkDone | | SPRPC2 WorkDone | | SPRPC2 Running
-enabl JPY SPRPC1 WorkDone | | SPRPC2 WorkDone | | SPRPC1 InQueue
- - KRW SPRPC2 WorkDone | | SPRPC1 WorkDone | | SPRPC2 InQueue
-disab MMN SPRPC1 WorkDone | | SPRPC2 WorkDone | | SPRPC1 InQueue
-esa NZZ SPRPC1 WorkDone | | SPRPC1 WorkDone | | SPRPC2 InQueue
- - PHP SPRPC2 WorkDone | | SPRPC2 WorkDone | | SPRPC2 InQueue
- - PLN SPRPC1 WorkDone | | SPRPC2 WorkDone | | SPRPC1 InQueue
- - SEK SPRPC2 WorkDone | | SPRPC1 WorkDone | | SPRPC2 InQueue
-agent SGD SPRPC1 WorkDone | | SPRPC2 WorkDone | | SPRPC1 InQueue
- - SKK SPRPC2 WorkDone | | SPRPC1 WorkDone | | SPRPC2 InQueue
- - USD SPRPC2 WorkDone | | SPRPC2 WorkDone | | SPRPC1 InQueue
- - ZAR SPRPC1 WorkDone | | SPRPC2 WorkDone | | SPRPC2 InQueue
- - javaa INFO: Work being sent by WJD to SPRPC1 [WPGBD3001484.UK.EUROPE.INTRANET] @ 16000
WJM_PC => More work remaining. Trying to connect to WJM_PC [WPGBD3001630.UK.EURO
ROE_PC => More work remaining. Trying to connect to ROE_PC [WPGBD3001447.UK.EURO
javac RAH_PC => More work remaining. Trying to connect to RAH_PC [WPGBD3001529.UK.EURO
javac JTN_PC => More work remaining. Trying to connect to JTN_PC [WPGBD3001545.UK.EURO
operable SAM_PC => More work remaining. Trying to connect to SAM_PC [WPGBD3001481.UK.EURO
10-Jan-2007 12:12:49 wingridjobdispatcher.util.WinGridJobDispatcherLogger logSev
SEVERE: WJD 10-01-2007 12:12:49: UnknownHostException. [WPGBD3001855.UK.EUROPE
10-Jan-2007 12:12:59 wingridjobdispatcher.util.WinGridJobDispatcherLogger logSev
SEVERE: WJD 10-01-2007 12:12:59: UnknownHostException. [WPGBD3001855.UK.EUROPE

```

Red arrows point to the following elements in the screenshot:

- #Job**: Points to the job ID column.
- WtcP1**: Points to the WTC column.
- StatusP1**: Points to the status column.
- WtcP2**: Points to the WTC column.
- StatusP2**: Points to the status column.
- WtcP3**: Points to the WTC column.
- StatusP3**: Points to the status column.

A red box highlights the "STAGE 1 (PROFILES)" and "STAGE 2 (PEP)" sections of the output. A red arrow points from this box to the text "Stages of Processing" on the right side of the image.

Screenshot 27: The WJD console showing running jobs

As the WJD establishes connection with the WTCs, the WJD job dispatching algorithm assigns jobs to the WTCs for processing. This algorithm makes sure that the dependencies between jobs are maintained through phases and incomplete jobs are reassigned as long as there is even one waiting and functional WTC in the WinGrid computation infrastructure.

The WJD console (screenshot 27) shows both the job status (*StatusP1*, *StatusP2*, *StatusP3*) and the status of all the WTCs (*WtcP1*, *WtcP2*, *WtcP3*) across all the three phases of processing. The jobs (*#Job*) that are to be processed are the foreign currency values extracted from either the IRS or RBF worksheet of the WJD application specific parameter file called *WinGridApplicationSpecific.properties.xls* file (screenshot 23).

The job status (*StatusP1*, *StatusP2*, *StatusP3*) can have eight possible values (each value is either 7 or 9 characters for formatting purposes), irrespective of whether it is first stage of processing (*STAGE1 [PROFILES]*), the second stage of processing (*STAGE2 [EFE]*) or the final stage of processing (*STAGE3 [PFE]*). These are:

InQueue → Job is in the queue. The next available WTC will get the job.

Assignd → Job has been assigned to a WTC.

Running → Job is currently running in a WTC.

WorkDne → Job is complete.

InQueue → A previously failed job is again in the queue. The next available WTC will get the job.

Assignd → A previously failed job has been assigned to a WTC.

Running → A previously failed job is running in a WTC.

WorkDne → A previously failed job is now complete.

The status of a WTC (WTCs are identified in the WJD console by its alias names [see screenshot 19]), can be determined by looking at columns that represent different phases of processing (*WtcP1* column for first stage processing, *WtcP2* column for second stage processing and *WtcP3* column for the third phase processing). If the WTC name cannot be found then the other messages displayed at regular interval by WJD must be checked. These messages are self-explanatory and might indicate that an WTC is not assigned work because the previous stage of work is not complete (because of underlying job dependencies between phases) or because there is no more work to be sent (all jobs are either assigned or running); the WTC computer could not be found (unknown host exception); the WTC processes is not listening for WJD etc. See screenshot 28. Being conversant with the details output by the WJD console will take some time.

