GSM 935291 이하빈. Habin Lee. Knowledge-based Approach for Flexible Workflow Management System. 유연한 워크플로우 자동화 시스템을 위한 지식 기반 접근법에 관한 연구. Graduate School of Management. 1998. 148p. Advisor Prof. Sung Joo Park. Text in ENGLISH.

ABSTRACT

Today's business environments are characterized by dynamic and uncertain environments. In order to effectively support business processes in such contexts, workflow management systems must be able to adapt themselves effectively. In this dissertation, the workflow is redefined in concept and represented with a set of business rules. Business rules play a central role in organizational workflows in context of cooperation among actors. To achieve business goals, they constrain the flow of works, use of resources, and responsibility mapping between tasks and actors using role concept. Business rules are explicitly modeled in the Knowledge-based Workflow Model (KWM) using frames.

To increase the adaptability of workflow management system, KWM has several distinctive features. First, it increases expressiveness of workflow model so that exception handling rules and responsibility mapping rules between tasks and actors as well as task scheduling rules are explicitly modeled. Secondly, formal definition of KWM enables one to define and to analyze correctness of workflow schema. Knowledge-based approach enables more powerful analysis on workflow schema including checking consistency and compactness of routing rules as well as terminality of a workflow. Thirdly, providing change propagation mechanism which assures correctness of workflow after the modification of workflow schema increases adaptability. Change propagation rules for the modification primitives are provided to manage workflow evolution. On the other hand, metarules that control rules in KWM are used to handle

exceptions that occur in a running workflow instance. Workflow participants can easily change workflow schema of a workflow instance with the support of extra rules and a metarule.

Based on KWM, K-WFMS (Knowledge-based WorkFlow Management System) has been implemented in client/server architecture. Inference shell of knowledge-based systems is employed for enactment of business rules and integrated with database systems. From a real application based on the KWM architecture, it has been shown that system performance can increase notably by reducing the number of rules and facts that are used in the course of workflow enactment.

Table of Contents

List of Figures

List of Tables

Chapter 1. Introduction

1.1 Background and Research Motivation

Although efficiency is still a central business goal, flexibility and changeability are becoming more and more prominent in business processes (Scott-Morton, 1994). Furthermore, increasing agility of an organization is considered as a critical success factor in a competitive environment of continually and unpredictably changing customer opportunities (Goldman et al., 1995). Agile organizations are apt to frequently change their business processes to satisfy fluctuating customers needs. Development of information system that supports business processes should be flexible to adapt to the changing business processes.

The workflow management system (WFMS) is considered as a key technology that automates business processes. The WFMS should be highly adaptive to changes on business processes in agile organizations. In a WFMS, business processes are represented using workflow model which has three main constructs; routes, rules, and roles (Marshak, 1994). Routing construct represents task sequences and a role represents one who is responsible for a task. Based on organizational model, a role can be defined with actor's department, position, and skills, etc. Rule is used to define routing and role constructs. It enables to define conditional or exceptional routings and conditional assignment of tasks to actors through role constructs. An adaptive WFMS should be flexible enough to handle the changes on these three constructs.

Some WFMSs are flexible (Reichert and Dadam, 1998; Casati et al., 1998; Dellen et al., 1997) in the sense that they provide adaptability for the changes on routing constructs such as adding or deleting tasks, or changing task sequences. These systems, however, do not provide capability to handle changes on the organizational structure

and business rules. The role definition can be affected by the changes on the organizational structure such as the merger and abolition of departments, change on the position hierarchy, and creation of temporal task force, etc. In an organization, there may exist heterogeneous departments and actor types, different routing conditions according to the types of actors, and flexible role instances that are responsible for a task. The rules change frequently due to BPR (Business Process Reengineering), empowerment, or restructuring. The existence of exceptional rules that may be applied for special workflow instances aggravates the complexity of rule management. The business rules can be directly affected by the changes on the routing and role constructs, of which the effects can also be cascaded, i.e. change on a business rule can affect other related business rules. Thus providing a change propagation facility for the changes on the three constructs is an inevitable component of adaptive WFMSs.

One of the main reasons that make WFMS inflexible can be found from the approach for describing workflows. There are three fundamentally different approaches: (1) communication-based models developed from the language/action perspective of Winograd and Flores (1986); (2) activity-based models that view workflow as a chain of production activities; and (3) object-based models that view workflow as a “complex web of interactions” among highly skilled knowledge workers. The first two approaches have been criticized for their inability to realistically and flexibly represent how work is performed (Lubich, 1995). The difficulty seems to be that the communication-based approaches only model commitments among humans (by representing conditions of satisfaction in the performance of work), and the activity-based approaches only model inputs, transformations, and outputs of a work process (while ignoring “human processes” in the performace of work). Thus work is viewed only as a sequence of

business functions (Sachs, 1995). Unfortunately, the most prevalent problem in workflow management systems (that are based on pre-planned routines) is that in work environments with task uncertainty, task allocation and coordination cannot be planned in advance (Hurts and de Greef, 1994). Thus the use of such proedures has resulted in a loss of flexibility to cope with exceptions that characterize “real-life” tasks (Lubich, 1995). The object-based model views an organization as a knowledge-processing system which acquires, processes, stores, and disseminates knowledge to its environment. Workflows modeling is to identify organizational resources including human and computers that are represented as objects or agents and to represent their interactions. Ganapathy (1996) proposed an approach which redefines the workflow concept so that it is viewed as the interactions among problem-solving entities in an organization. He subscribes to the view that an organization is a knowledge-processing system, and that the interactions among knowledge workers constitute knowledge-management episodes. The workflow concept is more closely aligned with the tacit view of work, i.e., it refers to activities such as problem-finding and problem-solving that occur in the performance of work. With object-based model, the role concept is highlighted because objects (or agents) interact each other with assessed role. However, business rules are hidden into the objects (or agents), which results in difficulty of rule management.

For an ideal adaptive WFMS for agile organizations, workflow models need to be enhanced in the following aspects:

- Expressiveness : It should provide constructs to represent conditional mapping relationships between roles and actors based on organizational model as well as complex business rules including exceptional rules.

- Model verification : It should allow analysis that assures the correctness of workflow specification including checking the occurrence of inconsistent, redundant, and incomplete business rules as well as non-terminality of processes.
- Change management : It should allow easy development of propagation mechanism against changes on the organizational structure and business rules as well as organizational procedures to assure the correctness of workflow model. Furthermore, exception handling mechanism in instance-level should be provided.

In this dissertation, a knowledge-based approach for workflow modeling and enactment is proposed. We define workflow as a set of business rules. This view is based on that an organization can be described with business rules. Business rules are appropriate to represent organizational context or policies (Ong and Lee, 1996). Usually, the execution order of tasks and responsible actors for the tasks are predefined through business rules in an organization. Furthermore, the existence of exceptional rules reflects culture of the organization.

The advantage of the approach proposed in this dissertation is as follows:

(1) Workflows can be defined under organizational context. Provision of rule modeling constructs enhances the expressiveness of workflow model, which enables complex business rules including exceptional rules in workflows.

(2) Adaptability of WFMS is increased. One of main issues for adaptable WFMS is providing a mechanism that permits workflow participants to customize the workflow schema according to the situations. Rule-based approach enables dynamic interpretation of workflow schema, i.e., different version of workflow schema can coexist and can be selectively fired for special workflow instances. This reduces the

burden having additional mechanism for handling instance level exceptions.

(3) Various kinds of workflow verification are possible. Conventional workflow verification is focused on the checking terminality of workflows. Knowledge-based approach enables consistency and compactness of rules as well as terminality of workflows.

1.2 Scope and Objectives of the Research

In this dissertation, KWM (Knowledge-based Workflow Model) is designed as an adaptive workflow model and K-WFMS (Knowledge-based Workflow Management System) is implemented based on KWM. As an adaptive workflow model, KWM should have following desirable features: (1) the expressive power of business rules of KWM are improved using knowledge-based approach to represent complex and heterogeneous business rules, (2) properties that assure correctness of KWM are proposed which can be analyzed using a rule verification technique, (3) management of organizational changes in KWM can be easier due to change propagation mechanism. Dependencies between modeling constructs are explicitly represented in KWM, and organizational changes that affect routes, rules, and roles in a workflow are propagated to corresponding constructs using the dependencies to assure the correctness of KWM.
The objectives of the research can be summarized as follows:

(1) Development of an workflow model

We develop a knowledge-based workflow model which adopt rule as a major modeling construct. The conditional routing, assignment of tasks to actors using role

concept, and exception handling for special workflow instance are represented as If-Then rules. KWM needs a formal foundation for self-complacency, accurate communication, extension and modification of the model, and formal comparisons with other frameworks. We develop a set-theoretic formalism of KWM.

(2) Development of an approach for workflow verification

Verification of a model is necessary to assure consistent, complete, and compact workflow modeling. In this thesis, soundness property of KWM is formally defined. Some properties that assure sound KWM are defined, and analysis techniques that can be used to verify the soundness property are developed.

(3) Development of a change propagation mechanism

A change propagation mechanism assures a sound workflow model against changes on the modeling constructs of KWM. In this research, dependency-based change propagation mechanism is developed. Dependencies between modeling constructs are represented as predicates. Based on the dependency predicates, propagation rules that assure soundness of workflow are proposed. Furthermore, an exception handling mechanism in instance-level is provided.

(4) Designing and implementation of a WFMS

Based on KWM, we design and implement K-WFMS. It has client/server architecture. Rules in KWM are translated into executable rule language (CLIPS), and are chained to schedule tasks and assign tasks to actors based on organizational facts that are extracted from organizational database. To help easy management of KWM, a

GUI (Graphic User Interface)-based model editor is developed.

1.3 Organization of the Dissertation

This dissertation is composed as follows. Chapter 2 reviews related research. The basic concept of WFMS is reviewed, and the relationship with organizational information systems (OIS) is characterized. Furthermore, researches on workflow modeling approach, workflow verification, and change management in WFMS are reviewed in chapter 2.

Chapter 3 describes the detail of KWM. Basic principles, modeling constructs, and formal definitions of KWM are addressed. An illustrative example is used to prove validity of KWM.

Chapter 4 defines properties for the soundness of KWM. Based on the soundness property, analysis techniques for checking the properties are described.

Chapter 5 presents a change propagation mechanism for KWM. Changes are classified as schema-level and instance-level changes. Propagation rules for schema-level changes are proposed using dependency predicates that are generated from workflow designer specified workflow schema. Instance-level exceptions are handled using metarule cocept. Propagatin rules that generate metarules for each exception are provided.

Chapter 6 describes on the implementation of K-WFMS. The overall architecture of K-WFMS, implementation strategy, and characteristics of each module of K-WFMS are described.

Finally, the contribution of the research and future research directions are discussed

in chapter 7.

Chapter 2. Literature Review

In this chapter, some researches are reviewed that are related with KWM. In section 2.1, the basic concept of WFMS is reviewed. The relationship between office information system and WFMS is reviewed in section 2.2. The workflow models proposed in literature are reviewed in section 2.3, and researches on workflow verification and change management of workflow model are reviewed in section 2.4 and section 2.5, respectively. Lastly, researches on business rule modeling in information system development are reviewed in section 2.6.

2.1 WFMS

Workflow Management Coalition (WfMC: 1994) defines a workflow management system as “a system that completely defines, manages and executes ‘workflows’ through the execution of software whose order of execution is driven by a computer representation of the workflow logic”. As a cooperative information system, WFMS is considered as a key technology that automates business processes.

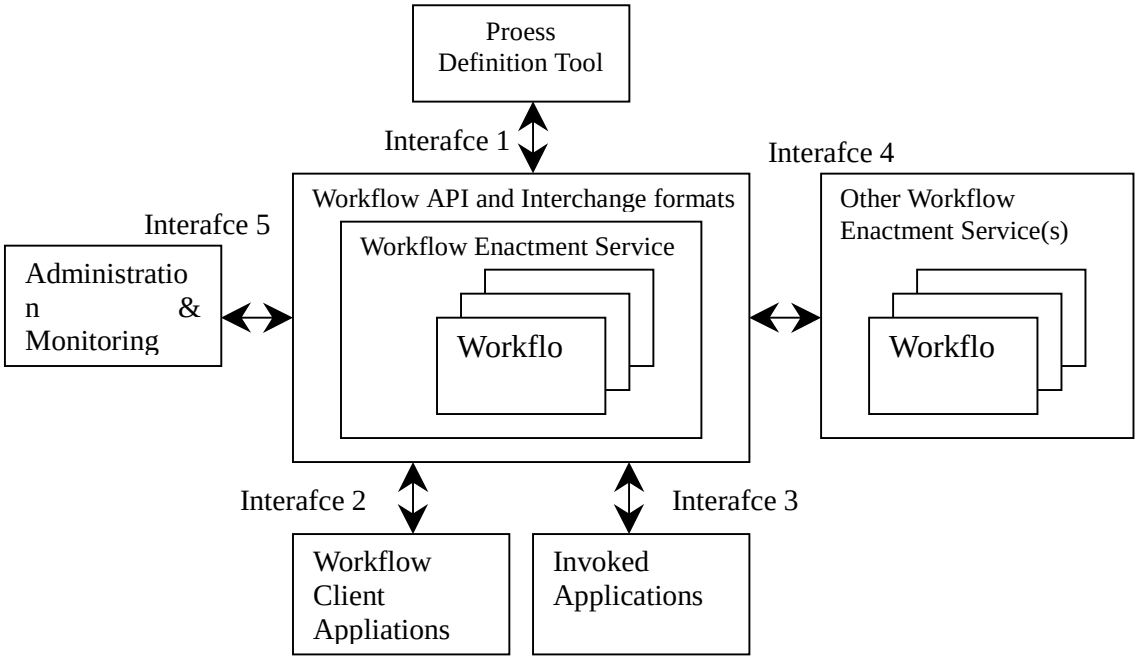
A WFMS consists of two basic components: a workflow modeling component and a workflow execution component (Ellis and Nutt, 1993). The workflow modeling component enables administrators, users, and analysts to define procedures and activities, analyze and simulate them, and allocate them to people. The execution

component is referred to as the “workflow engine”. The workflow engine routes artifacts among the tasks according to predefined routing conditions, and assigns tasks to actors using organizational roles. Handling exceptions that are caused by failures of workflow transaction or the occurrence of special workflow instance which needs special treatment is another important functionality of workflow engine. Lastly, workflow engine automatically launches an application system that supports execution of assigned task.

Abott and Sarin (1994) proposed three dimensions to classify WFMSs available on the market.

- Design versus runtime. Design-oriented workflow products emphasize the creation. Modeling, analysis, and simulation of workflow processes. Runtime workflow products are engines that provide generic routing and tracking services to applications.
- Mail- versus database-driven. Mail-driven workflow systems build on electronic mail as the basic underlying mechanism for routing and presenting work to users. Database-driven workflow system use underlying database technology to drive the process. Routing and status information is stored in tables that are queried by clients.
- Document- versus process-oriented. Document- (or data-) oriented workflow systems associate routing information with the particular data objects being worked on. Folder management and image routing systems fall into this category. They are good for handling manual, paper-based procedures electronically. Process-oriented workflow systems model the work process as a sequence of steps. Data objects are attached to steps in the process, but different objects can be routed at different steps in the process.

Figure 2.1 illustrates the major components and interfaces within the workflow architecture (WfMC, 1994). The workflow enactment service provides the run-time environment in which process instantiation and activation occurs, utilising one or more



[Figure 2.1] Workflow Reference Model (cited from WfMC(1994))

workflow management engines, responsible for interpreting and activating part, or all, of the process definition and interacting with the external resources necessary to process the various activities. The interface between the modeling and definition tools and the runtime workflow management software (interface 1) is termed the process definition import/export interface. The nature of the interface is an interchange format and API calls, which can support the exchange of process definition information over a variety of physical or electronic interchange media. Interface 2 defines communications between workflow enactment service and workflow client applications. The communications are for process control function, process status function, worklist/workitem handling function, and process supervisory functions etc. Interface 3 is for handling workflow enabled applications. Interface 4 is to handle the information and control flows between heterogeneous workflow systems. Lastly, interface 5 is a common interface standard for

administration and monitoring functions which will allow one vendor's management application to work with another's engine(s). This will provide a common interface which enables several workflow services to share a range of common administration and system monitoring functions.

2.2 Office Information System and WFMS

The researches on the office information system (OIS) in the early eighties contributed to the development of WFMS. OIS tried to address the automation of daily tasks performed by office workers. The development of these systems started in research laboratories under the basic assumption that office tasks are structured.

Bracchi and Pernici (1984) have classified office conceptual models which are the main component of OIS, on the basis of the fundamental elements that they take into consideration, into the four categories: data-based models, process-based models, agent-based models, and mixed models. Data-based models group data into *forms*, which are similar to paper forms in the traditional office. Types of data and the operations on data (storage, retrieval, manipulation, transmission) are the basic elements of these office conceptual models. Office activities are then seen as a series of operations on data. OFFICETALK-ZERO (Ellis and Nutt, 1980), OMEGA (Barber 1983), OFFIS (Konsynski et al., 1982), and OBE (Zloof, 1982) are the examples of data-based model. Process-based models analyze and describe office work by looking at different activities performed concurrently by the users and the system. The goal of process-based models is that of representing office activities in a coordinated way: thus the approach is not founded (as in data-based models) on operations performed by single users, but instead

on an integrated vision of all the activities performed in an office in order to execute certain tasks, with the purpose of a general control of office work. SCOOP (Zisman, 1978), ICN (Cook, 1980), OAM (Sirbu, 1981), OSL (Hammer and Kunin, 1980), and Ticom-II (Bailey et al., 1983) are the examples of process-based model. Agent-based models model an office from the viewpoint of the functions performed by active elements of the office environment (the agents). It describes the office by associating to the different agents a set of functions (i.e., the different roles that they take in performing their tasks, the domain within which they are authorized to act, and the set of relationships that link them to other agents). The example of agent-based model is Structural Model (Aiello et al., 1984). Lastly, mixed models explicitly assume more that one type of element as the basis for system specification, and define the relationships among these elements. OFS (Tsichritzis, 1982), DOMINO (Victor and Sommer, 1991), IML (Richter, 1981), OPAS (Lum et al., 1982), OFFICETALK-D (Ellis and Bernal, 1982), and SOS (Bracchi and Pernici, 1983) are the example of mixed-model.