```

C:\WINDOWS\system32\cmd.exe
[ CHF SPRPC1 WorkDone ] [ SPRPC2 WorkDone ] [ SPRPC1 WorkDone ]
[ CZK SPRPC2 WorkDone ] [ SPRPC1 WorkDone ] [ SPRPC2 WorkDone ]
[ DKK SPRPC1 WorkDone ] [ SPRPC2 WorkDone ] [ SPRPC1 WorkDone ]
[ EUR SPRPC2 WorkDone ] [ SPRPC1 WorkDone ] [ SPRPC2 WorkDone ]
[ GBP SPRPC1 WorkDone ] [ SPRPC2 WorkDone ] [ SPRPC1 WorkDone ]
H:\>java [ HKD SPRPC2 WorkDone ] [ SPRPC1 WorkDone ] [ SPRPC2 Running ]
Usage: jav [ HUF SPRPC1 WorkDone ] [ SPRPC2 WorkDone ] [ SPRPC1 Assignd ]
or jav [ INR SPRPC2 WorkDone ] [ SPRPC1 WorkDone ] [ InQueue ]
[ JPY SPRPC1 WorkDone ] [ SPRPC2 WorkDone ] [ InQueue ]
[ KRW SPRPC2 WorkDone ] [ SPRPC1 WorkDone ] [ InQueue ]
[ MWN SPRPC1 WorkDone ] [ SPRPC2 WorkDone ] [ InQueue ]
where opti [ NOK SPRPC2 WorkDone ] [ SPRPC1 WorkDone ] [ InQueue ]
-clien [ NZD SPRPC1 WorkDone ] [ SPRPC2 WorkDone ] [ InQueue ]
-serve [ PHP SPRPC2 WorkDone ] [ SPRPC1 WorkDone ] [ InQueue ]
-hotsp [ PLM SPRPC1 WorkDone ] [ SPRPC2 WorkDone ] [ InQueue ]
[ SEK SPRPC2 WorkDone ] [ SPRPC1 WorkDone ] [ InQueue ]
[ SGD SPRPC1 WorkDone ] [ SPRPC2 WorkDone ] [ InQueue ]
-ep < [ SKK SPRPC2 WorkDone ] [ SPRPC1 WorkDone ] [ InQueue ]
-class [ TWD SPRPC1 WorkDone ] [ SPRPC2 WorkDone ] [ InQueue ]
[ USD SPRPC2 WorkDone ] [ SPRPC1 WorkDone ] [ InQueue ]
[ ZAR SPRPC1 WorkDone ] [ SPRPC2 WorkDone ] [ InQueue ]
-WJD searching for work to assign WPGBD3001484.UK.EUROPE.INTRANET: MORE WORK REMA
10-Jan-2007 12:12:46 wingridjobdispatcher.util.WinGridJobDispatcherLogger logInf
INFO: STAGE 1 (PROFILES) STAGE 2 (EPE) STAGE 3 (PFE)
-versi [ AUD SPRPC2 WorkDone ] [ SPRPC2 WorkDone ] [ SPRPC1 WorkDone ]
-versi [ CAD SPRPC1 WorkDone ] [ SPRPC2 WorkDone ] [ SPRPC2 WorkDone ]
[ CHF SPRPC1 WorkDone ] [ SPRPC2 WorkDone ] [ SPRPC1 WorkDone ]
[ CZK SPRPC2 WorkDone ] [ SPRPC1 WorkDone ] [ SPRPC2 WorkDone ]
-showu [ DKK SPRPC1 WorkDone ] [ SPRPC2 WorkDone ] [ SPRPC1 WorkDone ]
-jre-r [ EUR SPRPC2 WorkDone ] [ SPRPC1 WorkDone ] [ SPRPC2 WorkDone ]
[ GBP SPRPC1 WorkDone ] [ SPRPC2 WorkDone ] [ SPRPC1 WorkDone ]
-? -he [ HKD SPRPC2 WorkDone ] [ SPRPC1 WorkDone ] [ SPRPC2 Running ]
[X [ HUF SPRPC1 WorkDone ] [ SPRPC2 WorkDone ] [ SPRPC1 Running ]
-enabl [ INR SPRPC2 WorkDone ] [ SPRPC1 WorkDone ] [ InQueue ]
[ JPY SPRPC1 WorkDone ] [ SPRPC2 WorkDone ] [ InQueue ]
-da [ KRW SPRPC2 WorkDone ] [ SPRPC1 WorkDone ] [ InQueue ]
-disab [ MWN SPRPC1 WorkDone ] [ SPRPC2 WorkDone ] [ InQueue ]
[ NOK SPRPC2 WorkDone ] [ SPRPC1 WorkDone ] [ InQueue ]
[ NZD SPRPC1 WorkDone ] [ SPRPC2 WorkDone ] [ InQueue ]
-esa [ PHP SPRPC2 WorkDone ] [ SPRPC1 WorkDone ] [ InQueue ]
[ PLM SPRPC1 WorkDone ] [ SPRPC2 WorkDone ] [ InQueue ]
-dsa [ SEK SPRPC2 WorkDone ] [ SPRPC1 WorkDone ] [ InQueue ]
-agent [ SKK SPRPC2 WorkDone ] [ SPRPC1 WorkDone ] [ InQueue ]
[ TWD SPRPC1 WorkDone ] [ SPRPC2 WorkDone ] [ InQueue ]
[ USD SPRPC2 WorkDone ] [ SPRPC1 WorkDone ] [ InQueue ]
-10-Jan-2007 12:12:46 wingridjobdispatcher.util.WinGridJobDispatcherLogger logInf
-10-Jan-2007 12:12:49 wingridjobdispatcher.util.WinGridJobDispatcherLogger logSev
INFO: Work being sent by WJD to SPRPC1 [WPGBD3001484.UK.EUROPE.INTRANET:1616000]
WJN_PC => More work remaining. Trying to connect to WJN_PC [WPGBD3001630.UK.EURO
ROB_PC => More work remaining. Trying to connect to ROB_PC [WPGBD3001447.UK.EURO
RAH_PC => More work remaining. Trying to connect to RAH_PC [WPGBD3001529.UK.EURO
JTN_PC => More work remaining. Trying to connect to JTN_PC [WPGBD3001545.UK.EURO
SAM_PC => More work remaining. Trying to connect to SAM_PC [WPGBD3001481.UK.EURO
10-Jan-2007 12:12:49 wingridjobdispatcher.util.WinGridJobDispatcherLogger logSev
SEVERE: WJD 10-01-2007 12:12:49 : UnknownHostException. [WPGBD3001855.UK.EUROPE
10-Jan-2007 12:12:59 wingridjobdispatcher.util.WinGridJobDispatcherLogger logSev
SEVERE: WJD 10-01-2007 12:12:59 : UnknownHostException. [WPGBD3001855.UK.EUROPE

```

Screenshot 28: The WJD console displaying messages pertaining to WTC (5 WTCs running on WJN_PC, ROB_PC, RAH_PC, JTN_PC, SAM_PC are not listening for WJD and hostname WPGBD3001855.UK.EUROPE.INTRANET cannot be found)

4.6 WinGrid Job Dispatcher (WJD) Execution Completion

The WJD stops communicating with the WTCs when all the jobs have been completed successfully. For each WTC listed in the WinGrid parameter file (screenshot 19) the WJD starts a separate process. The CPU time allotted to WJD is shared between these different processes. In technical terms each such process is called a thread. Just like a teacher in a classroom allots her stipulated lecture time between multiple pupils, similarly the WJD shares CPU time between the different threads (processes), wherein each thread is actually the communication channel to a particular WTC running on a particular computer.