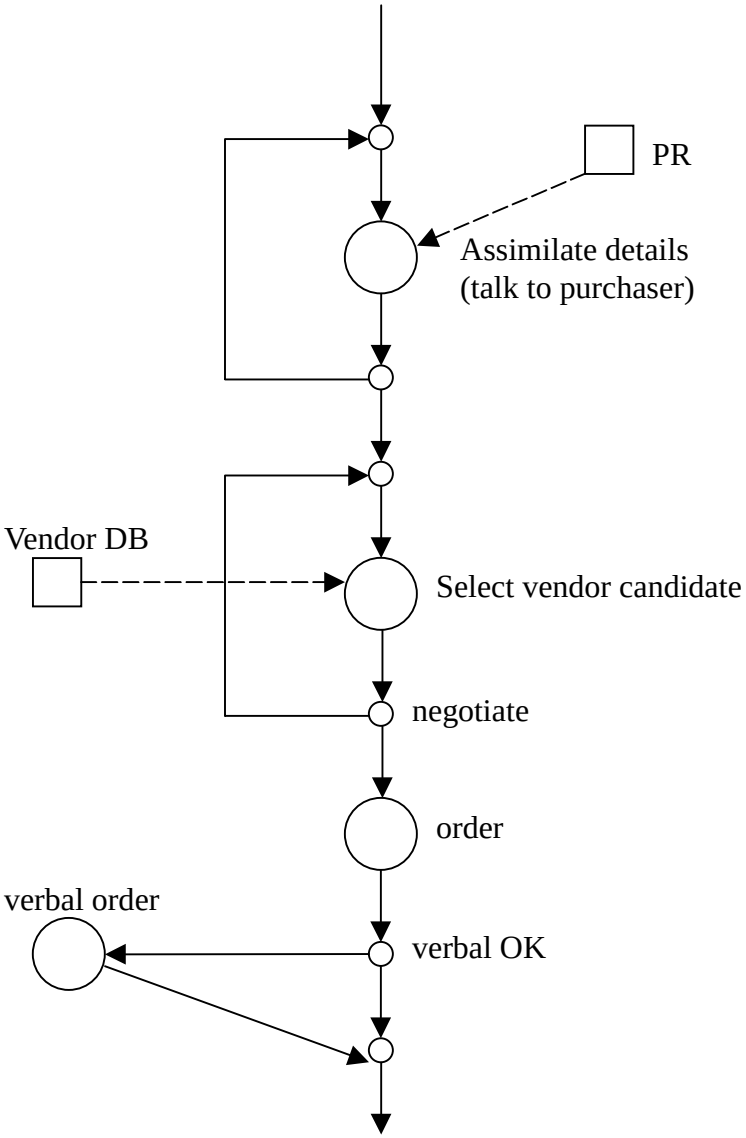
In the beginning of the nineties, these research prototypes, which never moved out of the research labs in to offices and organizations before, were either directly adopted by industry or strongly influenced the design of WFMS. Although there exists a significant overlap between WFMS and OIS, three significant differences exist between them (Abott and Sarin, 1994) (Mahling et al, 1995). At first, the emphasis in workflow management is on using computers to help manage business processes that may be comprised of many individual tasks. On the other hand, OIS focused on the automation of specific office tasks. The latter may be applied selectively to some tasks, but such task automation is not a prerequisite for using and benefiting from workflow. Secondly, workflow is tightly controlled and decisions are based on event results while OIS is

loosely controlled encouraging free wheeling. Thirdly, the target processes to be automated of the two systems are different. OIS usually aimed to automate business processes that are executed in office environment. WFMS consider both of business processes in office environment and manufacturing processes.

2.3 Workflow Modeling

Workflow modeling is a process in which major elements of target workflow are captured and represented using modeling constructs. Many workflow models in the literature are based on the office models of OISs in eighties (Mahling et al, 1995). For instance, X-Workflow (Olivetti Inc.) and FlowPath (Bull Inc.) are based on DOMINO and OFFIETALK, respectively.

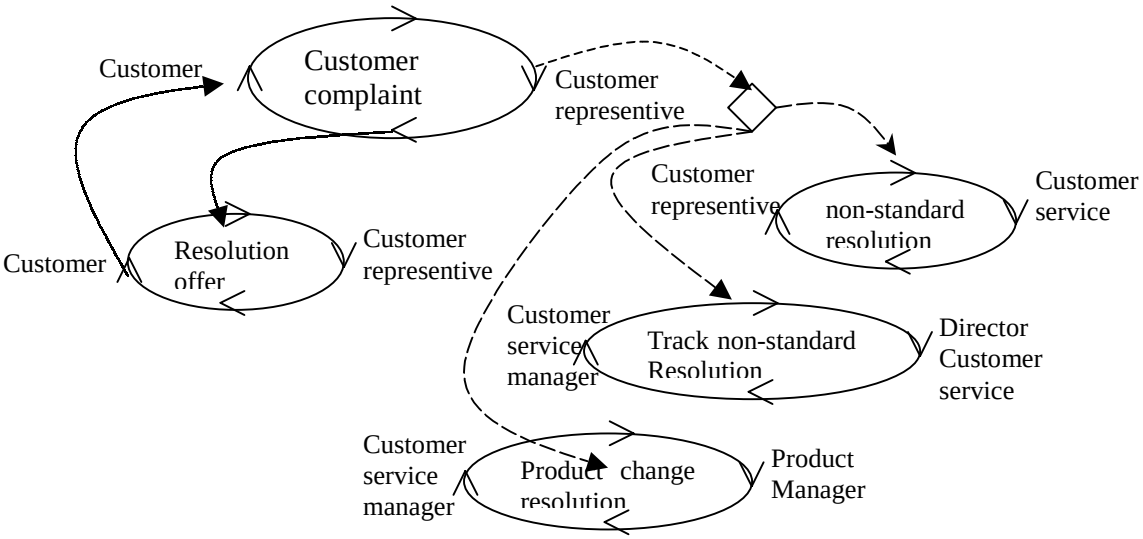
Workflow models in literature can be classified as three categories according to their basic approaches. At first, many works for workflow modeling are based on the input-process-output (IPO) approach (Gruhn, 1995; Ellis and Nutt, 1993; Wolf and Reimer, 1996; van der Aalst et al., 1994; van der Aalst, 1998). It provides task-oriented view on workflows, that is, a workflow is considered as a set of interrelated tasks which process inputs and produce outputs. This approach uses Petri net (Peterson, 1981) or IDEF (Bravoco and Yadav, 1985) as a basic model, and is good to model structured workflows such as business trip approval process and purchasing process. Figure 2.1 illustrates a workflow model based on extended Petri net (ICN). Secondly, language / action approach is also used for workflow modeling (Winograd, 1987; Flores et al., 1988; Michelis and Grasso, 1994; Kaplan et al., 1992). It is based on the conversations between workflow participants, and has merits for modeling unstructured workflow



[Figure 2.1] A workflow model (Vendor selection) based on ICN

such as project planning and customer complaint process. Based on the speech / act theory (Searl, 1969), it manages and automates conversation procedures in workflows. In Figure 2.2, each loop represents conversation process between two actors. A loop (i.e., a conversation) consists of four stages (represented as arrows): preparation, negotiation, performance, and acceptance. A customer name resides on the left-hand

side of the loop,



[Figure 2.2] A workflow model (customer complaint) based on Action Workflow Loop

and the right-hand side of the loop is performer. Thirdly, some research employ object-oriented approach for workflow modeling and enactment (Bose, 1996; Chang and Scott, 1996; Jennings et al., 1996). Bose (1996) presented five classes of objects as a key construct: roles, organization structures, procedures, transitions, and documents. In his model, workflows are executed through message passing between participating objects of the workflows. Both of Chang and Scott (1996) and Jennings et al. (1996) suggested agent based approach for workflow management. In their architecture, autonomous and problem solving agents interact via their own protocol to achieve workflow management goals.

2.4 Workflow Verification

Researches on the verification of workflow model are mainly performed using Petri-net theory (Hofstede et al., 1998; Adam et al., 1998; and Van der Aalst, 1998). Hofstede et al. (1998) some typical verification problems in workflow specifications are identified and their complexity is addressed. With their definition on the workflow structure, they proved that the initiation problem of tasks is NP-complete, and the termination problem and determining safeness of workflow structure are DSPACE(exp)-hard problems. Furthermore, they proved that the safeness of restricted workflow structure that a workflow structure without synchronisers and without cyclic decomposition structures can be verified in polynomial time.

Adam et al. (1998) and Van der Aalst (1997) use Petri nets as a tool for the representation, validation and verification of workflow procedures. Van der Aalst (1997) proposed WF-nets a workflow model that can be verified in polynomial time. Adam et al (1998) proposed Temporal Constraint Petri Net (TCPN) an extended Petri nets as a workflow model. Based on TCPN, they provided algorithms to verify followings;

- Identify inconsistent dependency specifications among tasks
- Test for workflow safety, i.e. test whether the workflow terminates in an acceptable state
- For a given starting time, test whether a workflow is schedulable with the specified temporal constraints.

It is possible to check termination of workflow and occurrence of dangling tasks using Petri-net, but it is difficult to check the correctness of routing condition or mapping rule between role and actor. The rule-based approach for workflow modeling given in this paper enables checking the correctness of the specification of routing conditions, the redundancy of rules, as well as the termination of workflow.

2.5 Change Management of Workflow Model

The issue of flexible workflow management has been addressed in Casati et al. (1998), Reichert and Dadam (1998), Dellen et al. (1997), and Bogia and Kaplan (1995). Casati et al. (1998) suggested a set of primitives that allow modifications of workflow schema, and introduced a taxonomy of policies to manage the evolution of running instances when the corresponding workflow schema is modified. Reichert and Dadam (1998) defined a complete and minimal set of change operations ($ADEPT_{flex}$) that support users in modifying the structure of a running workflow while maintaining its structural correctness and consistency. Dellen et al. (1997) suggested CoMo-Kit system which defines and implements an ontology for project planning. In the CoMo-Kit, it is possible to refine and extend the software process model during process execution using dependency management and change notification mechanism. In these researches, managing the changes such as adding or deleting tasks and changing predefined task sequences are the main concern without considering mechanisms to handle changes on organizational structure and business rules.

2.6 Business Rule Modeling

The importance of business rule modeling is addressed in the database system and information system development fields.

Business rule related research originally started from DBMS field. Researchers on database management system (DBMS) noticed that to develop intelligent business

databases and to manage evolution of these databases, it is necessary to analyze and manage the meaning of data as well as data itself. They stressed the importance of business rule base (Appletone, 1984), classified business rules to entity rules, attribute rules, and event rules (Moriarty, 1993), and suggested business rule analysis steps (Halle 1993). With the emphasis on business rules in database system, a system analysis methodology with rules as central constructs has been suggested (Herbst, 1996). In the research, business rules are defined and structured as a main component of system analysis and presented a meta-model for business rules.

On the other hand, business rules are considered as major means for requirement analysis in information system development. Some researches have studied business rules to automate business rule enforcement to deal with unanticipated situations in doing business activities (Sibley et al., 1992a, 1992b; Michael et al. 1992]. In these studies, logic or theorem was used to represent and model business policies, which is synonymous terms with business rules. An experimental policy workbench, which is a set of tools to model and analyze policies, to maintain a policy database, and to assist policy enforcement, was developed in these studies. Loucopoulos and Layzell (1989) stressed that the information system representation formalism should provide with semantic account, and suggested a conceptual modeling language based on object-oriented and logic paradigm. In their studies, business rules are one of three kinds of knowledge to be represented to build office model. Moriarty (1993b) claimed a new IS analysis paradigm, business rule analysis paradigm, to emphasize the importance of business rule modeling in IS development. Martin (1993) and Odell (1993a, 1993b) suggested a new object-oriented system analysis and design methodology, in which business rule model is an important constituent.

Chapter 3. KWM: Knowledge-based Workflow Model

3.1 Introduction

The basic principles of designing Knowledge-based Workflow Model (KWM) are the flexibility of the model, the expressiveness for complex business rules, and the formality for enabling the analysis of workflow. In KWM, a workflow is defined as a set of business rules for scheduling of tasks, mapping role and actors and routing work items. Business rules restrict and guide a workflow execution according to the state of an organization. The state of organization is represented as a set of attribute values of the organizational objects. For effective modeling of business rules in workflow, two heterogeneous knowledge, i.e., declarative knowledge representing state of an organization and procedural knowledge representing state-based behavior, are represented using frames.

The organization of this chapter is as follows. In section 3.2, business rules for supporting organizational activities are categorized, and business rules for managing workflows are extracted. The basic modeling constructs of KWM are explained in section 3.3, section 3.4, and 3.5 in detail. In section 3.6, the basic routing constructs of workflow are represented using KWM. The usefulness of KWM is shown by applying to an illustrative example in section 3.7. Lastly, the formal definition of KWM is explained in section 3.8.

3.2 Classification of Business Rules

In this dissertation, business rules for supporting organizational activities are classified as three categories, i.e., object integrity rule, task rule, and workflow rule. Object integrity rules deal with integrity constraints to accomplish the accuracy of objects in organizational database systems. These rules can take following forms and are referred for developing organizational database system [Codd, 1990].

- Domain integrity constraint : constraints about domains of attributes
- Column integrity constraint : extension of domain constraints. Additional constraints for a specific column.
- Entity integrity constraint : implies that no components of a primary key is allowed to have missing value
- Referential integrity constraint : for every value of a foreign key in a relation, there must exist a matching value of a primary key
- User-defined integrity constraint : organization practices, policy, or governmental registration to be reflected in the database

Task rules guide how tasks in an organization should be performed and are referred for developing application systems for supporting the tasks. For example, following business rules guide executing the task “Calculate trip cost” in a business trip approval workflow.

- “Traveling by airplane can be allowed in case of urgent duty”
- “Trip cost should be calculated based on the standard unit cost”

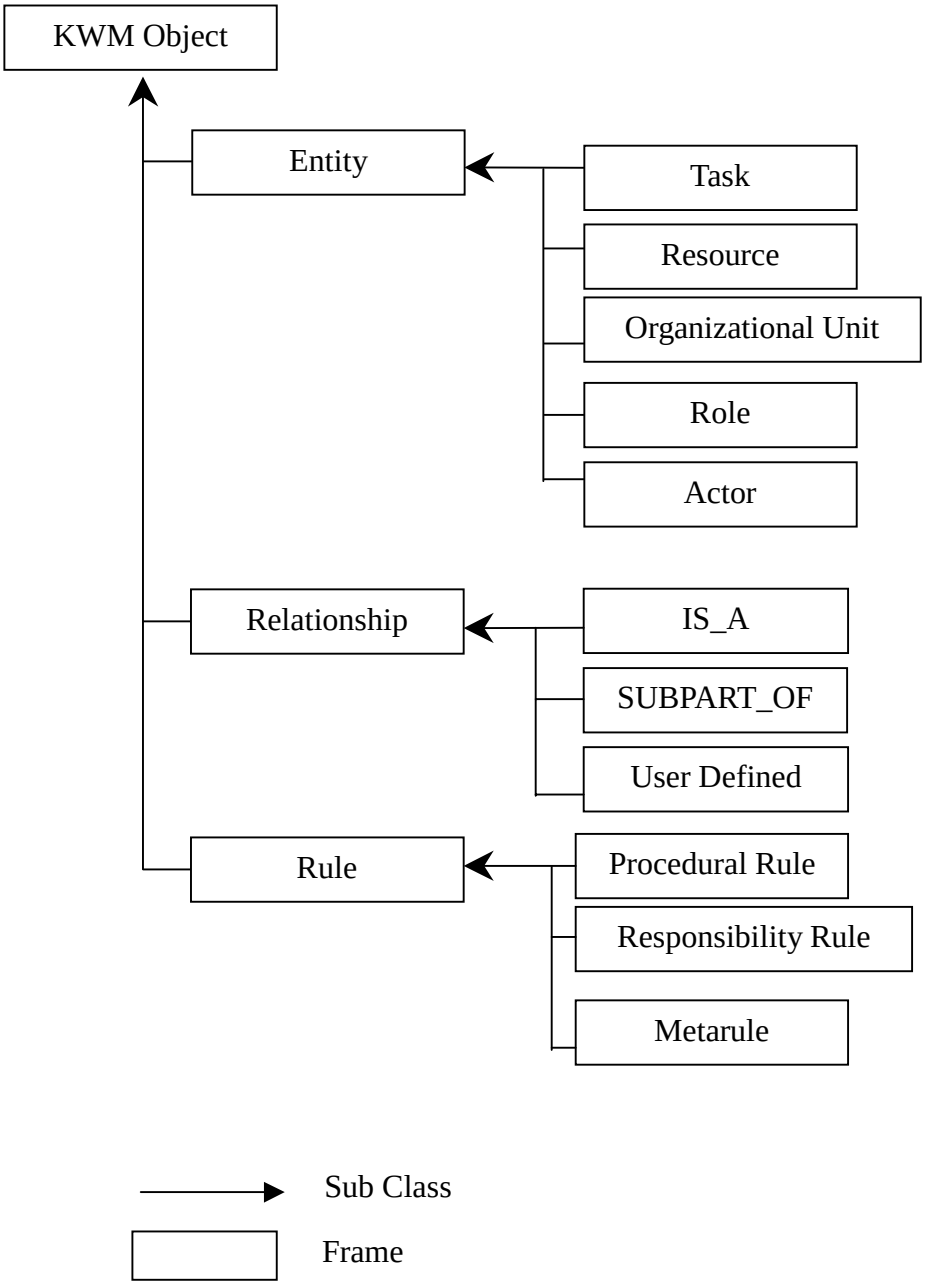
Lastly, workflow rules drive workflows according to organizational policy. Marshark (1994) suggested three main constructs for representing workflows, i.e., route, role, and rule. Route is the execution order among tasks, role is the representative name of agents who are in charge of a task, and rule is used to represent conditional routing or conditional assignments of tasks to actors. Accordingly, business rules that drive workflows are classified as three categories, i.e. procedural rule, responsibility rule, and exceptional rule. The procedural rule guides execution order among tasks in a workflow. The responsibility rule prescribes who are responsible for a task. And the exceptional rule is applied to special workflow instances.

3.3 Modeling Constructs of KWM

Workflow rules prescribe which tasks should be executed after completion of a task and who are responsible for the execution of the tasks using organizational context information. The organizational context information includes organizational structure, organizational resources, and state of the tasks composing a workflow. Accordingly, two heterogeneous knowledge, i.e., declarative knowledge representing state of an organization and procedural knowledge representing state-based behavior, should be modeled for representing workflow rules.

KWM represent workflows as a set of workflow rules and organizational objects. To represent the two heterogeneous knowledge, frame is used as a basic modeling construct. As shown in Figure 3.1, KWM provides three basic types of frame, i.e. entity frame, relationship frame, and rule frame to represent workflow rules. The entity and

relationship frames contain information which is necessary to control workflow, that is,



[Figure 3.1] Hierarchy of frames in KWM.

```
<frame> ::= (<frame-identifier>, {<slot>})
<slot> ::= (<slot-spec>, <slot-value>)
<slot-spec> ::= ( <attribute-name, <domain-type>) | 'CONDITION'
<slot-value> ::= <attribute-value> | {<condition-predicate>}
<condition-predicate> ::=  <variable-declaration> | <existential-element> |
                           <comparative-element> | <functional-element>
<variable-declaration> ::= '(' <frame-name> {'(' <slot-name> <variable> ')'} ')'
<variable> ::= '?'<variable-name>
<existential-element> ::=  '(' <frame-name> {'(' <slot-name> <value> ')'} ')'
<comparative-element> ::= '(test (' <operator> <argument> <value> '))'
<operator> ::= [ > | >= | < | <= | = | != ]
<argument> ::= <variable> | <constant>
<functional-element> ::= '(' <function-name> {<argument>} ')'
```

[Figure 3.2] Frame Specification Syntax of KWM.

they are used to represent organizational model and resources. The rule frames contain rules that control execution of workflow based on the states of entity and relationship frames. The workflow execution rules are classified as procedural rule, responsibility rule, and exceptional rule. Procedural rule schedules workflow tasks, responsibility rule can be considered as a function which relate a task to an actor or a group of actors according to role, and exceptional rule processes exceptional situations during task scheduling and mapping tasks to actors. The basic specification syntax of a KWM frame is shown in Figure 3.2.