After all the three phases of processing ends the WJD stops interacting with the WTC by closing these communication channels. It then waits for all these processes (threads) to finish – just like a teacher may wait until all the students have left the lecture room – before informing the user that it is now ready to assimilate all the results returned by the different WTCs. This information is conveyed through a message box that appears on the task bar (screenshot 29).

```

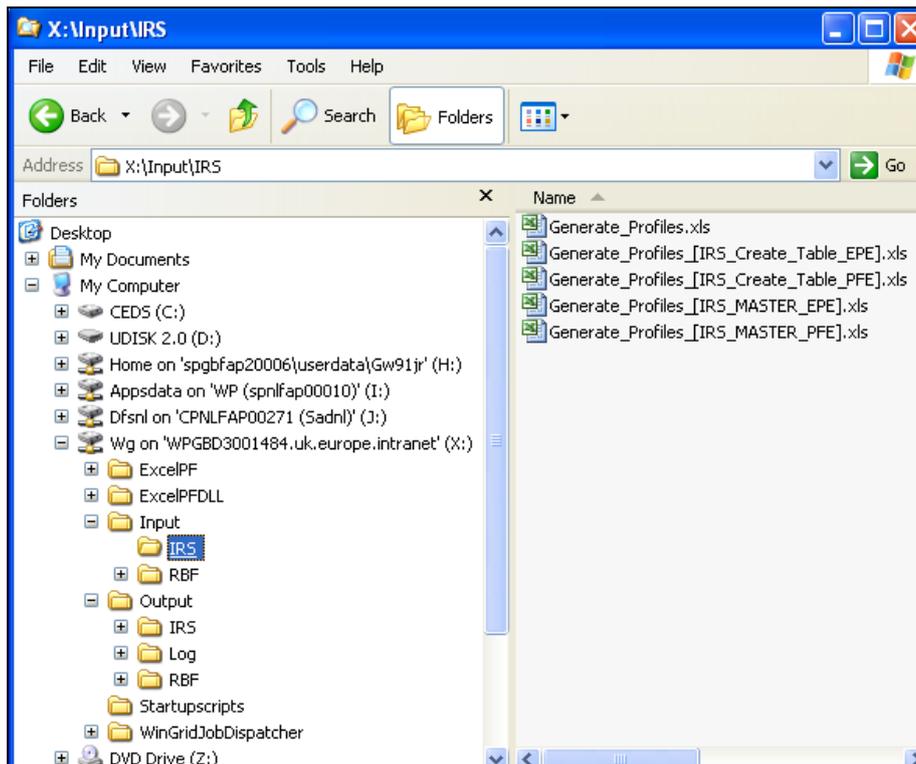
[ TWD E WorkDone ] [ C WorkDone ] [ F WorkDone ]
[ USD F WorkDone ] [ H WorkDone ] [ C WorkDone ]
[ ZAR A WorkDone ] [ H *WorkDone* ] [ B WorkDone ]
14-Jan-2007 18:48:55 wingridjobdispatcher.util.WinGridJobDispatcherLogger logInformation
INFO:
A GBP*AUD*INR null
H CZK*HKD*KRW*MKN*USD*ZAR null
E DKR*HUF*NZD*SGD AUD*CZK*HUF*NOK*SKK
F EUR*CAD*PHP*SKK CAD*DKK*INR*PHP*TWD
C CHF*JPY*NOK*PLN*SEK*TWD CHF*HKD*JPY*MKN*PLN*SGD
B null EUR*GBP*KRW*NZD*SEK*ZAR
WJD searching for work to assign 215-F: -----
14-Jan-2007 18:48:55 wingridjobdispatcher.util.WinGridJobDispatcherLogger logInformation
INFO: ##### Message EXITING THREAD. NO MORE JOBS WILL BE DISPATCHED [H] [215-F] @ 160
WJD searching for work to assign 211-B: -----
14-Jan-2007 18:48:56 wingridjobdispatcher.util.WinGridJobDispatcherLogger logInformation
INFO: ##### Message EXITING THREAD. NO MORE JOBS WILL BE DISPATCHED [B] [211-B] @ 160
WJD: Thread 211-B:60000 joined
14-Jan-2007 18:49:00 wingridjobdispatcher.util.WinGridJobDispatcherLogger logInformation
INFO: ----- Message EXITING THREAD. NO MORE JOBS WILL BE DISPATCHED [H] [217-H] @ 160
14-Jan-2007 18:49:00 wingridjobdispatcher.util.WinGridJobDispatcherLogger logInformation
INFO: ----- Message EXITING THREAD. NO MORE JOBS WILL BE DISPATCHED [A] [210-A] @ 160
WJD searching for work to assign 214-E: ----- ALL WORK HAS BEEN COMPLETED -----
14-Jan-2007 18:49:00 wingridjobdispatcher.util.WinGridJobDispatcherLogger logInformation
INFO: ##### Message EXITING THREAD. NO MORE JOBS WILL BE DISPATCHED [E] [214-E] @ 160
14-Jan-2007 18:49:01 wingridjobdispatcher.util.WinGridJobDispatcherLogger logInformation
INFO: ----- Message EXITING THREAD. NO MORE JOBS WILL BE DISPATCHED [D] [213-D] @ 160
WJD searching for work to assign 192.168.0.212: ----- ALL WORK HAS BEEN COMPLETED -----
14-Jan-2007 18:49:05 wingridjobdispatcher.util.WinGridJobDispatcherLogger logInformation
INFO: ##### Message EXITING THREAD. NO MORE JOBS WILL BE DISPATCHED [C] [192.168.0.212] @ 160
WJD: Thread 192.168.0.212:60000 joined
WJD: Thread 213-D:60000 joined
WJD: Thread 210-A:60000 joined
WJD: Thread 214-E:60000 joined
WJD: Thread 215-F:60000 joined
14-Jan-2007 18:49:06 wingridjobdispatcher.util.WinGridJobDispatcherLogger logInformation
INFO: ----- Message EXITING THREAD. NO MORE JOBS WILL BE DISPATCHED [G] [216-G] @ 160
WJD: Thread 216-G:60000 joined
WJD: Thread 217-H:60000 joined
14-Jan-2007 18:49:06 wingridjobdispatcher.util.WinGridJobDispatcherLogger logInformation
INFO: WJD is now collecting results.

```

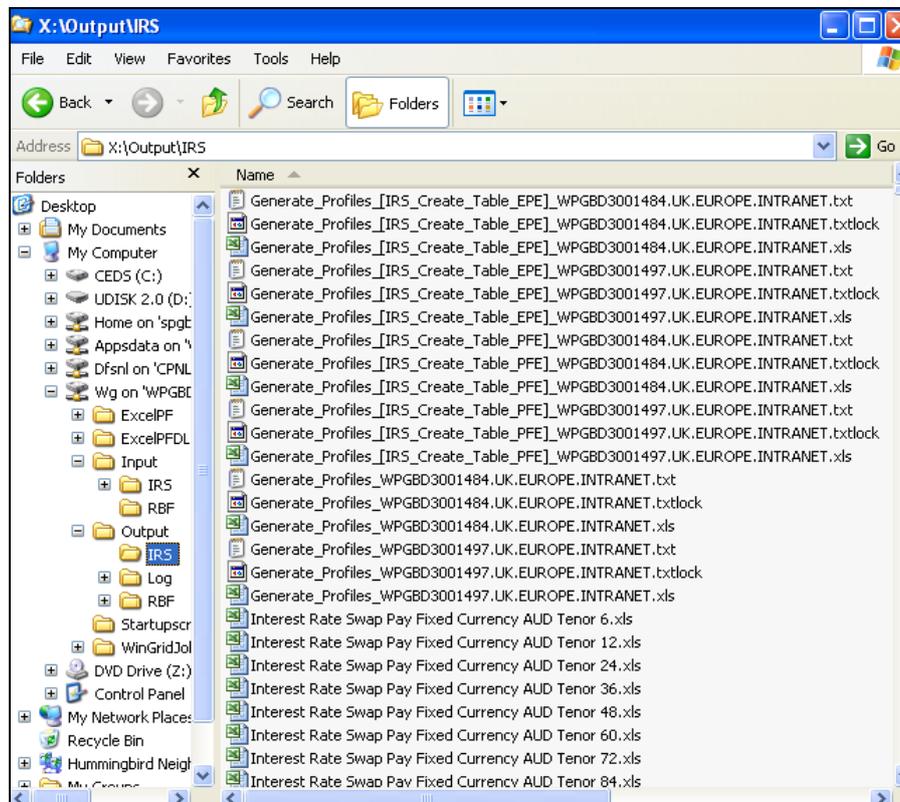
Screenshot 29: A WJD message box informs the user that all processing is complete and it is now ready to collate all the results returned by the WTCs

At this point all the processing is complete and all the data is in folder *X:\Output\IRS* or *X:\Output\RBS*, depending on whether WinGrid-enabled IRS or RBS simulation was executed. Both Phase 2 (creating EPE tables) and Phase 3 (creating PFE tables) of processing are dependent on Phase 1 (creating profiles). Analytics™ is called only in Phase 1. Phase 2 and Phase 3 of the processing actually imports data that is output by Analytics™ in the form of 1000s of text files containing risk calculations. Because there can be multiple WTC clients processing jobs in all the three phases therefore each client makes a copy of the original input file stored in either *X:\Input\IRS* or *X:\Input\RBS* (screenshot 30) and copies the same to either *X:\Output\IRS* or *X:\Output\RBS* respectively (screenshot 31). Each of these temporary output files are named by appending the WTC computer name (screenshot 19) to the original file name.

Thus, a computer with the name *WPGBD3001529.UK.EUROPE.INTRANET* would copy the file *Generate_Profiles_[IRS_Create_Table_EPE].xls* from *X:\Input\IRS* and place it in *X:\Output\IRS* with the filename *Generate_Profiles_[IRS_Create_Table_EPE]_WPGBD3001529.UK.EUROPE.INTRANET.xls*. This naming scheme allows each WTC to have its exclusive copy of the input file for all the three stages of processing.



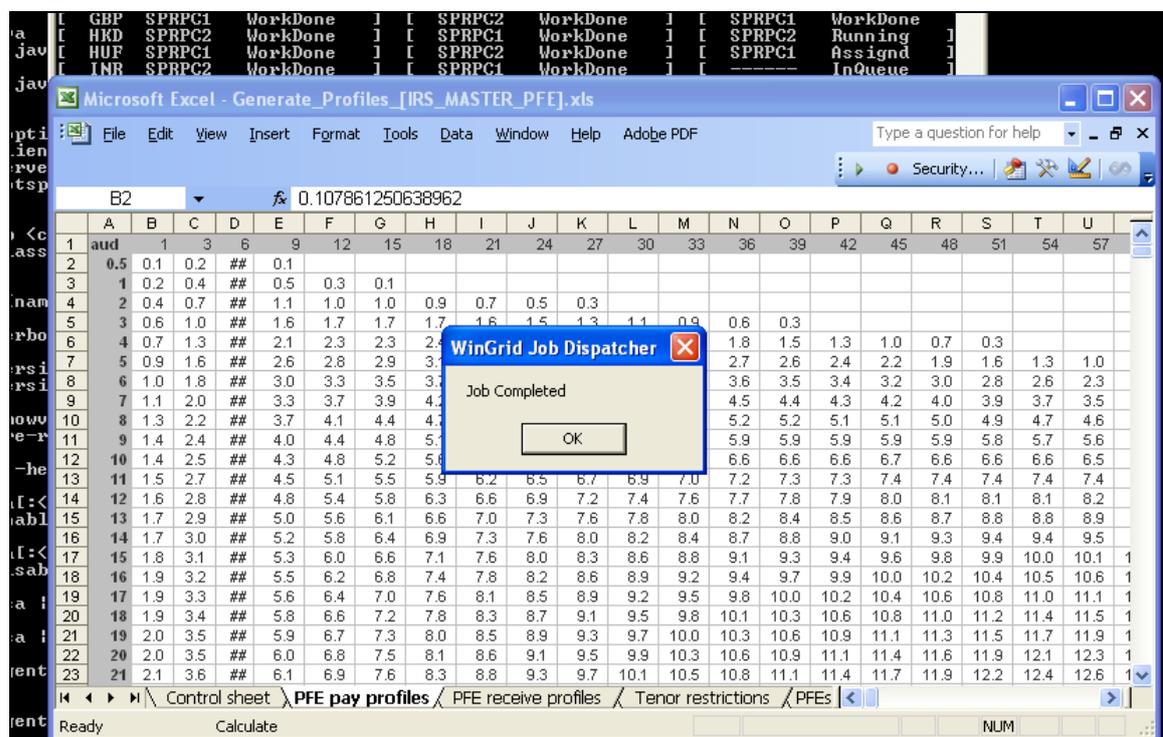