3.4 Entity and Relationship Frames

The Entity frame is an abstraction of all entities in an organization. As shown in

Figure 3.1, entity frame has five sub-frames (i.e., task, resource, organizational unit, role, and actor frame). The frame “Task” represents structural and behavioral properties of all the tasks that are performed in a workflow. Every workflow has two artificial tasks called “*Initiate*” denoting the start of a workflow and “*Terminate*” denoting the end of a workflow. The frame “Resource” abstracts all the resources that are used to execute tasks or to control workflow instances. Forms, files, desks, skills, and application software that support executing tasks in a workflow are examples of the resource frame. The frame “Organizational unit” abstracts all the organizational units that compose an organizational structure. Departments, research centers, and special units etc. are example of organizational units. The frame “Actor” abstracts organizational members who perform workflow tasks. Organizational members can be classified according to their skills, titles, and shapes etc. Students, professors, and employees can be sub frames of “Actor” frame.

The frame “Role” abstracts organizational roles that execute tasks with organizational responsibilities. In general, role is defined as “a set of activities that are generally carried out by an individual or group with some organizationally relevant responsibility” (Huckvale 1995). For example, an important role may be that of project managing. This role would be acted by one person at a time, and within the role there are many activities that that person would undertake: planning, reporting, monitoring, managing staff, liaising with suppliers, working with the client, and so on. The role of managing project *X* could be acted by me today, and by another person tomorrow. The role is separate from the people who act it.

Organizational roles can exist with different forms in an organization. The name of a department can be used as a role like “Accounts”. A functional position (like “Financial

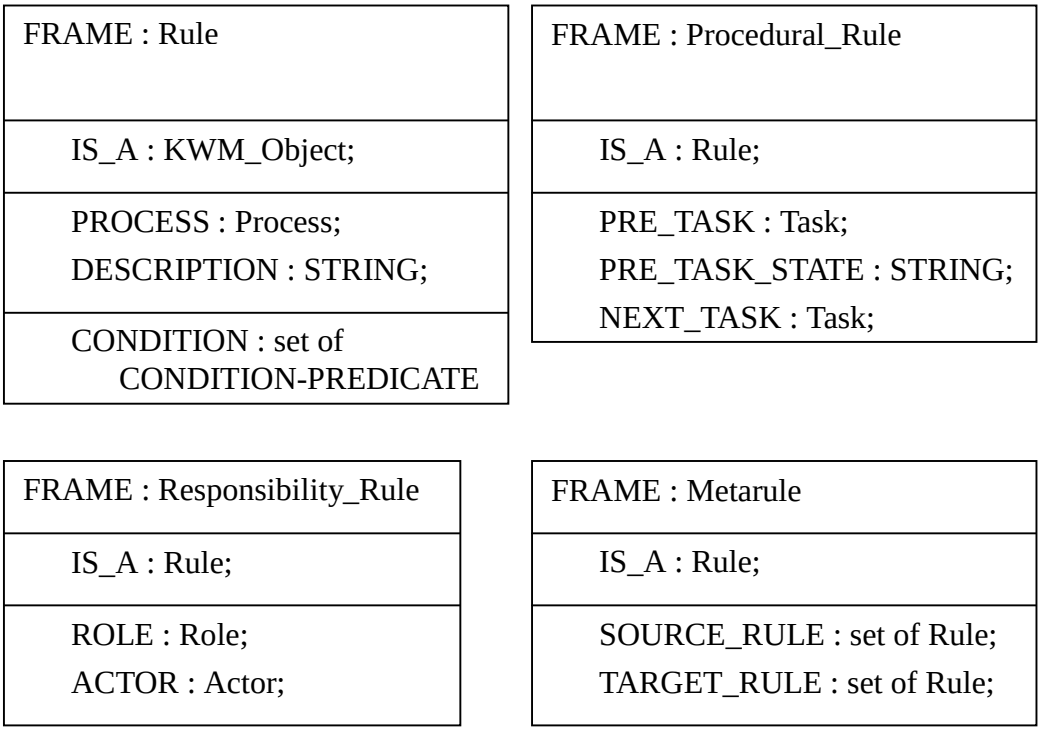
Director”), a class of person (like “Customer”), a job title (like “Project Manager”) also can be used as a role. In KWM, organizational roles are classified as deterministic role and non-deterministic role according to the number of role instances. The non-deterministic role is that the number of role instance is multiple and the role instance that is responsible for a task instance can be changed according to the state of workflow instances. The examples of the non-deterministic role are project manager, supervisor, etc. On the other hand, a deterministic role instance is the same for all workflow instances. Manager of finance department and President are the examples of deterministic role.

In KWM, relationship frame is an abstraction of the structural and behavioral relatedness between two entity frames. The relationships without own properties are represented using slot values of entity frames. On the other hand, the relationships with own properties are represented as frames. Using the entity frames and relationship frames, organizational model including organizational chart and resources are represented.

3.5 Rule Frames

Rule frames represent knowledge on the dynamic behavior of workflow engine such as task scheduling, assignment of tasks to actors, and exception handling. Each rule frame contains multiple slots to represent attribute values for rule management purpose as well as condition and action parts of a rule.

Figure 3.3 shows the specification structure of rule frames. Every rule frame is a sub class of the Rule frame with three slots; PROCESS, DESCRIPTION, and CONDITION.

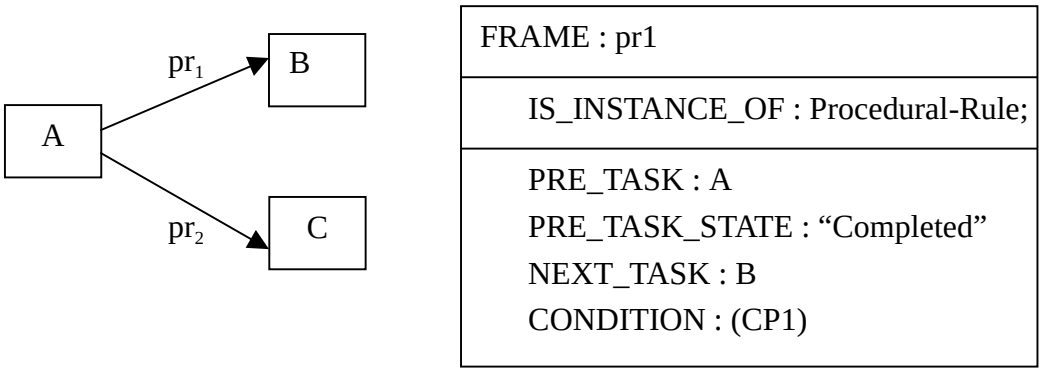


[Figure 3.3] The specification structure of rule frames

The PROCESS slot represents the process to which the rule is applied, and the DESCRIPTION slot represents verbal meaning of the rule. In the CONDITION slot, one or more condition predicates can be specified, and multiple condition predicates are connected with conjunctive relationship, that is, all the conditions should be satisfied to fire a rule.

3.5.1 Procedural-Rule frames.

The Procedural-Rule frames represent procedural view of a workflow. They define conditional sequences between tasks and also establish a communication network among actors in charge of tasks. In Figure 2, a Procedural-Rule frame illustrates that if the state of the task in the PRE_TASK slot is the value specified in the PRE_TASK_STATE slot, and all the conditions in the CONDITION slot are satisfied,



[Figure 3.4] An example of procedural_rule frame.

then the task that is specified in the NEXT_TASK slot is followed.

3.5.2 Responsibility-Rule frames.

In WFMS, the organizational view of a workflow is represented in role concept. Tasks are specified to be performed by roles to increase the flexibility of WFMS. The effective role modeling protects workflow model from the frequent organizational changes including the changes on the department hierarchy, employment or retirement of employees, and changes on the job position in an organization.

The role concept is implemented with the Responsibility-Rule frames in KWM. The Responsibility-Rule frame guides workflow engine to find actors who are in charge of a role. In a Responsibility-Rule frame, the ACTOR slot contains a frame and a slot from which the actor’s identifiers can be extracted. The CONDITION slot contains constraints that instances of the frame specified in the ACTOR slot should satisfy.

The Responsibility-Rule frame for a deterministic role is defined uniquely for all

workflows. The responsibility rule for finding an actor who is in charge of manager of finance department can be uniquely defined to find anyone who works for finance department with a managerial position. On the other hand, the Responsibility-Rule frames for a non-deterministic role can be contingently defined for each workflow. In a trip request processing workflow, the actor who is in charge of the role ‘supervisor’ can be determined by who the trip applicant is.

3.5.3 Metarule frames.

Metarule frames are needed to handle exceptional rules. Exceptional rules are defined as the rules that are applied to special workflow instances. It is often prescribed how to handle special workflow instances in an agile organization. Exceptional rules are defined to handle special instances such as a business process for a special task force, temporary appointment to reduce overload of a special position, and emergency measure to process special customer needs, etc.

Conceptually, a special workflow instance can be handled by substituting rules for the workflow instance. The specification structure of a metarule in Figure 2 represents that the set of rules specified in the SOURCE_RULE slot is substituted by the set of rules specified in the TARGET_RULE slot if the conditions specified in the CONDITION slot are satisfied for a workflow instance.

In Figure 3.5, frames mr1, mr2, and mr3 represent metarules that handle exceptions for procedural rules. The metarule frame mr1 handles an exception that skips a task for a special workflow instance. The metarule frame mr2 is defined to change the order between two tasks. Lastly, the metarule frame mr3 is to resolve conflicts. It fires only procedural-rule frame pr1 when conditions of two procedural-rule frames (pr1 and pr2)

FRAME mr1

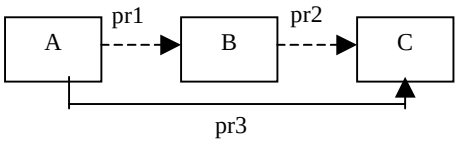
IS_A : Meta_Rule;

SOURCE_RULE: pr1, pr2;

TARGET_RULE : pr3;

CONDITION : ((CP11)…(CP1n));

END_FRAME



a) Skip task B

FRAME mr2

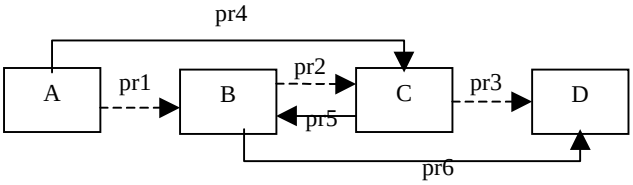
IS_A : Meta_Rule;

SOURCE_RULE: pr1, pr2, pr3;

TARGET_RULE: pr4 pr5 pr6;

CONDITION : ((CP21)…(CP2n));

END_FRAME



b) Change the order between two tasks B and C

FRAME mr3

IS_A : Meta_Rule;

SOURCE_RULE: pr1, pr2;

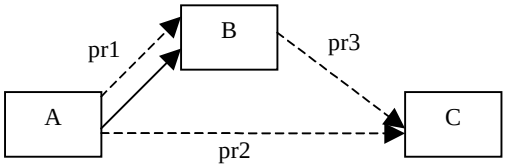
TARGET_RULE : pr1;

CONDITION :

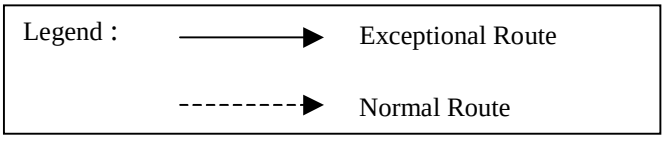
(pr1 (CONDITION TRUE))

(pr2 (CONDITION TRUE));

END_FRAME



c) Resolving conflicting rules



[Figure 3.5] Metarule frames for procedural-rule frames.

are satisfied concurrently.

The metarule frames for responsibility rules are useful for a situation when actor of a task is restricted to someone who has special skills or position for a specific workflow instance. There are two types of metarule frames for responsibility-rule frames. The first type represents the substitution relationship between responsibility-rule frames for non-deterministic role and deterministic role. The situation can arise when the mapping rule for a non-deterministic role (for instance, supervisor) should be ignored and other rule that finds actor of a deterministic role (for instance, the President) should be applied. The second type of metarule frames represents the substitution relationship between responsibility-rule frames for a non-deterministic role.

Logically, the exceptional rules can be handled using procedural-rule or responsibility-rule frames by adding additional conditions that distinguish the special workflow instances. The exceptional rules are, however, apt to change compared with normal rules. Separating exceptional rules from normal rules allows the easy management of rules.

3.5.4 Logic-based representation of procedural-rule frames

The procedural-rule frames are equivalent to well-formed formulas (wffs) of the first order predicate calculus for abstract representation. Every rule in a procedural-rule set *Rp* can be represented as the following wff;

TASK-STATE(x, COMPLETED) $\wedge$ CP1 $\wedge$ … $\wedge$ CPn

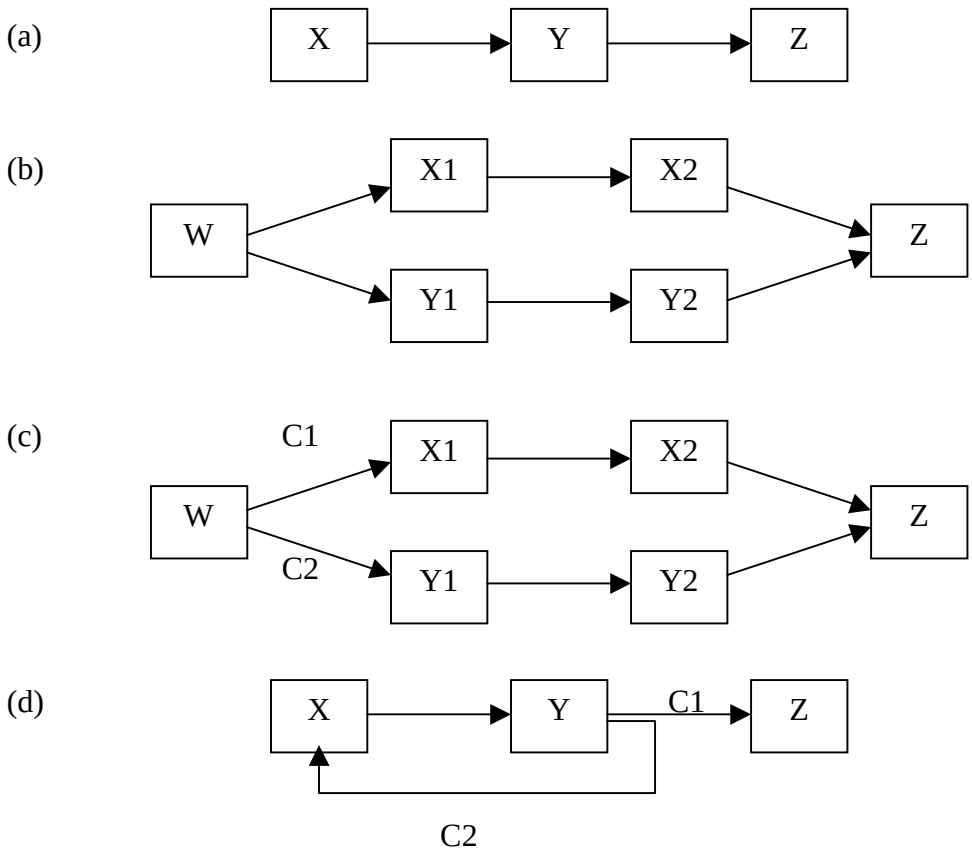
$\Rightarrow$ TASK-STATE(y,INITIATED)

The left-hand side (LHS) of the rule contains two kinds of predicates. The TASK-STATE predicate contains two terms representing a task and a state of the task,

respectively. The two terms are taken from the PRE-TASK slot and the PRE-TASK-STATE slot of a procedural-rule frame. The predicate CPi represents a condition predicate specified in the CONDITION slot. The right-hand side of the rule represents the successor of the task in LHS.

3.6 Routing Constructs

One of the main issues for workflow management is the routing of tasks to be



[Figure 3.6] Four routing constructs: (a) sequential routing; (b) parallel routing; (c) conditional routing; (d) iterative routing.

executed. The workflow management coalition (WfMC) identified four routing constructs (WfMC, 1996). In KWM, the four routing constructs in Figure 3.6 are represented using procedural rules as follows;

- Sequential routing

  Rule 1 : TS(X, C) $\Rightarrow$ TS(Y, I)

- Parallel routing

  Rule 2A : TS(W, C) $\Rightarrow$ TS(X, I) Rule 2B : TS(W, C) $\Rightarrow$ TS(Y, I)

  Rule 3A : TS(X, C) $\Rightarrow$ TS(Z, I) Rule 3B : TS(Y, C) $\Rightarrow$ TS(Z, I)

- Conditional routing

  Rule 4A : TS(W, C) $\wedge$ COND1 $\Rightarrow$ TS(X, I) Rule 4B : TS(W, C) $\wedge$ $\neg$COND1 $\Rightarrow$ TS(Y, I)

  Rule 5A : TS(X, C) $\Rightarrow$ TS(Z, I) Rule 5B : TS(Y, C) $\Rightarrow$ TS(Z, I)

- Iterative routing

  Rule 6A : TS(X, C) $\wedge$ COND2 $\Rightarrow$ TS(Y, I) Rule 6B : TS(X, C) $\wedge$ $\neg$COND2 $\Rightarrow$ TS(X, I)

The predicate “TS” is used for “TASK-STATE” and constant terms “C” and “I” are also used for “COMPLETED” and “INITIATED”, respectively.