Screenshot 30: The input files for the IRS and RBS simulations are stored in directory X:\Input\IRS and X:\Input\RBS respectively



Screenshot 31: The output files for the IRS and RBS simulations are stored in directory X:\Output\IRS and X:\Output\RBS respectively

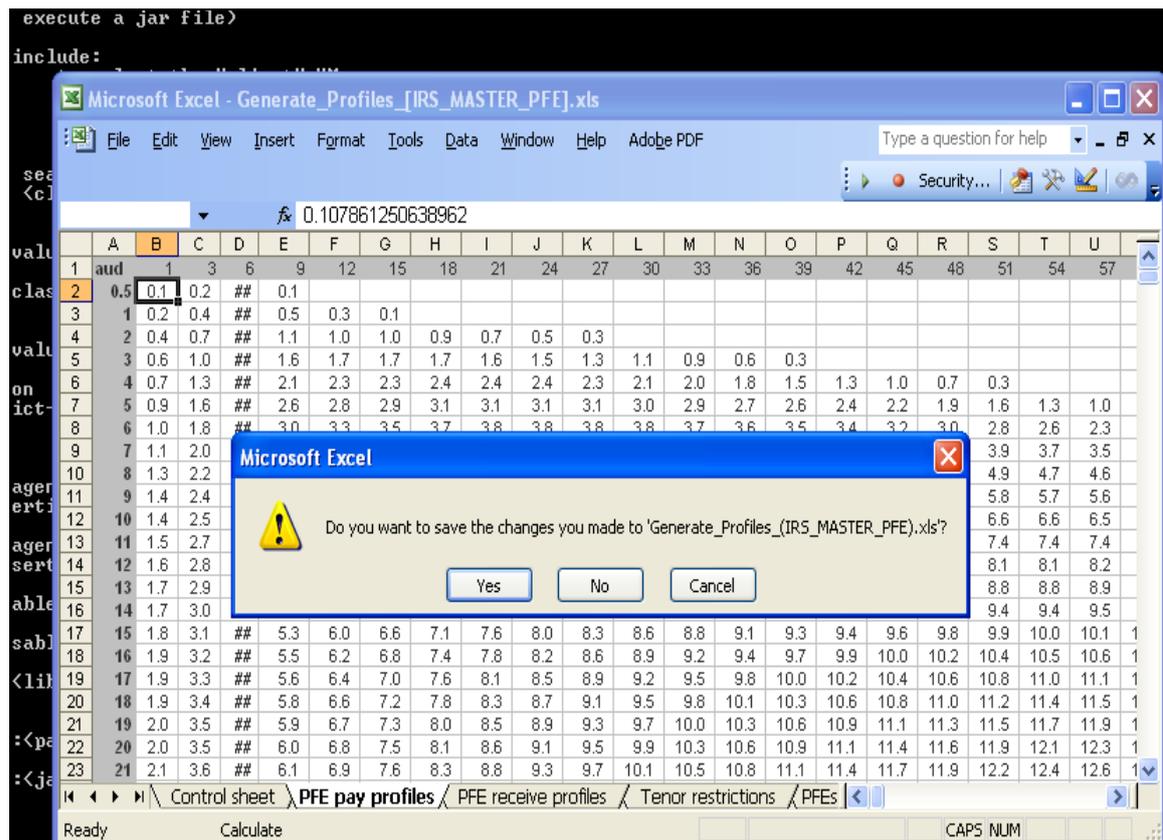
We now revert back to our earlier discussion on the procedure to start collecting results. The user should click OK in the WJD message box (screenshot 29) to signal WJD to start collating the individual *Generate_Profiles_[xxx_Create_Table_EPE]_yyy.xls* and *Generate_Profiles_[xxx_Create_Table_PFE]_yyy.xls* results returned by the WTC, wherein xxx is either IRS or RBF and yyy is the computer name (see screenshot 31). This process involves the WJD opening the *Generate_Profiles_[xxx_Master_EPE].xls* and *Generate_Profiles_[xxx_Master_PFE].xls* (see screenshot 30), wherein xxx is either IRS or RBF, and copying results from the temporary WJD result files.

After WJD has completed this process the user is informed of the same through the “Job Completed” message box (screenshot 32) that appears on the taskbar.



Screenshot 32: WJD informs the user that the result collection is over by displaying a “Job Completed” message box.

The source MASTER_PFE and the MASTER_PFE files for both IRS and RBF simulations are read only. The user will therefore have to save the file to a different location. If the user tries to close these MASTER files without saving the contents then Excel™ prompts the user with the “Save Changes?” dialogue box (screenshot 33).