Tasks are executed sequentially if the execution of one task is followed by the next task (Rule 1). The parallel routing implies that X and Y can be executed at the same time or in any order if W is completed (Rule 2A and Rule 2B), and Z can be completed when X and Y have been completed (Rule 3A and Rule 3B). On the other hand, conditional routing expresses that X or Y can be executed after W is completed according to the conditions (Rule 4A and Rule 4B). Z is executed after either X or Y is

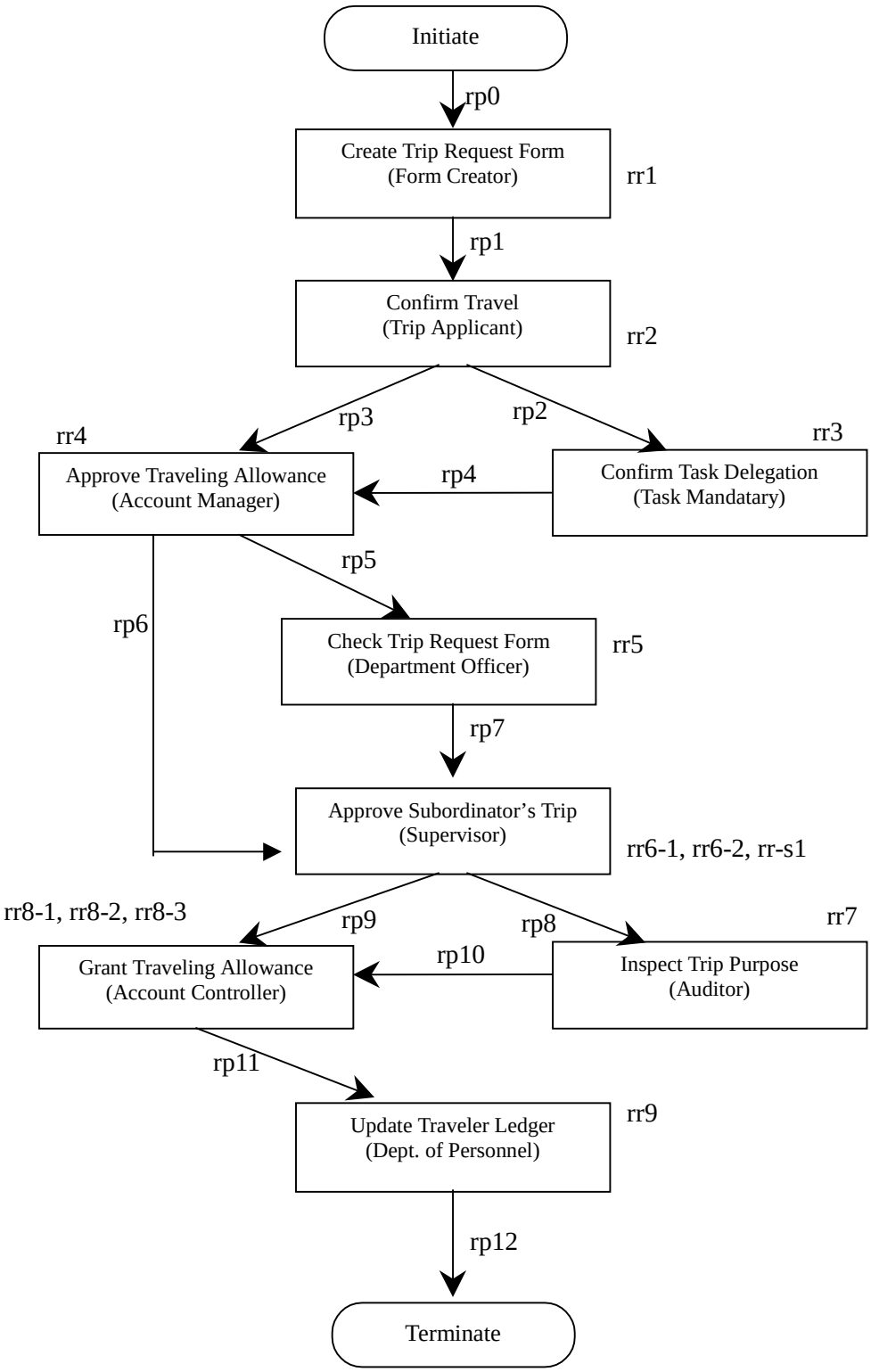
completed (Rule 5A and Rule 5B). Lastly, the iterative routing means that one or more tasks should be repeated until certain condition is satisfied (Rule 6A and Rule 6B).

3.7 An illustrative example

The KWM is applied to the business trip approval process at a university (KAIST: Korea Advanced Institute of Science and Technology) in Korea which has implemented BPR. The overall flow of the “AS_IS” business trip approval process is depicted in Figure 3.7, where each rectangle represents a task and a directed arc represents transition of a workflow instance. In a rectangle, task name and the role that is in charge of the task are specified. Each arc is attached with corresponding procedural rule, and tasks are attached with responsibility rules. The goal of the process is to deal with business trip requests and grants travel allowance according to the organizational rules. The trip applicants can be all members of the university, professors, students, employees, or researchers who work for affiliated research institutes.

3.7.1 Procedural-Rule frames

A workflow instance is initiated when a trip applicant submits electronically a filled-in trip request form for approval. For the case when a secretary fills in the form and submit on behalf of the applicant, all the trip applicants should confirm first (rp1). If the applicant is an employee or a professor assigned to an administrative position, he/she must notify the trip to his/her mandatary to assure the continuity of his/her duties (rp2).


Initiate
rp0
Create Trip Request Form
(Form Creator)
rr1
rp1
Confirm Travel
(Trip Applicant)
rr2
rp3
rp2
rr4
rr3
Approve Traveling Allowance
(Account Manager)
rp4
Confirm Task Delegation
(Task Mandatary)
rp5
rp6
Check Trip Request Form
(Department Officer)
rr5
rp7
Approve Subordinator's Trip
(Supervisor)
rr6-1, rr6-2, rr-s1
rr8-1, rr8-2, rr8-3
rp9
rp8
rr7
Grant Traveling Allowance
(Account Controller)
rp10
Inspect Trip Purpose
(Auditor)
rp11
Update Traveler Ledger
(Dept. of Personnel)
rr9
rp12
Terminate

[Figure 3.7] Business trip approval process at KAIST (AS-IS)

The trip request form is routed to an account manager for approval (rp3). After the approval of fund, the request form is routed to a department officer if the trip applicant is a student (rp5). Otherwise, the form is routed directly to the applicant's supervisor (rp6). After the supervisor's approval, the form can be routed in different route according to the duration of the trip. If the trip duration exceeds 7 days, the auditor inspects the purpose of the trip (rp8) who can approve or reject the trip request. When the request is approved, the request form is routed to account controller who grants the traveling allowance to the trip applicant's account. Then the applicant's trip is notified to the department of personnel, and the workflow is finished.

Figure 3.8 shows the specification of procedural-rule frames for the example. The procedural-rule frame rp10 has null value for the CONDITION slot. If the task "Inspect Trip Purpose" is completed and the state of the task is 'Approved', the workflow instance is directly routed to the task "Grant Traveling Allowance". On the other hand, rp8 conditionally routes workflow instance from the task "Approve Subordinator's Trip" to "Inspect Trip Purpose". If trip duration that is specified in the 'duration' slot of the "Trip Request Form" frame exceeds 6 days, the rule frame is fired.

FRAME : rp8
IS_INSTANCE_OF(procedural_rule)
PROCESS : “Business_Trip_Approval”; DESCRIPTION : “If a supervisor approve subordinator’s trip request and the trip duration exceeds 6 days, the inspector should inspect the trip request”; PRE_TASK : Approve_Subordinator’s_Trip PRE_TASK_STATE : “Approved” NEXT_TASK : Inspect_Trip_Purpose
CONDITION : (Trip_Request_Form (duration ?dur)) (test (>= ?dur 7))

FRAME : rp10
IS_INSTANCE_OF(procedural_rule)
PROCESS : “Business_Trip_Approval”; DESCRIPTION : “If a supervisor approve subordinator’s trip request and the trip duration exceeds 6 days, the inspector should inspect the trip request”; PRE_TASK : Inspect_Trip_Purpose PRE_TASK_STATE : “Approved” NEXT_TASK : Grant_Traveling_Allowance
CONDITION : NULL

[Figure 3.8] Specification examples of procedural-rule frames

3.7.2 Responsibility-rule frames

In general, the supervisor of a trip applicant is the manager of the department where the trip applicant belongs (rr6-1). If a trip applicant is a manager of a department, however, his/her supervisor is the manager of the trip applicant's next super department (rr6-2).

On the other hand, the account controller of a trip account is determined according to the type of trip applicant. If the trip applicant is a student or professor without any administrative position, the account controller is the one who works for the academic & student services department (rr8-1). Otherwise, the account controller is determined according to the type of the account from which the traveling allowance is granted. If the traveling allowance is granted from the research project account, the account controller is the one who works for the research management department (rr8-2). In other cases, the account controller is the one who works for the finance department (rr8-3).

In Figure 3.9 – 3.10, the rule frames are used to determine actors for the non-deterministic role 'Supervisor' and 'Account_Controller'. The predicates in the CONDITION slot of the rule frames represent entity frames, and the occurrence of the same variable (represented by attaching '?') in different frames means that the slot values for the variables should be identical.

FRAME : rr6-1
IS_INSTANCE_OF(responsibility_rule)
PROCESS : “Business_Trip_Approval”; DESCRIPTION : “Traveler’s supervisor is one who works for the Department with manager position which the traveler belongs”; ROLE : Supervisor ACTOR : WorkFor.actor_id
CONDITION : (Traveler (Department ?dept-id)) (Department (dept_id ?dept-id) (mnger_pos ?m-pos)) (WorkFor (dept_id ?dept-id)(actor_id ?supervisor_id)(position ?m-pos))

FRAME : rr6-2
IS_INSTANCE_OF(responsibility_rule)
PROCESS : “Business_Trip_Approval”; DESCRIPTION : “The supervisor of a department manager is the manager Of super-department of the department he belongs.” ROLE : Supervisor ACTOR : WorkFor.actor_id
CONDITION : (Traveler (Department ?dept-id)) (Department (dept_id ?dept-id) (super-dept ?s-dept)) (Department (dept_id ?s-dept) (mnger_pos ?m-pos)) (WorkFor (dept_id ?s-dept)(actor_id ?supervisor_id)(position ?m-pos))

[Figure 3.9] A specification example of responsibility_rule frame for “supervisor”

FRAME : rr8-1
IS_INSTANCE_OF(responsibility_rule)
PROCESS : “Business_Trip_Approval”; DESCRIPTION : “If a traveler is a student or professor without assigned Position, the account controller of Academic & Student Service department grants the traveling allowance” ROLE : Account_Controller ACTOR : Task_Charge.actor
CONDITION : (Traveler (T_Id ?t-id) (T_Department ?dept)) (or (Student (S_Id ?t-id))(Professor (P_Id ?t-id) (Position NIL))) (Task_Charge (Task ‘Grant_Traveling_Allowance’) (Department ‘Academic_&_Student_Service’) (Actor ?a-id))

FRAME : rr8-2
IS_INSTANCE_OF(responsibility_rule)
PROCESS : “Business_Trip_Approval”; DESCRIPTION : “If a traveler is not a student or professor with assigned Position and the account type is ‘project’, the account controller of Research Management department grants the traveling allowance” ROLE : Account_Controller ACTOR : Task_Charge.actor
CONDITION : (Traveler (T_Id ?t-id) (T_Department ?dept)) (Trip_Request_Form (Account ?acc)) (not (Professor (P_Id ?t-id) (Position NIL))) (not (Student (S_Id ?t-id))) (Account (Account_Id ?acc) (Type ‘project’)) (Task_Charge (Task ‘Grant_Traveling_Allowance’) (Department ‘research_management’) (Actor ?a-id))

[Figure 3.10] A specification examples of responsibility_rule frame for “Account Controller” (*Continue*)

FRAME : rr8-3
IS_INSTANCE_OF(responsibility_rule)
PROCESS : “Business_Trip_Approval”; DESCRIPTION : “If a traveler is not a student or professor with assigned Position and the account type is ‘general’, the account controller of Research Management department grants the traveling allowance” ROLE : Account_Controller ACTOR : Task_Charge.actor
CONDITION : (Traveler (T_Id ?t-id) (T_Department ?dept)) (Trip_Request_Form (Account ?acc)) (not (Professor (P_Id ?t-id) (Position NIL))) (not (Student (S_Id ?t-id))) (Account (Account_Id ?acc) (Type ‘general’)) (Task_Charge (Task ‘Grant_Traveling_Allowance’) (Department ‘finance’) (Actor ?a-id))

[Figure 3.10] Specification examples of responsibility-rule frames for “Account Controller”

3.7.3 Metarule frames

Two exceptional rules exist in the example workflow. At first, if a trip applicant is a director of an affiliated research institute, the vice President of KAIST becomes the applicant's supervisor although the formal supervisor of the research institute is the President (rm2). This exceptional rule existed temporarily to reduce the workload of the President. Secondly, if trip applicant is an employee who is delegated to another department, then the sequence between tasks "Approve Traveling Allowance" and "Approve Subordinator's Trip" is reversed (rm3). In Figure 3.11, metarule frame rm1 substitutes the responsibility-rule frame rr6-1 with rr6-2 for the trip of a manager of a department. On the other hand, metarule frame rm2 handles an exceptional situation for the trip of a director of an affiliated research institute. The responsibility-rule frame rr-s1 represents mapping relationship between an actor and the deterministic role 'Vice_President'.

FRAME : rm1
IS_INSTANCE_OF(metarule)
PROCESS : “Business_Trip_Approval”; DESCRIPTION : “If a traveler is manager of a department, the manager of Superdepartment of the department becomes his supervisor” SOURCE_RULE : rr6-1 TARGET_RULE : rr-s1
CONDITION : (Traveler (T_Id ?t-id) (T_Department ?dept)) (Department (dept_id ?dept) (mnger_pos ?m-pos)) (WorkFor (dept_id ?dept)(actor_id ?t-id) (position ?m-pos))

FRAME : rm2
IS_INSTANCE_OF(metarule)
PROCESS : “Business_Trip_Approval”; DESCRIPTION : “If a traveler is a director of an affiliated reseach institute, the supervisor is the vice President although his formal supervisor is the President.” SOURCE_RULE : rr6-1 TARGET_RULE : rr6-2
CONDITION : (Traveler (T_Id ?t-id)) (WorkFor (actor_id ?t-id) (position ‘director-of-affiliated-research-institute’))

[Figure 3.11] Specification examples of metarule frames.

3.8 Formal Definition of KWM

In this section, a set-theoretic formalism of KWM is presented. Formal definition is required for several reasons; self-complacency, communication support, extension and modification of the framework, and formal comparisons with other models. It can be also used as formal foundations to develop the computerized modeling environments based on KWM, because different development tools and underlying platforms can be applied toward more integrated and intelligent modeling environment.

Set theory provides the means to construct formalisms which specify objects [Ziegler 84]. Since this approach is mathematically sound and provides primitive constructs, it has been frequently adopted, e.g. [Kim 94], and [Kang 95].

Each class of frames is represented by a formalism which prescribes its parameters and any governing constraints. Thus, formal model definitions prescribe a list of parameters which are set-theoretic constructs, and list of constraints.

A KWM *w* is defined as follows;

DEFINITION 3.1 (WORKLFOW MODEL)

A KWM defines a workflow with a 3-tuple, $w = (\boldsymbol{E}, \boldsymbol{Rl}, \boldsymbol{Ru})$, where $w \in \boldsymbol{W}$, ***E*** is a set of entity frames and ***Rl*** is a set of relationship frames and ***Ru*** is a set of rule frames. A frame *f* in ***E***, ***Rl,*** or ***Ru*** is defined as a product of slot and value pairs, that is,

$f = (s_1, v_1) \times (s_2, v_2) \times \ldots\ldots \times (s_{n-1}, v_{n-1}) \times (s_n, v_n)$.

If two frames are the instances of the same class, these two frames have the same slots. Also, a slot of a frame can be a relationship of the frame, and the value of the slot

can be another frame with which the frame has a relationship.

DEFINITION 3.2 (ENTITY FRAME)

An entity frame in *E* belongs to one category among five kinds of objects; tasks, resources, organizational units, roles, and actors. That is, $\boldsymbol{E} = \boldsymbol{T} \cup \boldsymbol{Re} \cup \boldsymbol{U} \cup \boldsymbol{Ro} \cup \boldsymbol{A}$ where ***T*** is a set of tasks, ***Re*** is a set of resources, ***U*** is a set of organizational units, ***Ro*** is a set of roles, and ***A*** is a set of actors.

DEFINITION 3.3 (RELATIONSHIP FRAME)

A relationship frame is 3-tuple, $rl = (so, si, \boldsymbol{P})$, where $rl \in \boldsymbol{Rl}$, *so* is a source slot that contains source entity for the relationship, *si* is a sink slot that contains sink entity for the relationship, and ***P*** is a set of property slots of the relationship.

The definitions 3.4 and 3.5 are the formal definitions related with system-defined relationships, IS_A and Is_Part_Of.

DEFINITION 3.4 (IS_A RELATIONSHIP)

A relation IS_A is a binary relation on **E** and **R**, that is $IS\_A \subseteq E \times E \cup R \times R$, that defines a patial order on the sets, E and R. For a pair $(x,y) \in IS\_A$, denoted by IS_A(x, y), it is said that x is a subobject of y or y is a superobject of x. Sup is a function from $E \cup R$ to $\wp(E) \cup \wp(R)$ such that $Sup(x) = \{y \mid (x,y) \in IS\_A\}$ where $\wp(E)$ and $\wp(R)$ are the power sets of E and R, respectively. Sup(x) is called the set of parent objects of x. Sub is a function from $E \cup R$ to $\wp(E) \cup \wp(R)$ such that $Sub(x) = \{y \mid (y, x) \in IS\_A\}$. Sub(x) is called the set of child objects of x.

DEFINITION 3.5 (IS_PART_OF RELATIONSHIP)

An IS_PART_OF relation is a binary relation on **E**. For a pair (x, y) ∈ IS_PART_OF, denoted by IS_PART_OF(x, y), it is said that x is an aggregation object of y. Agg is a function from O to $\wp$(O) such that Agg(y) = {x | (x, y) ∈ IS_PART_OF}. Agg(y) is called the set of aggregated object of y.

The definitions from 3.6 to 3.9 are formal definitions on the rule frames of KWM.

DEFINITION 3.6 (RULE FRAME)

A rule frame in *Ru* belongs to one category among three kinds of rules; procedural rules, responsibility rules, and metarules. That is, ***Ru*** = ***Rp*** ∪ ***Rr*** ∪ ***Rm*** where ***Rp*** is a set of procedural rule, ***Rr*** is a set of responsibility rule, and ***Rm*** is a set of metarule. The set ***Rp*** is a set of rules that conditionally connect tasks with the tasks followed. The set ***Rr*** is a set of rules that conditionally relate roles with actors. The set ***Rm*** is a set of rules that conditionally relate two or more procedural rules or responsibility rules.

DEFINITION 3.7 (PROCEDURAL_RULE FRAME)

A procedural_rule frame is 6-tuple, *pr = (p, d, pT, pTS, nT, c)* where *pr* ∈ ***Rp***, *p* is a source slot that contains target process, *d* is a description slot on the rule frame, *pT* is a source task slot, *pTS* is a state slot of the source task, *nT* is a sink task slot, and *c* is a condition slot.

DEFINITION 3.8 (RESPONSIBILITY_RULE FRAME)

A responsibility_rule frame is 5-tuple, *rr = (p, d, ro, a, c)* where *rr* ∈ ***Rr***, *p* is a source slot that contains target process, *d* is a description of the rule frame, *ro* is a slot that contains a role frame, *a* is a slot that contains an actor frame, and *c* is a condition slot.

DEFINITION 3.9 (METARULE FRAME)

A metarule frame is a 5-tuple, *rm = (p, d,* ***Sr****,* ***Tr****, c)* where *rm* ∈ ***Rm***, *p* is a source slot that contains target process, *d* is a description of the rule frame, ***Sr*** is a set of source rule frames, ***Tr*** is a set of target rule frame, and *c* is a condition slot.

The definition from 3.10 to 3.16 are concerned with condition slot of rule frames.

DEFINITION 3.10 (VALUE OF A CONDITION SLOT)

The value of a condition slot is a *formula* which returns TRUE or FALSE, and consists of series of *conditional elements* that have conjunctive relationship each other, i.e.,

$cv = ce_1 \wedge ce_2 \wedge \ldots \wedge ce_n$ where ce_i is i^{th} conditional element.

In the definition 3.10, the CONDITION slot of a rule frame takes a formula that consists of conditional elements that are connected conjunctively. If the formula returns TRUE, the rule frame can be fired (i.e., a work item can be routed to followed tasks, a task can be assigned to actors, and exceptional rules are applied to the workflow instance).

DEFINITION 3.11 (CONDITIONAL PREDICATE)

A conditional predicate is expressed as one or two conjunctive predicate elements. Each

predicate element can be one of the variable declaration, existential element, comparative element, functional element, i.e.,

$cp = pe_1 \wedge pe_2 \wedge \ldots \wedge pe_m$ where pe_j is j^{th} predicate element and

$pe_j = pce \vee test\text{-}ce \vee not\text{-}ce \vee or\text{-}ce \vee and\text{-}ce$

where *vd* is variable declaration,

ee is existential element,

ce is comparative element, and

fe is functional element.

The conditional predicate is again composed of one or two conditional element. The conditional element can be classified as four types. Variable binding binds a variable with an attribute value of an object. The following conditional element declares a variable (?sno) which takes a value of the student_no attribute of an entity frame "Student".

CP1: (Student (student_no ?sno))

Existential elements check whether there exists any frame instance that has certain values. The values can be a constant or a variable a value is bound to at the predefined conditional element. The following conditional predicates are legal expression.

CP2: (Book (title "Excellence in Practice") (publisher "Future Strategies Inc."))

CP3: (Book (title ?title) (publisher ?pub))

(CSBook (title ?title) (publisher ?pub))

Comparative elements compare two or more values according to the operator of the element. The reserved words "test" indicates that the conditional element is a comparative element.

CP4: (Student (student_no ?sno))

(test (>= ?sno 950000))

CP5: (Student (student_no ?sno))

(test (= ?sno ?pno))

A functional element is an element that contains a function call. A function returns a value or a list of values after processing on the input arguments. The number of input arguments can be multiple.

CP6: (test (= (sum 5 6) 11))

DEFINITION 3.12 (EQUALITY OF EXISTENTIAL ELEMENT)

Two existential elements ee_1 and ee_2 are equal if and only if the followings are satisfied;

i) ee_1.frame = ee_2.frame,

ii) ee_1.slot = ee_2.slot, and

iii) ee_1.value = ee_2.value

DEFINITION 3.13 (EQUALITY OF COMPARATIVE ELEMENT)

Two comparative elements ce_1 and ce_2 are equal if and only if the followings are satisfied;

i) ce_1.operator = ce_2.operator,

ii) if (ce_1.argument = variable) $\wedge$ (ce_2.argument = variable)

get_frame_slot(ce_1.argument) = *get_frame_slot*(ce_2.argument)

where get_frame_slot( ) is a function that returns frame and slot names a variable is bound to.

else ce_1.argument = ce_2.argument

iii) ce_1.value = ce_2.value

DEFINITION 3.14 (EQUALITY OF FUNCTIONAL ELEMENT)

Two functional elements fe_1 and fe_2 are equal if and only if the followings are satisfied;

i) fe_1.function = fe_2.function,

ii) for each argument arg of fe_1

if ($fe_1.arg_i$ = variable) $\wedge$ ($fe_2.arg_i$ = variable)

get_frame_slot($fe_1.arg_i$) = *get_frame_slot*($fe_2.arg_i$) where arg_i is i^{th} argument of the function

else $fe_1.arg_i = ce_2$.argument

DEFINITION 3.15 (EQUALITY OF CONDITIONAL PREDICATE)

Two conditional predicates cp_1 and cp_2 are equal if and only if their predicate elements are the same.

DEFINITION 3.16 (EQUALITY OF CONDITION SLOT)

Two condition slots c_1 and c_2 are equal if and only if their conditional predicates are the same.

DEFINITION 17 (DIFFERENCE BETWEEN CONDITION SLOTS)

A function *DiffCond*(c_1, c_2) returns conditional predicates in c_1 excepting that are contained also in c_2

DEFINITION 18 (INTERSECTION BETWEEN CONDITION SLOTS)

A function *IntersCond*(c_1, c_2) returns the conditional predicates in c_1 that are contained also in c_2

Chapter 4. Verification of KWM

4.1 Properties for Sound KWM

The purpose of workflow model verification is to determine whether the model represents target workflow correctly. There exist some verification techniques for workflow based on Petri-net (Hofstede et al., 1998; Adam et al.,1998; Van der Aalst, 1998). The techniques are limited to the verification of routes such as checking termination of workflow or occurrence of dangling tasks. The rule-based approach of KWM allows verification of correct specifications of rules as well as routes of workflow model. To ensure a KWM represents a workflow correctly, it should satisfy a specific property, soundness. To define soundness of KWM, some properties are defined.

DEFINITION 4.1 (CIRCULARITY) A KWM *w* has a circularity iff any of the following conditions is satisfied;

for each pr $\in$ **Rp**,

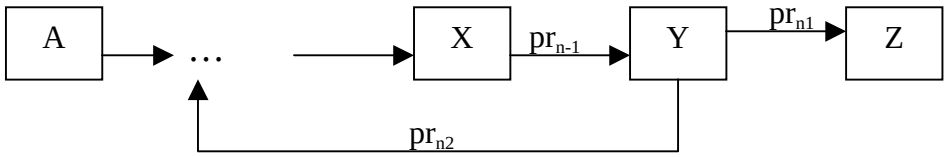
i) $\exists$pr such that (pr.NEXT_TASK = t) $\wedge$ (t $\in$ $\mathbf{T_p}$) $\wedge$ (pr.CONDITION = Ø)

where $\mathbf{T_p}$ is a set of predecessors of t.

ii) $\exists$pr such that (pr.NEXT_TASK = t) $\wedge$ (t $\in$ $\mathbf{T_p}$) $\wedge$(pr.CONDITION $\neq$ Ø) $\wedge$

(dom(pr.CONDITION) $\wedge$ dom(pr.CONDITION) $\neq$ Ø)

Occurrence of circularity in a KWM means that workflow instance enters into a loop which prevents termination of the workflow instance. The situation can be happened under two situations. In Figure 4.1, procedural_rule frame pr_{n2} routes work



$(pr_{n2}.CONDITION = \varnothing) \vee (dom(pr_{n1}.CONDITION) \wedge dom(pr_{n2}.CONDITION) \neq \varnothing)$

[Figure 4.1] Occurrence of circularity

items to a predecessor of the task Y. If the CONDITION slot of the pr_{n2} has null value, work items are always returned to the predecessor of Y which results in a loop. In the case that the CONDITION slot of the pr_{n2} has non-null value, the domain of the CONDITION slot of the pr_{n2} should not be duplicated with that of pr_{n1}. If it is duplicated with that of prn1, there happens a loop for the workflow instance which has the duplicated value.

DEFINITION 4.2 (MISSING RULE) A KWM *w* has a missing rule iff any of the followings is not satisfied;

i) $\exists pr_1$ such that $(pr_1 \in$ **Rp)** $\wedge (pr_1.PRE\_TASK = INITIATE)$

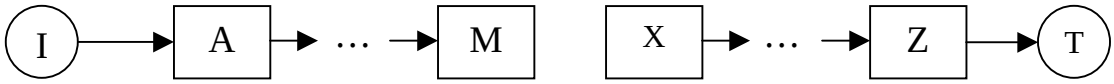
ii) $\exists pr_2$ such that $(pr_2 \in$ **Rp)** $\wedge (pr_2.NEXT\_TASK = TERMINATE)$

iii) for each t $\in$ **IMT** where **IMT** = **T** - {INITIATE, TERMINATE}

$\exists pr_1, pr_2$ such that

$(pr_1, pr_2 \in$ **Rp)** $\wedge (pr_1.PRE\_TASK = t) \wedge (pr_2.NEXT\_TASK = t)$

The occurrence of missing rule in a KWM means that some procedural_rule frames are not defined which results in disconnection of a workflow. In Figure 4.2, a



[Figure 4.2] Occurrence of missing rule

procedural_rule frame that connects task M or its predecessor with task X or its successors should be defined.

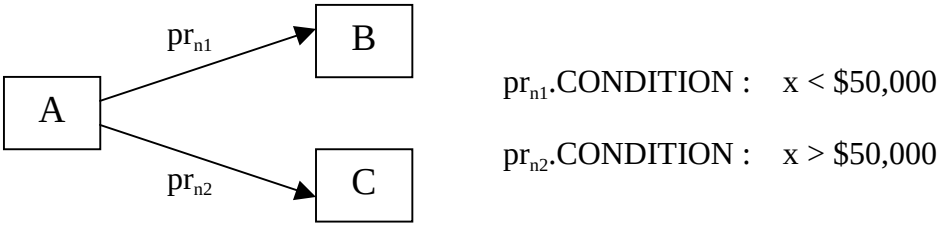
DEFINITION 4.3 (MISSING VALUE) A set of rule frames **Rp(t)** which connect from task *t* to its successors has missing value for constraining o ∈ **O(t)** where **O(t)** is a set of objects that are constrained by **Rp(t)** if and only if,

for each o ∈ **O(t)**,

$$\vee_{\text{for all pre} \in \mathbf{Rp(t)}} \text{pr.CONDITION|o != dom(o)}$$

where pr.CONDITION|o is a projected condition of pr.CONDITION, which is restricted as a condition of object o.

The occurrence of missing value can be happened in the course of defining conditional routing constructs. In Figure 4.3, procedural_rule frame pr_{n1} and pr_{n2} have an exclusive relationship that one of them can be fired after completion of task A. The rule frames route work item according to the value of a variable x which represents state of object attribute. As the two rule frames do not consider the case variable x binds $50,000, the two rule frames have missing value.



[Figure 4.3] Occurrence of missing value

From the above three properties, we can define the first property of soundness of KWM.

DEFINITION 4.4 (TERMINALITY) A KWM *w* can be terminated if and only if circularity, missing-rule, and missing-value are not occurred in the set of rule frames.

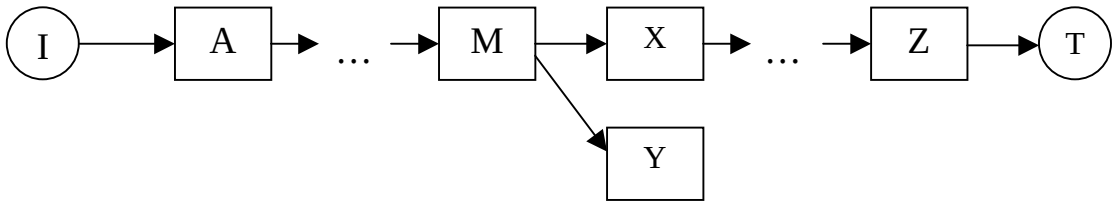
The definition 4.5 defines the second property to be a sound KWM.

DEFINITION 4.5 (COMPLETENESS) A KWM *w* is complete iff termination of a workflow instance means all the task instances that compose the workflow instance are completed. i.e., the following condition should be satisfied

for each t ∈ **MT** where **MT** = **T** - terminate

$\exists$pr such that (pr ∈ **Rp)** ∧ (pr.PRE_TASK = t)

The completeness of KWM can be assured if there is no dangling task which has no predecessor or successor. The occurrence of dangling task can generate the situation that a workflow instance can be terminated even though some tasks are not completed. In



[Figure 4.4] Occurrence of dangling task

Figure 4.4, task Y does not has successors, which could be uncompleted even though the workflow instance has been terminated.

DEFINITION 4.6 (COMPACTNESS) A KWM *w* is compact iff any of the rule frame is not subsumed or duplicated with other rule frame, i.e., the following conditions should be satisfied

i) for each pr_1 such that $pr_1 \in$ **Rp**

$!\exists pr_2$ such that ($pr_2 \in$ **Ru)** $\wedge$ (pr1.NEXT_TASK = pr2.NEXT_TASK) $\wedge$

(pr1.PRE_TASK = pr2.PRE_TASK) $\wedge$

(IntersCond(pr_1.CONDITION, pr_2.CONDITION) = pr_1.CONDITION)

ii) for each rr_1 such that $rr_1 \in$ **Rr**

$!\exists rr_2$ such that ($rr_2 \in$ **Rr)** $\wedge$ (rr1.ROLE = rr2.ROLE) $\wedge$

(IntersCond(rr_1.CONDITION, rr_2.CONDITION) = rr_1.CONDITION)

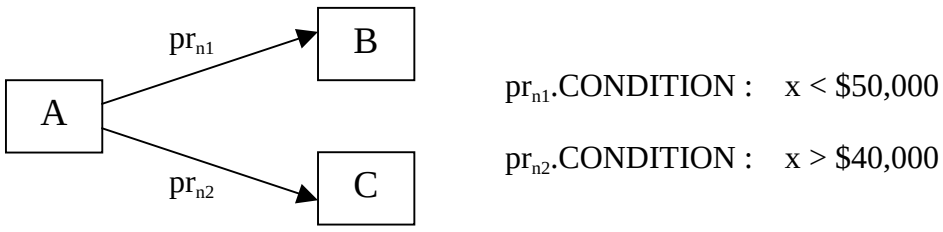
The compactness property can be violated if there are redundant rules. The occurrence of redundancy means that some rules or literals in a rule can be removed without affecting the soundness of a KWM. A rule is redundant if it is subsumed or

duplicated with other rules. A subsumed rule is that the antecedents of the rule consist of a subset of the antecedents of other rule that has the same consequents with the subsumed rule. If two rules have the same antecedents and consequents, the rules are duplicated. The redundancy between rule frames can be happened in both of procedural rule set and responsibility rule set.

DEFINITION 4.7 (CONSISTENCY) A KWM *w* is consistent iff unintended consequents are not produced from **Ru**, i.e.,

for each pr_1 such that $pr_1 \in$ **Rp**

$!\exists pr_2$ such that $pr_2 \in$ **Rp** $\wedge$ IntersCond(pr_1.CONDITION, pr_2.CONDITION) $\neq \varnothing$



[Figure 4.5] Occurrence of conflicting rules

Finally, we define soundness of KWM as follows;

DEFINITION 4.7 (SOUNDNESS OF KWM) A KWM, ***w*={*E, Rl, Ru*}**, is sound if and only if it satisfies the following properties:

(i) *Terminality* : The set ***Ru*** assures termination of all instances of a workflow.

(ii) *Task Completeness* : The set ***Ru*** assures not occurring dangling tasks.

(iii) *Compactness (Rule Minimality)* : The set ***Ru*** assures not occurring redundant rules.

(iv) *Routing Consistency* : There are no conflicting rules in the set ***Ru***.

(v) *Referential Integrity* : There does not exist illegal reference in ***W***.

The first four properties guarantee the soundness apart from certain anomalies in the set of rule frames. The meaning of referential integrity is twofold. First, it restricts the participants to relationships in KWM to be valid entities. That is, if an entity instance that participates in a relationship is removed, the relationship instance should also be removed. Secondly, the referential integrity prevents illegal constraints which constrain the state of non-existent entities or relationships. The rule frames constrain their activation time using the state of entities or relationships in the CONDITION slot. If the condition is defined on the state of non-existing objects, the referential integrity is violated.

4.2 KWM Verification Algorithm

Table 1 summarizes the anomalies that violate the soundness of KWM. The verification of the first four properties can be performed by detecting the anomalies in the set of rules defined in a KWM, which cause violation of the properties. Generally, the development of techniques for detecting anomalies in knowledge-base is difficult as the determination of rules to be checked for an anomaly can not be previously defined. The rule frames in KWM can be categorized according to target frame to which the rules are related. For example, checking conflicting rules for routing consistency can be performed to only a set of rules that connect a task with its successor.

ALGORITHM 4-1 (CHECKING CIRCULARITY)
Given a set of procedural_rule frames Rp,
1. set $T_1 = T_2 = \{initiate\}$, $T_3 = \varnothing$,
2. while (T_1 != T)
    set $T_1 = T_1 \cup T_2$
    for each $t_p \in T_2$,
      set $T_3 = \{ t_S | \exists pr \in Rp$ such that pr.PRE_TASK = $t_p \wedge$ pr.NEXT_TASK = $t_S \}$
       for each $t_S \in T_3$,
          if ($t_S \in T_1$) $\wedge$ (pr.CONDITION = NULL)
              print “There is a cycle which is started from t_s to t_p ”.
          else if $t_S \in T_1 \wedge$ pr.CONDITION $\neq$ NULL
              execute algorithm 4.6
              if (algorithm4.6 returns conflicting rule)
                  print “There is a cycle which is started from t_s to t_p ”.
              else add task t_n to T_1.
          set $T_S = T_S \cup T_3$
       end

Table 4-1. Anomalies that violate soundness of KWM.

Property		Rule Set	Explanation
Termi-nality	Circularity	*Rp1* = {TS(Initiate, C) ⇒ TS(X,I), TS(X, C)⇒TS(Y, I), TS(Y, C)⇒TS(X, I), TS(Y, C) ⇒ TS(Terminate, I), ∀(u) [TS(u, I) ⇒ TS(u, C)]}	The set *Rp1* has circularity so that a workflow instance enters a loop between task X and Y.
	Missing Rule	*Rp2* = { TS(Initiate, C) ⇒ TS(X, I), TS(Y, C) ⇒ TS(Z, I), TS(Z, C) ⇒ TS(Terminate, I), ∀(u) [TS(u, I) ⇒ (TS(u, C)]}	The sub set *Rp2* is missing a rule that connects task X and task Y. In the case, the workflow instances can not progress after task X is completed.
	Missing Values	*Rp3* = { TS(Initiate, C) ⇒ TS(X, I), TS(X,C) ∧ LARGER(v,10)⇒ TS(Y,I), TS(X,C) ∧ SMALLER(v, 5)⇒ TS(Z, I), …, ∀(u) [TS(u, I) ⇒ TS(u, C)]}	If a workflow instance that binds the variable with a value between 5 and 10 is created, the workflow instance becomes dead.
Completeness		*Rp4* = {TS(Initiate, C) ⇒ TS(X, I), TS(X, C)⇒TS(Y, I),TS(X, C) ⇒ TS(Z, I), TS(Y, C) ⇒ TS(Terminate, I), ∀(u) [TS(u, I) ⇒ TS(u, C)]}	The task Z does not affect the route of workflow instances. The task Z should be connected to the task Terminate.
Compactness		*Rp5* = { TS(X,C)∧COND1(x)∧COND2(y) ⇒ TS(Y, I), TS(X, C) ∧ COND1(x) ⇒ TS(Y, I), COND1(x) ∧ TS(X, C) ⇒ TS(Y, I)}	In the set *Rp5*, the first rule is subsumed by the second rule, and the second rule is duplicated with the third rule.
Consistency		*Rp6* = { TS(X, C) ∧ LARGER(x, 5) ⇒ TS(Y, I), TS(X, C) ∧ SMALLER(x, 10) ⇒ TS(Z, I)}	The first two rules in the set *Rp6* may infer conflicting hypotheses when a workflow instance binds the variable x with a value between 5 and 10.

end

set $T_2 = T_S - T_1$

set $T_3 = T_S = \varnothing$

end

In the algorithm 4.1, the set T_1, T_2, T_3, and T_S means the set of predecessors of current task, the set of current tasks, the set of successors of current task, and the set of successors of T2 respectively. The algorithm checks for each current task ($t_p \in T_2$) whether any of its successors is the element of its predecessors (T_1). In the case that the successor is the element of its predecessors, the algorithm checks whether the condition slot value of the procedural rule which connects from the current task to the successor is NULL. If the value is NULL, it concludes an occurrence of circularity. On the other hand, if the value is not NULL, it checks whether there exists any conflicting rule. The existence of any conflicting rule with the rule means that there happens a circularity if a workflow instance has a value in the duplicated region that are restricted by the two rules commonly.