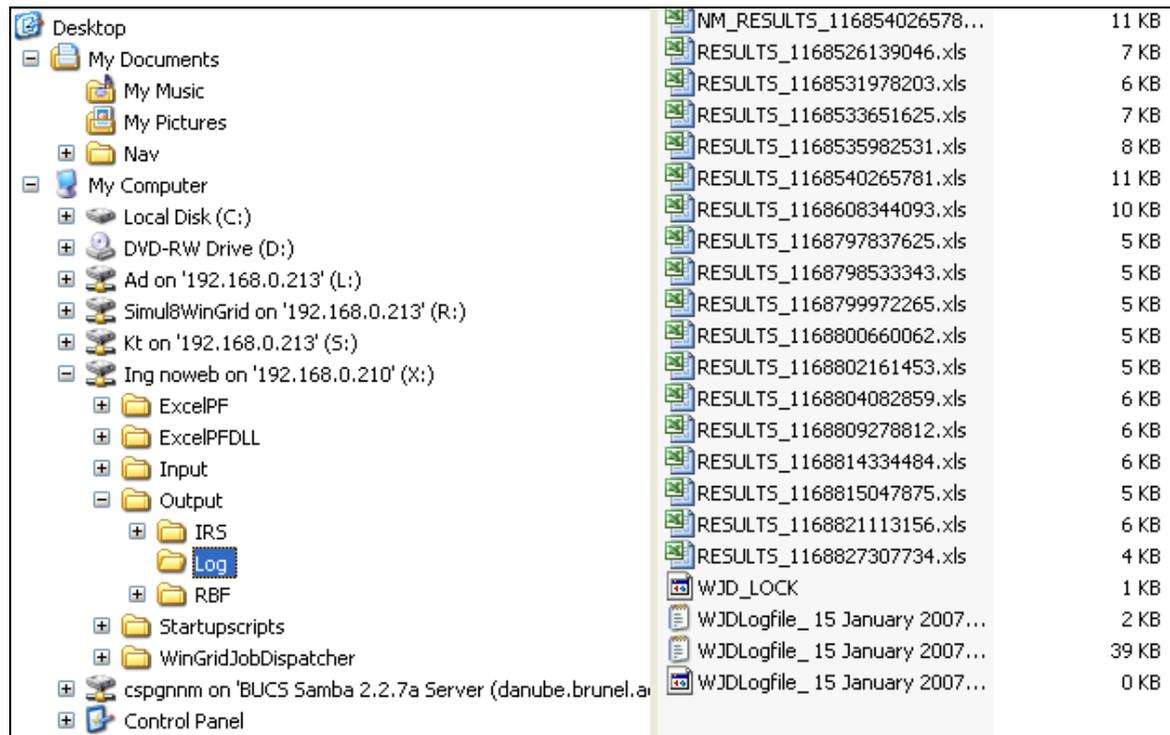


Screenshot 33: The “Save Changes?” dialogue box is displayed if the user tries to close the MASTER files without first saving the data. The MASTER files are read only. Therefore these files have to be saved at a different location

5. PERFORMANCE RESULT COLLECTION

How long does it take to run the IRS and RBS simulations? We can get an answer to that by finding the performance results that are stored in directory *X:\Output\Log* (screenshot 34). The results are stored in the Excel™ file format. The name of the file is kept unique by appending the time the file was created (in milliseconds from 1st January 1970 – Unix concept of time) to the String “RESULTS_”.

The information recorded include the time when the program was started, time a work unit was dispatched to a WTC and the time the results were returned, time of result collection and the time the program finally completed execution. Using this data a lot of useful information can be derived, for example, the total number of units processed by different WTCs, the time taken to complete the different phases of processing, the number of times the same work unit was sent before it was successfully processed and so on. Screenshot 35 shows this tab-delimited results Excel™ file.



Screenshot 34: All results are stored in directory X:\Output\Log. This directory also stores the WJD log files (section 4.4)

	A	B	C	D	E	F	G	H
1	1168823425375	START	START	START	START			
2	1168823445125	SENT	PHASE1	CAD	C			
3	1168823447484	SENT	PHASE1	CZK	B			
4	1168823461296	SENT	PHASE1	GBP	H			
5	1168823465765	SENT	PHASE1	EUR	G			
6	1168823476515	SENT	PHASE1	HUF	E			
7	1168823504234	SENT	PHASE1	JPY	F			
8	1168823572796	SENT	PHASE1	CZK	B			
9	1168823599421	SENT	PHASE1	CAD	C			
10	1168823672750	SENT	PHASE1	CAD	C			
11	1168823676000	SENT	PHASE1	CZK	B			
12	1168823688953	SENT	PHASE1	EUR	G			
13	1168823689078	SENT	PHASE1	CAD	C			
14	1168823697078	SENT	PHASE1	EUR	A			
15	1168823708921	SENT	PHASE1	KRW	G			
16	1168823777468	SENT	PHASE1	GBP	H			
17	1168823819593	SENT	PHASE1	JPY	F			
18	1168823993875	RECV	PHASE1	CAD	C			
19	1168823993890	SENT	PHASE1	NOK	C			

Screenshot 35: The RESULTS file stores data collected by the WJD during processing

6 WTC AND WJD ERRORS

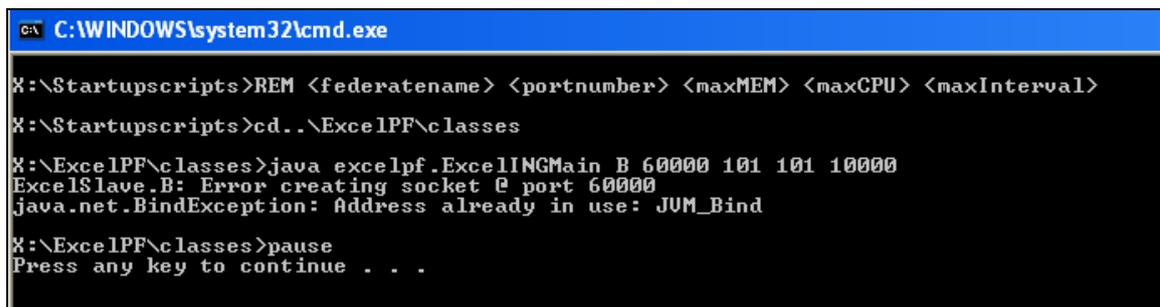
6.1 WinGrid Thin Client (WTC) Errors

This section presents some of the error conditions encountered by WTC during processing. Some of these errors can be easily overcome while others, unfortunately, will require you to restart WTC.

WinGrid architecture is designed to interface with third party applications. These third party applications have their own program code and are executed in a separate part of the computer memory. WinGrid can only call COM functions / methods (read as tasks) defined by these applications and expect them to be executed properly. If there is a problem executing these COM methods then an error occurs and the WTC will need to be restarted. However, this will not affect processing of the WJD jobs as long as there is even one WTC running properly.

6.1.1 Multiple Instances of WTC Running

If the user runs more than one WTC process in one computer then an error condition will occur and the new WTC process that was started will exit. The WTC process running previously in the same computer will not be affected. The user will be shown a JVM_Bind error before exit (screenshot 36).



```

C:\WINDOWS\system32\cmd.exe
X:\Startupscripts>REM <federatename> <portnumber> <maxMEM> <maxCPU> <maxInterval>
X:\Startupscripts>cd..\ExcelPF\classes
X:\ExcelPF\classes>java excelpf.ExcelINGMain B 60000 101 101 10000
ExcelSlave.B: Error creating socket @ port 60000
java.net.BindException: Address already in use: JVM_Bind
X:\ExcelPF\classes>pause
Press any key to continue . . .