ALGORITHM 4.2 (CHECKING MISSING VALUES)

Given a set of rule frames Rp,

for each task $t \in T$,

set Rp(t) = {pr ∈ Rp | pr.PRE_TASK = t}

set O(t) = { o ∈ O | o is restricted in pr.CONDITION and pr ∈ Rp(t)}

for each o ∈ O(t),

set IV(o) = {dom(pr.CONDITION|o) | pr.CONDITION|o is a projected condition of pr.CONDITION, which is restricted as a condition

```
                on object o}
        DIFF = dom(o)
        while IV(o) ≠ ∅
                DIFF = DIFF - dom( pr.CONDITION|o)
                IV(o) = IV(o) – dom( pr.CONDITION|o)
        end
        if DIFF ≠ ∅
                print “missing value for constraining o”
    end
end
```

The algorithm 4.2 determines whether there exists any missing value in a set of procedural-rule frames. The occurrence of missing values in a procedural rule set means that some parts of domain of an object, which is Cartesian product of domains of the object’s slots, are not used for defining routing rules after the completion of a task. To check missing values in a set of procedural-rule frames, following steps are followed. At first, for each task *t* in the task set *T*, all the procedural-rule frames that have task *t* as the value of the PRE_TASK slot are extracted. Secondly, all the objects that are used to define conditions in the CONDITION slot of the procedural-rule frames extracted in the first step are selected. Lastly, for each object in the second step, the union of restricted domains of the object that are determined by conditions of procedural-rule frames is calculated. If the union of restricted domain of the object is equal to the domain of the object, there is no missing value. Otherwise, the procedural-rule frames in the first step have missing values for the restriction of the object.

ALGORITHM 4.3 (CHECKING MISSING RULES AND DANGLING TASKS)

IMT = T - {INITIATE, TERMINATE}

For each $t \in$ IMT

pRp(t) = {pr $\in$ Rp | pr.PRE_TASK = t}

nRp(t) = {pr $\in$ Rp | pr.NEXT_TASK = t}

if pRp(t) = $\varnothing$

print “Task t has no successor”

else if nRp(t) = $\varnothing$

print ‘Task t has no predecessor”

else if nRp(t) = $\varnothing$ and nRp(t) = $\varnothing$

print “Task t is an isolated task”

The algorithm 4.3 is used for checking missing rules and dangling tasks. In the algorithm, for each task of the set of intermediary tasks (IMT), procedural_rule frames are identified that connect the task with its predecessors or successors. If there does not exist any procedural_rule frame that connects the task with its successor, the task is concluded as a dangling task. On the other hand, if the task has no predecessor, missing rules exist.

ALGORITHM 4.5 (CHECKING REDUNDANT RULES)

Given a set of procedural rule frames Rp,

for each pair $(t_1, t_2) \in$ T

set Rp(t_1, t_2) = {pr|$\exists$ pr $\in$ Rp s.t pr.PRE_TASK = $t_1 \wedge$ pr.NEXT_TASK = t_2}

if ∃ pr_1, pr_2 in Rp(t_1, t_2)
s.t. (IntersCond (pr1.CONDITION, pr2.CONDITION) = pr1.CONDITION) ∧
(DiffCond(pr2.CONDITION , pr1.CONDITION) ≠ ∅)
print “procedural rule pr2 is subsumed by procedural rule pr1”
end
else if ∃ pr_1, pr_2 in Rp(t_1, t_2)
s.t. (DiffCond(pr1.CONDITION , pr2.CONDITION) = ∅) ∧
(DiffCond(pr2.CONDITION , pr1.CONDITION) = ∅)
print “procedural rule pr1 is duplicated with procedural rule pr2”
end
end

ALGORITHM 4.6 (CHECKING CONFLICTING RULES)

Given a set of rule frames Rp,
for each task t ∈ T,
set Rp(t) = {pr ∈ Rp | pr.PRE_TASK = t}
set O(t) = { o ∈ O | o is restricted in pr.CONDITION and pr ∈ Rp(t)}
for each o ∈ O(t),
set IV(o) = {dom(pr.CONDITION|o) | pr.CONDITION|o is a projected
condition of pr.CONDITION, which is restricted as a
condition of object o}
set DIFF = dom(o)
set COMPL = ∅
while IV(o) ≠ ∅
if (COMPL ∩ dom(pr.CONDITION|o) ≠ ∅)
print “part of dom(pr.CONDITION|o) is duplicated”
set DIFF = DIFF - dom(pr.CONDITION|o)

```
                set COMPL = dom(o) - DIFF
                set IV(o) = IV(o) - dom(pr.CONDITION|o)
            end
        end
    end
```

Algorithm 4.5 identifies subsumed or duplicated rule frames. In the algorithm 4.6, the set COMPL is complementary of difference of restricted regions of object attribute values. If any of them duplicates with the set COMPL, the duplicated region results in confliction between two rules.

Chapter 5. Change Management in KWM

5.1 Introduction

In this chapter, a mechanism for managing changes on KWM is proposed. Changes are classified as schema-level change or instance-level change. A schema-level change modifies the structure (routing path and assignment of tasks to actors) of a workflow schema, and all the workflow instances that are running under the old workflow schema are changed to follow the new schema. Schema-level changes are usually, performed by workflow designers. The main issues of management of schema-level changes are version management of evolving workflow schema, assuring soundness of modified workflow schema, and migrating workflow instances that are running under old workflow schema into new schema. On the other hand, an instance-level change

modifies workflow schema for a workflow instance. The changes are usually performed by workflow participants of a workflow instance. In this case, the original workflow schema is used to generate normal workflow instances, and the modified workflow schema exists temporarily until the special workflow instance is terminated. Management of instance-level changes provides a mechanism for handling exceptions for special workflow instances. The main issues of management of instance-level changes are to provide modification primitives that derive the temporal workflow schema from original workflow schema without violation of soundness, a mechanism that monitors status of dynamically changed workflow instance, and a mechanism that undo a temporal change effect. The two types of change are summarized in Table 5.1.

Feature / Level	Trigger	Scope	Duration	Management Issues
Schema-level	WF Designer	All instance	Long-term	. Version Management . Maintaining soundness . Migration of workflow instances into new schema
Instance-level	WF Participant	An instance	Temporal	. Maintaining soundness . Status tracking . Propagation for “Undo”

Table 5.1 Classification of changes on workflow and their features.

The dependencies between frames are used for propagating change effects in KWM.

Section 5.2 identifies the main dependencies in KWM. The approaches for management of schema-level changes and instance-level changes are addressed in section 5.3 and 5.4.

5.2 Dependency Predicates

Predicates that represent dependencies among frames of KWM are listed in Table 5.2. Three types of dependency predicates are considered. The first one is predicate that represents dependencies between entity frames. The relationships that are explained in section 3.2 are transformed into predicates that represent dependencies between entity frames.

The second one is predicate that represents dependency between rule frames. Three predicates are considered; 'XOR-firing', 'AND-firing', and 'Substitute'. Only one of

Type	Predicate	Meaning
Entity vs. Entity	IS_A(o1, o2)	Object *o1* inherits from object *o2*
	SUBPART_OF(o1, o2)	Object *o1* is subpart of object2
	works_for(a, u, p)	Actor *a* work for organizational unit *u* with position p
	used_at(re, t)	Resource *re* is used at task *t*
	responsible_for(ro, t)	Role *ro* is responsible for task *t*
	…	…
Rule vs. Rule	XOR-firing(r_1, r_2,…r_n)	One of the rules r_1 , r_2, …, r_n can be fired
	AND-firing(r_1, r_2,…r_n)	All of the rules r_1 , r_2, …, r_n should be fired
	Substitute(r_1, r_2, rm)	Metarule *rm* substitute a set of rules r_1 with a set of rule r_2
Rule vs. Entity	Splitted(t_1, T_s, Rp_s)	Task set T_s is splitted from t_1 with procedural rule set Rp_s

	Jointed(t_1, T_j, Rp_j)	Task set T_j is jointed at t_1 with procedural rule set Rp_j
	Precedence(t_1, t_2, rp)	Task t_1 precedes t_2 with procedural rule rp
	Role-charge(ro, rr)	A responsibility-rule frame *rr* finds actors who are in charge of role *ro*.

Table 5.2 The predicates that represent dependencies among frames of KWM.

the rule frames that are used as arguments of the predicate 'XOR- firing' can be fired. On the other hand, the rule frames that are used as arguments of the predicate 'AND-firing' should be fired concurrently. The predicate 'Substitute' represents the substitution relationship between normal rules and special rules that are represented by a metarule frame.

The last type of predicate represents dependencies between entity and rule frames. The predicates 'Splitted', 'Joined', 'Precedence', and 'Role-charge' correspond to the type. The predicate 'Splitted' contains three arguments that represent a fork task (t_1), a set of splitted tasks (T_s), and a set of procedural-rule frames (Rp_s) respectively. It means that tasks in T_s are splitted from a fork task t_1, and procedural-rule frames in Rp_s connect the fork task with the splitted tasks. The predicate 'Joined' also contains three arguments. The first argument represents join task, and the second argument represents predecessors of the join task. The procedural-rule frames that are contained in the third argument connect the join task with its predecessors. The predicate 'Precedence' is derived from procedural-rule frame. It represents dependencies between ordered tasks and a procedural rule that define the order. On the other hand, the predicate 'Role-charge' is derived from responsibility-rule frame. It represents dependencies among a role, charged actors, and a responsibility-rule frame that define the mapping relationship.

The dependency predicates are derived from user defined workflow schema. For

example, algorithm 5.1 derives XOR-firing dependencies from a workflow schema. The algorithm for deriving XOR-firing dependencies between procedural-rule frames is to find a set of rule frames that exclusively constrain on the domain of the same objects.

The exclusive procedural-rule frames can be found from conditional routing constructs. The algorithm 5.1 constructs a set (*Rp(t1)*) of procedural-rule frames that should be checked after completion of a task. For each procedural-rule frame in the set *Rp(t1)*, the rule frame is added to a pseudo-exclusive rule set (*XOR(pr1)*). The other rule frames in the set *Rp(pr1)* that constrain the same objects with the procedural-rule frame are, then, successively compared to check whether their intersection of constrained domains of the objects is null or not. If the intersection is null, the procedural-rule frames are added to the set pseudo-exclusive rule set.

ALGORITHM 5.1 (FINDING EXCLUSIVE PROCEDURAL-RULE FRAMES)

Given a set of procedural-rule frames Rp,

$\forall t, t \in T$ where T is set of tasks,

$Rp(t1) = \{pr \in Rp \mid pr.PRE\_TASK = t\}$

$\forall pr_1, pr_1 \in Rp(t)$,

$O(pr_1) = \{o \mid o$ is an object whose domain is restricted in pr_1.CONDITION$\}$

$$Rp(pr_1) = \{pr \mid pr \in Rp(t1), O(pr)=O(pr_1), \text{where } O(pr) \text{ is defined as similar with } O(pr_1)\}$$

$XOR(pr_1) = \{ pr_1\}$

$\forall pr, pr \in Rp(pr_1)$,

if $pr \wedge (\wedge_{pri \in XOR(pr1)} pri.CONDITION) = \varnothing$

$pr \in XOR(pr_1)$

if $(\vee_{pri \in XOR(pr1)} pr_i.CONDITION = X_{oi \in O(pr1)} dom(O_i)$ where X means cartesian product)

exit

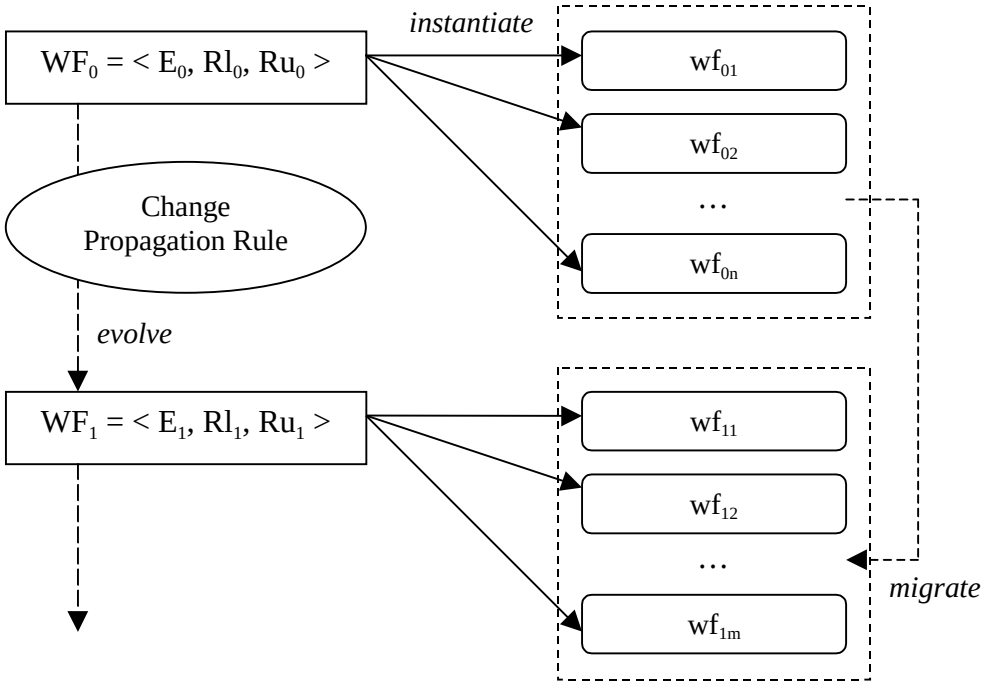
if $\vee_{pri \in XOR(pr1)} pr_i.CONDITION = X_{oi \in O(pr1)} dom(O_i)$

Add predicate XOR-firing(pr1, pr2, …, prn) for all pr1, pr2, … , prn ∈ XOR(pr1)

If the comparison is finished for all other rule frames, the union of the constrained domain of rule frames in pseudo-exclusive rule set is calculated. If the union is the same with the entire domain of the objects, then the set of procedural-rule frames constitutes an XOR-firing dependency. The algorithm for deriving XOR-firing dependencies between responsibility-rule frames is similar with the algorithm 5.1. The difference is to compose the set *Rp(t1)* with the responsibility-rule frames that have the same value in the ROLE slot.

5.3 Management of schema-level changes

Schema-level changes transform from old version of workflow schema to a new version. As shown in Figure 5.1, all modifications on a workflow schema should be



[Figure 5.1] Workflow schema evolution

followed with propagation of the change effects to assure soundness of the new workflow schema. Furthermore, the workflow instances that are running under old version should be able to migrate to the new version.

In this section, change propagation rules for workflow modification primitives are proposed. The focus is mainly on the structural changes such as insertion and deletion of tasks in a workflow schema. The propagation rules for the changes of exceptional rules that affect other rules, however, are also suggested. The propagation rules are based on the dependency predicates introduced in the former section. Using the predicates in Table 5.2, change propagation scope is identified, and proper update on the affected frames by the change is performed. The change propagation rules, then, are used to automatically modify frames of KWM or notify model builder the anomalies caused by the changes. At the last part of this section, policies that can be adopted for workflow instance migration are reviewed, and an approach for workflow migration is proposed.

5.3.1 Propagation rules for changes on tasks

The changes on tasks that should be considered are insertion of a new task, deletion of an existing task, and altering order between tasks. The changes mainly affect procedural-rule frames that have dependencies with the tasks. The propagation rules consist of antesequents (marked by IF) and consequents (marked by THEN). An antesequent is composed of a modification primitive and dependency predicates. A consequent is composed of one or more actions that should be performed in order.

(1) Insertion of a new task

Six cases of task insertion can be considered.

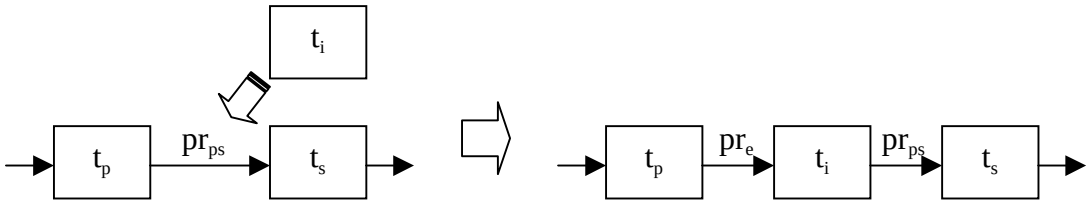
- AddSeqTask (predecessor t_p, successor t_s, task t_i) : inserts a new task t_i between two sequential tasks t_p and t_s in the schema. Figure 5.2 describes the effect of this primitive.

IF : AddSeqTask (t_p, t_s, t_i) $\wedge$ Precedence(t_p, t_s, pr_{ps})

THEN: update pr_{ps}.PRE_TASK = t_i;

$Rp = Rp \cup \{pr_e\}$

where $pr_e \notin Rp \wedge pr_e$.NEXT_TASK = $t_i \wedge pr_e$.PRE_TASK = t_p;



[Figure 5.2] Inserting a task between two sequential tasks

- AddSplitTask(preTask t_p, sucTask t_s, task t_i, Condition cond): insert a new task t_i between two tasks t_p and t_s as a conditional successor of t_p in the schema. In this case, t_p is a splitting task and t_s is a join task. As shown in Figure 5.3, this primitive insert a new branch from splitting task t_p.

Case 1;

IF: AddCondTask(t_p, t_s, t_i, cond) $\wedge$ Splitted(t_p, T_s, Rp_s) $\wedge$ XOR-firing(Rp_s)

THEN:

$\forall pr, pr \in Rp_s$, update pr.CONDITOIN;

$Rp = Rp \cup \{pr_e\}$ where $pr_e \notin Rp \wedge pr_e$.NEXT_TASK = t_i

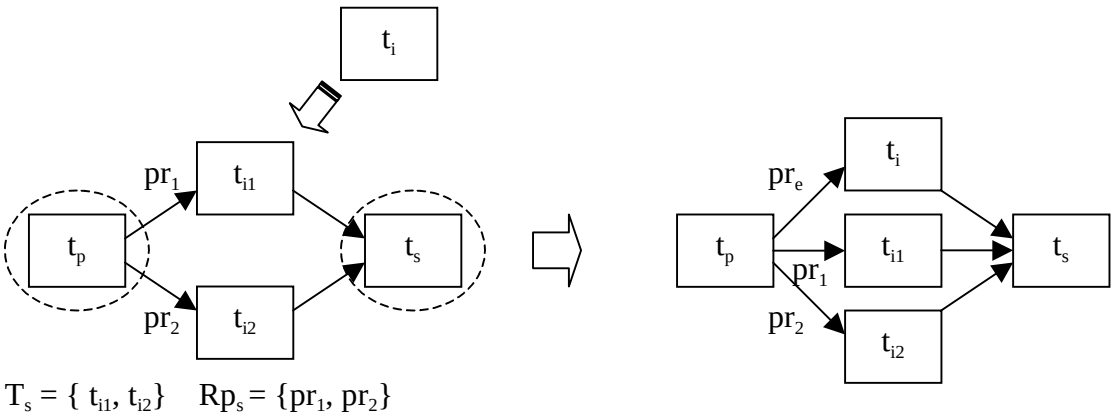
$\wedge pr_e$.PRE_TASK = $t_p \wedge pr_e$.CONDITOIN = cond;

Case 2;

IF: AddCondTask(t_p, t_s, t_i, cond) $\wedge$ Splitted(t_p, T_s, Rp_s) $\wedge$ AND-firing(Rp_s)

THEN: $Rp = Rp \cup \{pr_e\}$ where $pr_e \notin Rp$

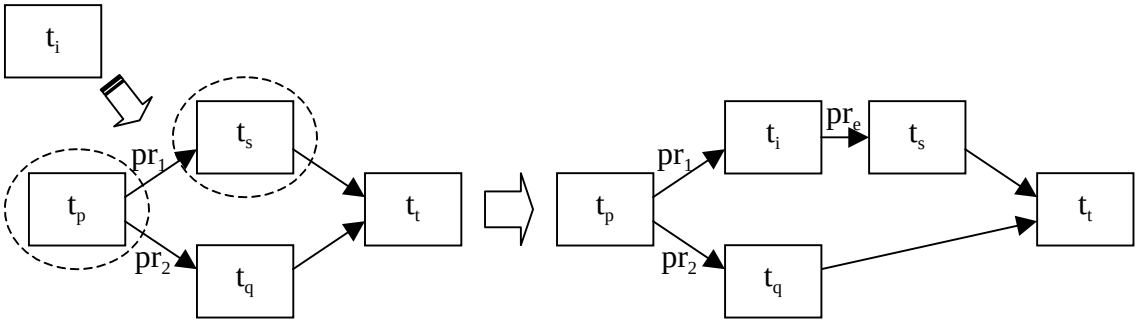
$\wedge pr_e$.NEXT_TASK = $t_I \wedge pr_e$.PRE_TASK = t_p



[Figure 5.3] Inserting a task between conditional routing

- AddInterFolkTask(preTask t_p, sucTask t_s, task t_i): insert a new task t_i between two

tasks t_p and t_s that is splitted from t_p as a sequential predecessor of t_s in the schema. In this case, t_i becomes a splitting task from t_p instead of t_s (Figure 5.4).



$T_s = \{ t_s, t_q\}$ $Rp_s = \{pr_1, pr_2\}$

[Figure 5.4] Inserting a task between a fork task and its successor

IF : AddInterFolkTask(t_p, t_s, t_i) $\wedge$ Splitted(t_p, T_s, Rp_s) $\wedge$ $t_s \in T_s$

THEN : Update pr.NEXT_TASK = t_i where pr $\in$ Rp $\wedge$ pr.NEXT_TASK = t_s;

Rp = Rp $\cup$ {pr_e} where $pr_e \notin$ Rp $\wedge$ pr_e.NEXT_TASK = t_s

$\wedge$ pr_e.PRE_TASK = t_i;

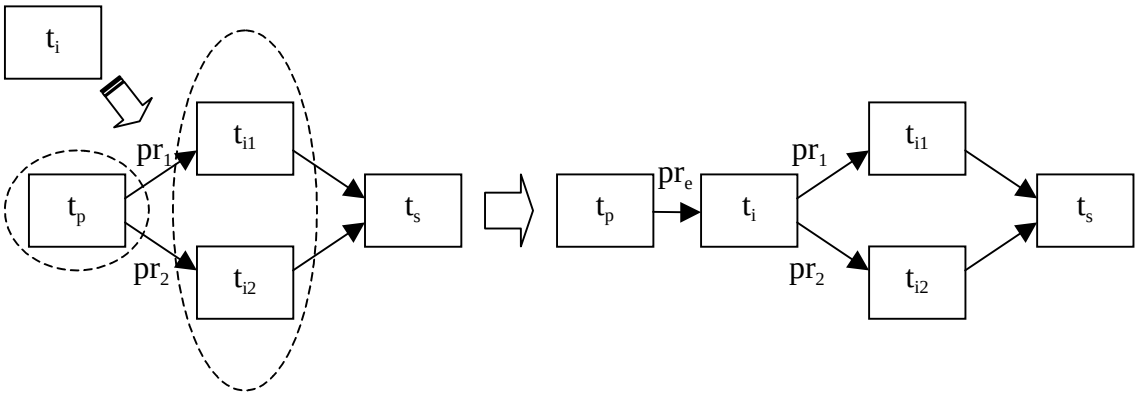
- AddFolkTask(preTask t_p, sucTask T_s, task t_i): insert a new task t_i between a fork task t_p and a set of splitted tasks T_s in the schema. As shown in Figure 5.5, the inserted task t_i becomes the fork task instead of t_p.

IF : AddFolkTask(t_p, T_s, t_i) $\wedge$ Splitted(t_p, T_s, Rp_s)

THEN : $\forall$pr, pr $\in$ Rp_s, update pr.PRE_TASK = t_i;

$Rp = Rp \cup \{pr_e\}$ where $pr_e \notin Rp \wedge pr_e.NEXT\_TASK = t_i$

$\wedge\ pr_e.PRE\_TASK = t_p$;



$T_s = \{ t_{i1}, t_{i2}\}$ $Rp_s = \{pr_1, pr_2\}$

[Figure 5.5] Inserting a task between conditional routing

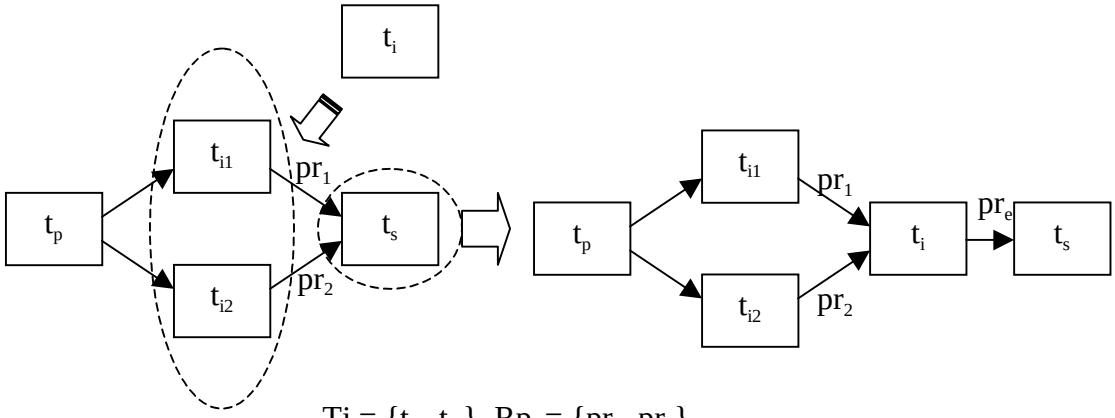
- AddJoinTask(preTask T_j, sucTask t_s, task t_i): insert a new task t_i between a set of splitted tasks T_j and a join task t_s in the schema. As shown in Figure 5.6, the inserted task t_i becomes the join task instead of t_s.

IF : AddJoinTask(T_j, t_s, t_i) $\wedge$ Splitted(t_s, T_j, Rp_j)

THEN : $\forall pr, pr \in Rp_j$, update pr.NEXT_TASK = t_i

$Rp = Rp \cup \{pr_e\}$ where $pr_e \notin Rp \wedge pr_e.NEXT\_TASK = t_s$

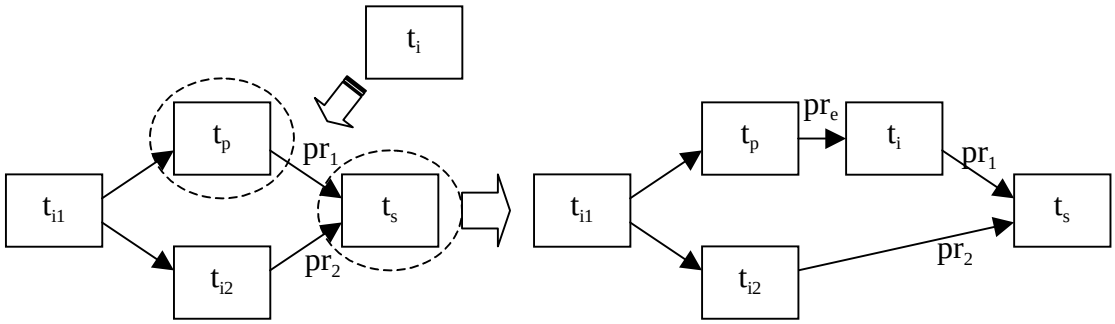
$\wedge\ pr_e.PRE\_TASK = t_i$;



$Tj = \{t_{i1}, t_{i2}\}, Rp_j = \{pr_1, pr_2\}$

[Figure 5.6] Inserting a task between conditional routing

- AddInterJoinTask(preTask t_p, sucTask t_s, task t_i) : insert a new task t_i between two tasks t_p and t_s that is a join task of t_p, as a sequential successor of t_p in the schema. In this case, t_i becomes a predecessor of join task t_s instead of t_p (Figure 5.7).



[Figure 5.7] Inserting a task between a join task and its predecessor

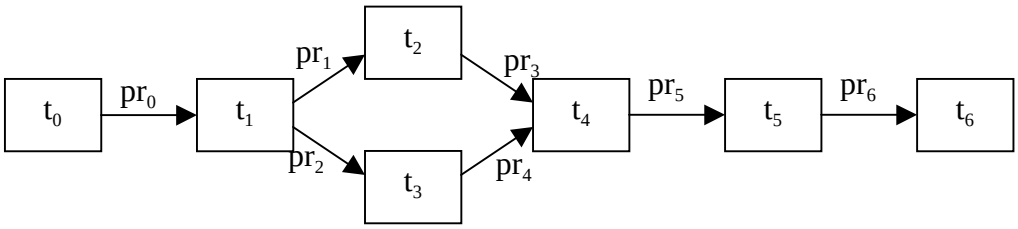
IF : AddInterJoinTask(t_p, t_s, t_i) $\wedge$ Joined(t_s, T_j, Rp_j) $\wedge$ $t_p \in T_j$

THEN : Update pr.PRE_TASK = t_i where pr $\in$ Rp $\wedge$ pr.PRE_TASK = t_p;

$$Rp = Rp \cup \{pr_e\} \text{ where } pr_e \notin Rp \wedge pr_e.NEXT\_TASK = t_i$$
$$\wedge\ pr_e.PRE\_TASK = t_p;$$

(2) Deletion of a task

When a task is deleted from workflow schema, the effects of the change are different according to the position of the task in the workflow net. Five cases can be considered; i.e. the task is in the middle of a sequential routing, the task is fork task, the task is join task, the task is a splitted task from its predecessor, and the route is joined at



[Figure 5.8] An example workflow for task deletion

its successor. The five cases are explained using an example workflow in Figure 5.8.

- Deletion of a task (t_5) in the middle of a sequential routing:

  IF : Delete(t_5) $\wedge$ Precedence(t_4, t_5, pr_5) $\wedge$ Precedence(t_5, t_6, pr_6)

  THEN : $Rp = Rp - \{ pr_5 \}$; update $pr_p.NEXT\_TASK = t_s$;

- Deletion of a fork task (t_1): predecessor of the deleted fork task becomes fork task instead of the deleted task.

IF : Delete(t_1) $\wedge$ Precedence(t_0, t_1, pr_0) $\wedge$ Splitted(t_1, {t_2, t_3}, {pr_1, pr_2})

THEN : Rp = Rp – { pr_0 }; update pr_1.PRE_TASK = t_0; update pr_2.PRE_TASK = t_0;

- Deletion of a join task(t_4): successor of the deleted join task becomes join task instead of the deleted task.

IF : Delete(t_4) $\wedge$ Precedence(t_4, t_5, pr_5) $\wedge$ Joined(t_4, {t_2, t_3}, {pr_3, pr_4})

THEN : Rp = Rp – { pr_5 }; update pr_3.NEXT_TASK = t_5;

update pr_4.NEXT_TASK = t_5;

- Deletion of a successor of a fork task (t_2): procedural-rule frames that connect the task with its predecessor and successor are removed. Furthermore, some dependency predicates should be modified. Two cases should be considered; i.e. whether the procedural-rule frame that connects the task with its predecessor has XOR-firing relationship or AND-firing relationship with other rule frames.

Case 1:

IF : Delete(t_2) $\wedge$ Precedence(t_p, t_2, pr_p) $\wedge$ Precedence(t_2, t_s, pr_s)

$\wedge$ Splitted(t_p, T_p, Rp_p) $\wedge$ ($t_2 \in T_p$) $\wedge$ XOR-firing(Rp_p)

THEN : Rp = Rp – { pr_p, pr_s }; Rp_p = Rp_p – { pr_p};

$\forall$pr, pr $\in$ Rp_p , update pr.CONDITION

Case 2:

IF : Delete(t_2) $\wedge$ Precedence(t_p, t_2, pr_p) $\wedge$ Precedence(t_2, t_s, pr_s)

$\wedge$ Splitted(t_p, T_p, Rp_p) $\wedge$ ($t_2 \in T_p$) $\wedge$ AND-firing(Rp_p)

THEN : $Rp = Rp - \{ pr_p, pr_s \}$; $Rp_p = Rp_p - \{ pr_p \}$;

- Deletion of a predecessor of a join task (t_2): procedural-rule frames that connect the task with its predecessor and successor are removed. Furthermore, some dependency predicates should be modified.

IF : $Delete(t_2) \wedge Precedence(t_p, t_2, pr_p) \wedge Precedence(t_2, t_s, pr_s)$
$\wedge Joined(t_s, T_s, Rp_s) \wedge (t_2 \in T_s)$
THEN : $Rp = Rp - \{ pr_p, pr_s \}$; $Rp_s = Rp_s - \{ pr_s \}$;

(3) Modify operation

Modify operations that can trigger other operations are modification of condition-slot value of a rule frame and changing the order between tasks.

- ModifyCondition (procedural-rule pr, Condition cond) : This primitive substitute existing value of condition-slot of rule frame ‘pr’ with new value ‘cond’. In this case, condition values of other rule frames that have XOR-firing relationship with the rule frame should be also modified. The following rule propagate the change effects.

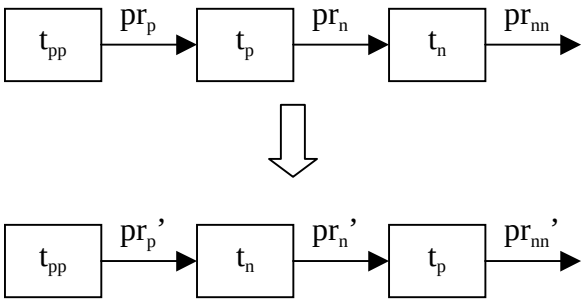
IF $ModCondition(pr, cond) \wedge XOR\text{-}firing(Rp_p) \wedge (pr \in Rp_p)$
THEN $\forall pr_n, pr_n \in Rp_p \wedge pr_n \neq pr$, update pr_n.CONDITION

- AltTaskOrder (preTask t_p, nextTask t_n): this primitive alter an order between two sequential tasks. Altering an order between two tasks are permitted to tasks that are connected only sequentially.

IF : AltTaskOrder(t_p, t_n) $\wedge$ Precedence(t_{pp}, t_p, pr_p) $\wedge$ Precedence(t_p, t_n, pr_n) $\wedge$ Precedence(t_n, t_{nn}, pr_{nn})

THEN: $Rp = Rp \cup \{pr_p', pr_n', pr_{nn}'\}$ where $(pr_p', pr_n', pr_{nn}' \notin Rp) \wedge$ $(pr_p'.PRE\_TASK = pr_p.PRE\_TASK) \wedge (pr_p'.NEXT\_TASK = t_n) \wedge$ $(pr_n'.PRE\_TASK = pr_n.NEXT\_TASK) \wedge (pr_n'.NEXT\_TASK = pr_n.PRE\_TASK) \wedge$ $(pr_{nn}'.PRE\_TASK = t_p) \wedge (pr_{nn}'.NEXT\_TASK = pr_{nn}.PRE\_TASK)$;

$Rp = Rp - \{pr_p, pr_n, pr_{nn}\}$;



[Figure 5.9] Changing order between two sequential tasks.

5.3.1 Propagation rules for changes on exceptional rules

One of the complex changes that affect other constructs is the change on the exceptional rule. In general, the exceptional rules are changed more frequently than workflow structure that is composition of tasks. They are easily affected by the change of organizational structure, and policy for doing business in turbulent environment. In KWM, the special workflow instances that deviate from normal workflow schema are processed by metarule frames. Insertion or deletion of metarule frames affect existence of other rule frames or dependency predicates. For example, the deletion of a task from a workflow schema is chained to deletion of procedural-rule frames. If a metarule frame has the procedural-rule frames as the value of SOURC_RULE slot, the metarule frame should be removed.

The change primitives for exceptional rules are as follows.

- ModChange(rm, cond) : This primitive modifies the value of CONDITION slot of a metarule frame rm. This means that the condition that classify special workflow instances is changed. In this case, the effect of the change is relatively small. The value of the CONDITION slot of a metarule that handles the exceptional rule should be changed and the change does not affect other frames.

  IF ModCondition(rm, cond)

  THEN update rm.CONDITION = cond;

- AddMetaRule(metarule rm, rule Ru, Entity E, Relationship Rl) : This primitive inserts a new exceptional rule. Additional exceptional rule can be added for routing work items or assigning tasks to actors. To handle the change a metarule frame and additional exceptional rule frames should be added to rule set.

  IF AddMetaRule(rm, Ru_e, E_e, Rl_e)

  THEN $Rm = Rm \cup \{rm\}$; $Ru = Ru \cup Ru_e$; $E = E \cup E_e$; $Rl = Rl \cup Rl_e$

- RemoveMetaRule(metarule rm) : This primitive removes an exceptional rule. An exceptional rule can be removed because organizational policy is changed. In this case, the metarule frame and exceptional rule frames which handle the exception should be removed from rule set.

  IF RemoveMetaRule(rm) $\wedge$ Substitute(R_1, R_2, rm)

  THEN Rm = Rm - {rm}; Ru = Ru – R_2;

- Changes on the rules in the TARGET_RULE slot :

5.3.2 Propagation rules for changes on responsibility rules

The assignment of tasks to actors is mainly changed because of management techniques like empowerment and restructuring. As a result of applying the techniques, organizational members take new roles and delegate some roles to others. These changes can be propagated through responsibility-rule frames. Organizational changes that affect responsibility rules are as follows.