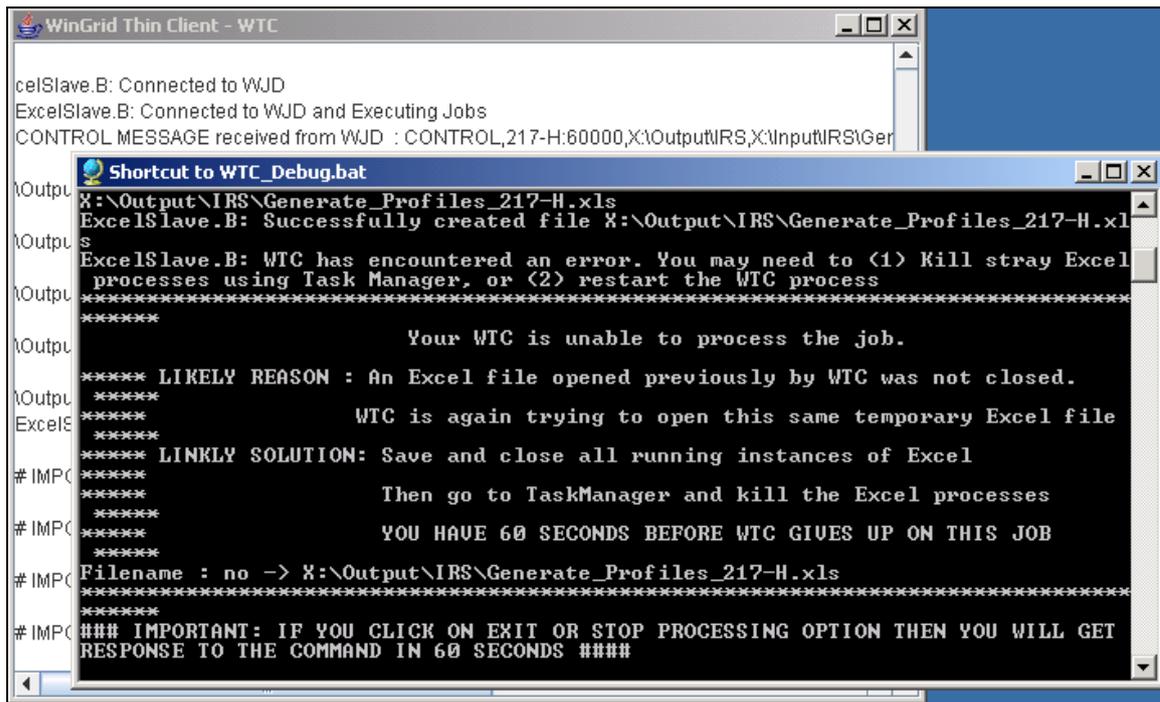
```

Screenshot 36: More than one instance of WTC has been started in the same computer

6.1.2 WTC was Closed Forcibly

The normal procedure of closing a WTC is through the *Exit* option in the WinGrid menu that is accessible from the System Tray (screenshot 15). Once this *Exit* option is evoked the WTC takes 5 seconds to complete the necessary housekeeping (example, closing Excel™ files that have been previously opened) before exiting.

If the WTC was forcibly closed by using the (X) option present in the top right-hand corner of either the white WTC window or the black command window (this is in case the WTC was started using *WTC_Debug.bat*), then an error may occur when the WTC is restarted again. The error message is self-explanatory (screenshot 37). It tells you that WTC is trying to open a file that had already been opened earlier, but not closed. It is giving you 60 seconds to close this file using Windows Task Manager, after which this job will be reassigned.



Screenshot 37: More than one instance of WTC has been started in the same computer

During these 60 seconds you will see the red WTC error icon  in the System Tray. This suggests that WTC has encountered an error and is expecting user intervention for rectification of the problem. After these 60 seconds have passed then the job is again in the WJD queue. If other WTCs are not busy then this job is allocated and processed elsewhere.

If you are busy with your work and you see the red WTC error icon then you need not do anything. Processing of jobs will continue in other nodes. You may also notice that the red icon disappears for 10-15 seconds and then reappears again. This is because the same job (more specifically, a job in the “same phase” that requires the same file to be opened –the file the WTC could not successfully open the previous time around) has again been dispatched to your WTC. ***If you are busy with your work please ignore this behavior too. However, when you get a chance do check the error message and close any stray Excel™ processes running on your PC using the Windows Task Manager (you need not close the WTC process to do this). This will enable the WTC to again process jobs in your PC.***

6.1.3 COM Failure

A COM failure (screenshot 38) occurs because the third-party software (Excel™ and Analytics™ in our case) invoked by WinGrid using the third-party defined COM interface had failed to do a job. Once a COM failure has occurred you will see the red WTC icon on the Windows System Tray. ***If you are busy then you need not do anything.***

To again enable your PC to process jobs you may be required to do the following:

- (1) Exit the WTC application.

6.1.6 Multiple WinGrid Icons in Windows System Tray

Many a times you will notice multiple WinGrid icons in the System Tray area of windows (see screenshot 41). Thankfully, only one of these icons is actually active. The rest of the inactive icons are displayed probably because the System Tray has not refreshed. The active icon is the one which allows the user to access the WinGrid menu (screenshot 15) through right click.



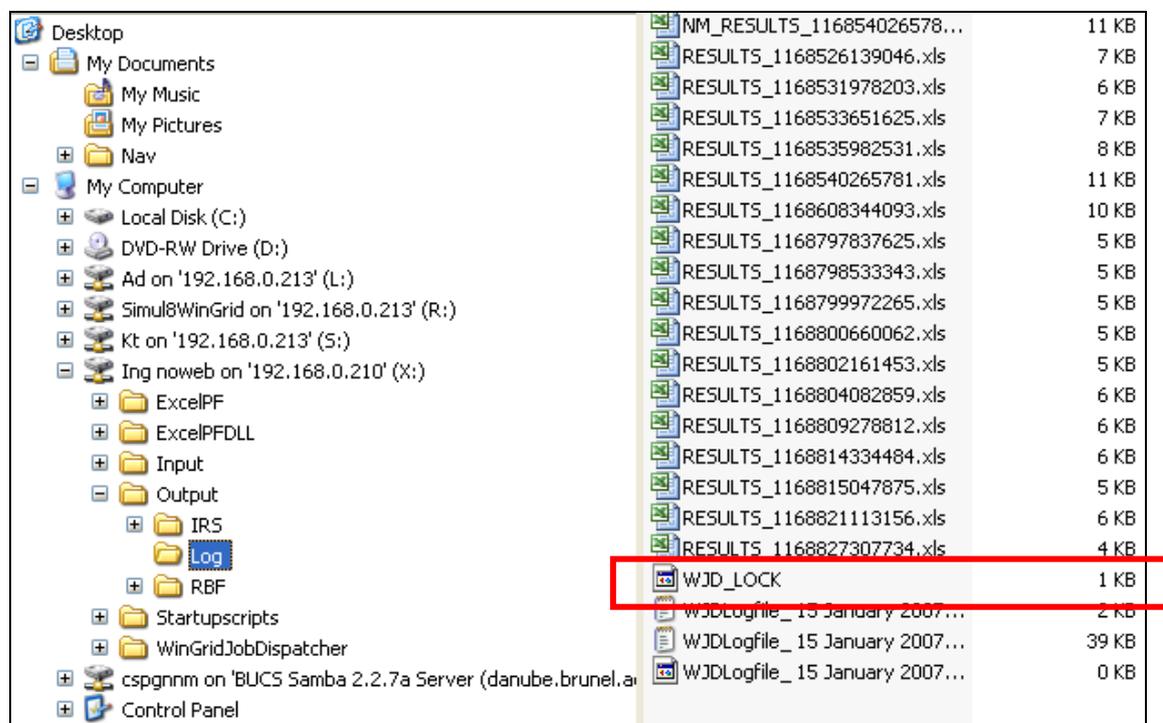
Screenshot 41: Multiple WinGrid icons in System Tray

6.2 WinGrid Job Dispatcher (WJD) Errors

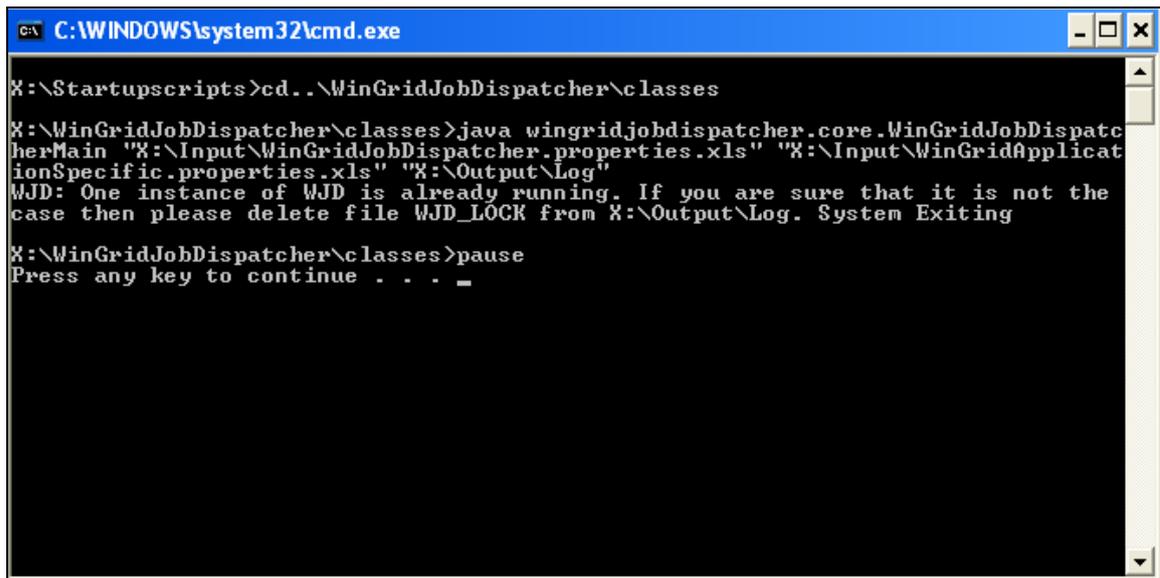
6.2.1 WJD already Running

This is not strictly an error. There are checks in the system which makes sure that only one instance of WJD can be started any one time. This check is implemented by creating a temporary file called *WJD_LOCK* in the *X:\Output\Log* directory (screenshot 42) when the WJD is started. This file is again deleted when the WJD process has completed execution.

When a WJD process is started in any computer it first checks to see whether *WJD_LOCK* is present in *X:\Output\Log* directory. If yes, then the user encounters the message shown in screenshot 43.



Screenshot 42: The WJD_LOCK file is created in X:\Output\Log directory

A screenshot of a Windows command prompt window titled "C:\WINDOWS\system32\cmd.exe". The prompt shows the user navigating to a directory and running a Java command. The output indicates that one instance of the WinGrid Job Dispatcher (WJD) is already running, and the system is exiting. The user then enters a pause command.

```
C:\WINDOWS\system32\cmd.exe
X:\Startupscripts>cd.. \WinGridJobDispatcher\classes
X:\WinGridJobDispatcher\classes>java wingridjobdispatcher.core.WinGridJobDispatcherMain "%:\Input\WinGridJobDispatcher.properties.xls" "%:\Input\WinGridApplicationSpecific.properties.xls" "%:\Output\Log"
WJD: One instance of WJD is already running. If you are sure that it is not the case then please delete file WJD_LOCK from X:\Output\Log. System Exiting
X:\WinGridJobDispatcher\classes>pause
Press any key to continue . . . _
```

Screenshot 43: WJD informing user that one instance of WJD is already running

If the user is sure that it is not the case and no other WJD is presently running, and the WJD_LOCK was not automatically deleted by the previous WJD execution because it was forcibly closed, then the WJD_LOCK file may be safely deleted. Restarting *WJD.bat* will now start the WinGrid Job Dispatcher without any error.

7. MISCELLANEOUS

7.1 Acknowledgements

The author would like to thank **Robert Watson** from XXX for his contribution in the development of this new system. He has been most patient with the author and has extended his help and support towards implementation, testing and installation of the WinGrid software.

Thanks are also due to **Jonathan Berryman** (XXX), my PhD supervisor **Dr. Simon Taylor** (Brunel University), **Rahul Talwalkar** (XXX, Singapore), **Meeta Talwalkar** (Brunel University), **Jasbir Singh Heer** (XXX) and **Andrew McFadyen** (XXX).

7.2 Contact Information

Navonil Mustafee

Research Student
Centre for Applied Simulation Modelling (CASM)
School of Information Systems, Computing & Mathematics
Brunel University, Uxbridge, Middlesex UB8 3PH
Telephone : 01895265727
Email 1: navonil.mustafee@brunel.ac.uk