- Task delegation : Usually some tasks are performed in several departments in a large organization. For example, the task “Grant trip allowance” is performed in department of finance, department of research management, and the academic & student services departments of each school at KAIST. The business rule can be changed to delegate the task of department of research management to department of finance.
- Merge of departments: In this case, tasks that are performed in a department is delegated into new department to assure continuity of workflows.
- Changes of role characteristics : Usually, actors who are in charge of a role need special skills or organizational position for executing some tasks. The needs can be changed as organization evolves.

The changes on the responsibility rules can be propagated by removing (or inserting) responsibility-rule frame, or modifying conditions of responsibility-rule frames.

5.3.4 An illustrative example

The business trip approval process in Figure 3.7 has been changed as a result of BPR project at KAIST. At first, the task "Check Trip Request Form" executed by a department officer is going to be removed from the process because computerized form processing system automatically checks the correctness of the form (Event1). Secondly, the task "Grant Traveling Allowance" is going to be executed only in the finance department and the academic & student services department (deletion of rule rr8-3), and

- For Event1 = DELETED(Check_Trip_Request_Form),

(1) DELETED(Check_Trip_Request_Form) ∧
Precedence(Check_Trip_Request_Form, Approve Subordinator's_Trip, rp7) ∧
Precedence(Approve_Traveling_Allowance, Check_Trip_Request_Form, rp5)
⇒ DELETED(rp5) ∧ DELETED(rp7)
DELETED(rp5) ∧ XOR-firing(rp5, rp6)
⇒ UPDATE-SLOT(rp6.CONDITION)
// In this case, condition spec. of rp6 should be removed

(2) DELETED(Check_Trip_Request_Form) ∧
Charged(Check_Trip_Request_Form, Department_Officer)
⇒ DELETED(Department_Officer)
DELETED(Department_Officer) ∧ Role-charge(Department_Officer, rr5)
⇒ DELETE(rr5)
// rr5 is a responsibility rule for the role Department Officer

- For Event2 = DELETED(rr8-3),

DELETED(rr8-3) ∧ XOR-firing(rr8-1, rr8-2, rr8-3, rr8-4)
⇒ UPDATE-SLOT(rr8-1.CONDITION) ∧ UPDATE-SLOT(rr8-2.CONDITION)
∧ UPDATE-SLOT(rr8-4.CONDITION)

- For Event3 = DELETED(rm3),

DELETED(rm3) ∧
Substitute({rp2,rp3,rp4,rp5,rp6,rp7,rp8,rp9},{rp-s1, rp-s2, rp-s3, rp-s4, rp-s5},rm3)
⇒ DELETE(rp-s1, rp-s2, rp-s3, rp-s4, rp-s5)

[Figure 5.10] Change propagation chains for the example workflow.

the mapping conditions of other rule frames that are related with XOR-firing dependency should be changed (Event2). Lastly, the exceptional rule for delegated employees is going to be removed, and the workflow instances for the trip of delegated employees should be processed as other normal instances (Event3). Figure 5.10 shows the change propagation chains using the propagation rules. The change propagation chains are used to notify workflow modeler the frames that should be updated.

5.3.5 Migration of workflow instances into new schema

Casati et al. (1998) suggested the policies the workflow administrator (WFA) can adopt to manage running workflow instances upon a modification of their schema. They are summarized as follows:

- *Abort*: all instances of old schema are aborted, and the newly created instances will start following new schema;
- *Flush*: all existing instances terminate following old schema. In the meantime, no new instance of old schema will be started. When all instances are finished, new instances can start following new schema
- *Progressive*: different decisions for different instances are taken, according to instance state or instance history. Multiple schema versions may exist at the same time.

The three policies can be adopted in K-WFMS to manage running instances. The first two solutions are the simplest from the WFMS point of view, since only one version of the schema exists at the same time, and all active instances therefore follow the same schema version. From the client point of view, the first two solutions are, however, inconvenient, since they should initiate instances again or could not initiate instances during the schema-evolution period. In this section, an approach that adopts the third policy is detailed.

To define instance migration rules, workflow instance history should be defined.

DEFINITION 5.1 (INSTANCE HISTORY) An instance history is a 9-tuple,

$$H_I = < t_i, s, d_i, d_c, \alpha, r_\alpha, t_p, D, O_d >$$

where t_i is task instance, s is state of t_i, d_i is initiated date of t_i, d_c is completed date of t_i, α is actor identifier, r_α is the role which the actor α is in charge of t_i, t_p is a task instance which transited work item to t_i, D is a set of data instances processed by t_i, and O_d is a set of operations on data items.

Whether a workflow instance actually can be migrated to new version, depends on the current states of the workflow instance. In K-WFMS, the migration of running instances to the new version is restricted only to that do not violate consistency of organizational state. Organizational state is represented by entity and relationship frames in KWM. Actors in a workflow read or update organizational state during executing assigned tasks to them. Migration of workflow instances should be determined based on the dependencies between tasks which update data instances. For example, a running workflow instance that has a completed task instance which wrote to a data item and the task schema is removed from new schema could not be migrated

to the new version. The migration rules are defined according to the modification primitives that are applied to old version resulting in new version. The changes that affect the migration decision are that affect workflow schema that is composed with predecessors of currently initiated tasks. For each workflow instance, the task schema that compose a workflow schema are classified as following three categories:

- The set of task schema which any of their instances are member of instance history, and the state of the task instance is “initiated” or “pended”, i.e.,

  $\mathbf{T_{initiated}} = \{ \varphi(h_i.t_c) \mid (h_i \in H_I) \wedge (h_i.s = \text{“INITIATED”} \vee h_i.s = \text{“PENDED”}) \}$

  where $\varphi(t)$ is a function which returns task schema of a task instance t.
- The set of task schema which any of their instances are member of instance history, and the state of the task instance is “completed”, i.e.,

  $\mathbf{T_{before}} = \{\varphi(h_i.t_b) \mid (h_i \in H_I) \wedge (h_i.s = \text{“COMPLETED”})\}$
- The set of task schema which are successor of currently initiated or pended tasks, i.e., $\mathbf{T_{after}} = \{ T_a \mid \forall T_i \in T_{initiated}, T_a \in Succ(T_i)\}$

  where Succ(T) is a function which returns a set of successors of T.

For each workflow instance, the modification primitives that affect task schema in $\mathbf{T_{initiated}}$, $\mathbf{T_{after}}$, or $\mathbf{T_{before}}$ are checked to determine whether the workflow instance can be migrated to new version or not. The cases considerable are as follows;

- Addition of new tasks to $\mathbf{T_{after}}$: When the tasks are added into $\mathbf{T_{after}}$, the workflow instance can be migrated to new version if any of added tasks does not read from data instance which any predecessor of currently initiated task have been updated. That is, if following condition is satisfied, the workflow instance can not be migrated to new version.

$(WF_{new} = WF_{old} \cup T_{new}) \wedge (T_s \in T_{new}) \wedge (\forall T_i, T_s \in Succ(T_i)) \wedge (T_p \in T_{before}) \wedge$ writes(T_p, d) $\wedge$ reads(T_s, d)

- Addition of tasks in $\mathbf{T_{initiated}}$: If tasks are added to old version as siblings of currently initiated task, the workflow instance can be migrated to new version by initiating added tasks.
- Deletion of tasks in $\mathbf{T_{initiated}}$: If tasks are deleted from old version, the workflow isntance can be migrated to new version by aborting the initiated task.
- Deletion of tasks : Any of predecessors of currently initiated task is deleted from old version. In this case, the workflow instance can be migrated if the deleted task schema is not element of $\mathbf{T_{before}}$. That is, if following condition is satisfied, it can not be migrated to new version.

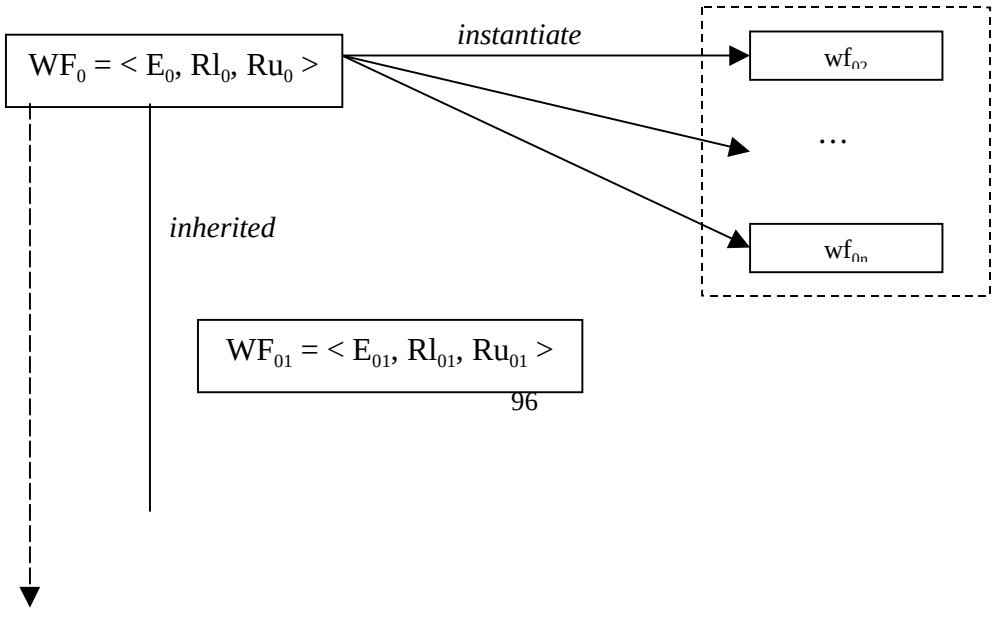
  $(WF_{new} = WF_{old} - T_o) \wedge (T_o \subseteq \mathbf{T_{before}})$
- Changing routing conditions : if the history of the workflow instance is changed because of the changed routing conditions, the workflow instance can not be migrated. Otherwise, it can be migrated to new version.
- Changing order between two tasks: If the taks are elements of $\mathbf{T_{after}}$, the workflow instance can be migrated to new version.

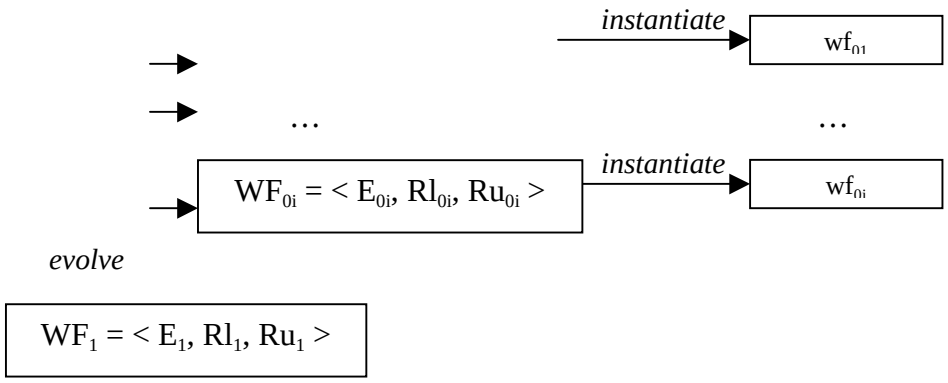
5.4 Management of instance-level changes

Management of instance-level changes is critical issue for increasing adaptability of WFMS. The effects of the change exist temporarily and are abolished after termination of the workflow instance. Using the mechanism which manages instance-level changes, workflow participants are able to deviate from premodeled task sequences of a workflow at run-time. In this dissertation, the basic ideas for handling instance-level

changes are inheriting original workflow schema and dynamic interpretation of workflow schema. Inheritance of workflow schema means that the workflow schema that are applied to a special workflow instance inherits frames from original workflow schema and additional frames that are needed to handle exception of the workflow instance are added. With this approach, the original workflow schema is not affected from the instance-level changes, and various version of workflow schema for special workflow instances coexist with the original workflow schema. This results in the needs for dynamic interpretation of workflow schema. Dynamic interpretation of workflow schema means that for each time a workflow schema is interpreted, frames can be added or removed dynamically. This is the characteristic of rule-based system.

To manage instance-level changes, three mechanisms should be considered. At first, system should guarantee that all consistency constraints that have been ensured prior to a dynamic change are also ensured after the modification. Secondly, the mechanism for providing workflow status tracking service is also important as temporal existence of workflow tasks. Lastly, an authorization mechanism that permits the instance-level changes only to authorized workflow roles. As the provision of the authorization mechanism is less critical to discuss than mechanism for handling the effect of instance-level changes, we do not consider them further in this research.





[Figure 5.11] Inheritance of workflow schema for handling instance-level changes

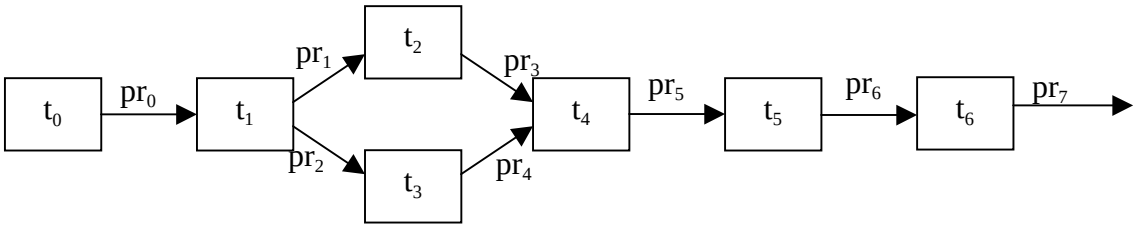
In Figure 5.11, the workflow schema WF_{01} ~ WF_{0i} inherit from original workflow schema and additional rule frames are added according to participants' modification of a workflow instance. The additional rule frames are stored in another knowledge set, and they are merged with original workflow schema in the middle of progress of the workflow instance.

5.4.1 Propagation rules for dynamic workflow configuration

The instance-level changes that should be considered can be classified as follows; (1) skip some tasks, (2) Insert undefined tasks, (3) Delegate assigned tasks to other actors, and (4) Change sequences between tasks

The instance-level changes that are requested by workflow participants are handled using temporal metarule frames. The temporal metarule frames are created in the middle of a workflow instance execution and abolished after termination of it. In the remained part of this section, propagation rules that generate the temporal rule frames are

explained for each modification primitives for instance-level changes. Figure 5.12 is used as an example workflow. The modification primitives whose meanings are explained in section 5.3 are used again.



[Figure 5.12] An example workflow for instance-level changes

(1) Skip some tasks : A workflow participants can change workflow schema for a workflow instance by skipping some successors of currently activated task. To skip a task, delete primitive can be used. For the request, following rule create a temporal metarule frame and additional procedural rule frames.

IF Delete(t_4) $\wedge$ Precedence(t_2, t_4, pr_3) $\wedge$ Precedence(t_3, t_4, pr_4) $\wedge$

Joined(t_4, {t_2, t_2}, {pr_2, pr_3})

THEN

$Rp_i = Rp_i \cup \{pr_3', pr_4'\}$ where $pr_3', pr_4' \notin Rp \wedge$

(pr_3'.PRE_TASK = pr_3.PRE_TASK) $\wedge$ (pr_4'.PRE_TASK =

pr_4.PRE_TASK) $\wedge$ (pr_3'.NEXT_TASK = t_5) $\wedge$ (pr_4'.NEXT_TASK = t_5);

$Rm_i = Rm_i \cup \{rm\}$ where $rm \notin Rm_i \wedge$

rm.SOURCE_RULE = {pr_3, pr_4, pr_5} ∧ rm.TARGET_RULE = {pr_3', pr_4'};

(2) Insert additional tasks : In this case, the insert primitives that are discussed in section 5.3.1 can also be considered. The propagation rules for the primitives, however, are different with those of them in that they add additional rule frames and do not update any existing rule frames.

- AddSeqTask (t_5, t_6, t_i) :

  IF : AddSeqTask (t_5, t_6, t_i) ∧ Precedence(t_5, t_6, pr_6)

  THEN:

  $Rp_i = Rp_i \cup \{pr_{5i}, pr_{i6}\}$ where $pr_{5i}, pr_{i6} \notin Rp$

  (pr_{5i}.PRE_TASK = pr_6.PRE_TASK) ∧ (pr_{5i}.NEXT_TASK = t_i) ∧

  (pr_{i6}.PRE_TASK = t_i) ∧ (pr_{i6}.NEXT_TASK = pr_6.PRE_TASK);

  $Rm_i = Rm_i \cup \{rm\}$ where $rm \notin Rm$ ∧ rm.SOURCE_RULE = {pr_6} ∧

  rm.TARGET_RULE = {pr_{5i}, pr_{i6}};

(3) Change order between tasks

The order between two tasks can be changed only if the tasks are sequentially connected.

- AltTaskOrder(t_5, t_6) :

  IF : AltTaskOrder(t_5, t_6) ∧ Precedence(t_4, t_5, pr_5) ∧ Precedence(t_5, t_6, pr_6) ∧

Precedence(t_6, t_7, pr_7)

THEN:

$Rp_i = Rp_i \cup \{pr_5', pr_6' , pr_7'\}$ where $(pr_5', pr_6' , pr_7' \notin Rp) \wedge$

$(pr_5'.PRE\_TASK = pr_5.PRE\_TASK) \wedge (pr_5'.NEXT\_TASK = t_6) \wedge$

$(pr_6'.PRE\_TASK = pr_6.NEXT\_TASK) \wedge (pr_6'.NEXT\_TASK = pr_6.PRE\_TASK) \wedge$

$(pr_7'.PRE\_TASK = t_5) \wedge (pr_7'.NEXT\_TASK = pr_7.PRE\_TASK)$;

$Rm_i = Rm_i \cup \{rm\}$ where $rm \notin Rm \wedge rm.SOURCE\_RULE = \{pr_5, pr_6, pr_7\}$

$\wedge rm.TARGET\_RULE = \{ pr_5', pr_6' , pr_7'\}$;

(4) Delegate assigned tasks to other actors

- DelegateTask (task t_i, actor a_j, role r_k) : this primitive delegates a task instance to other actor with role.

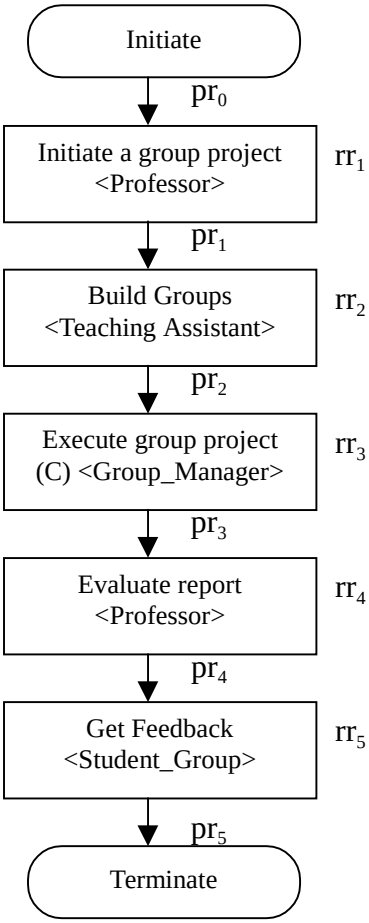
(5) Decomposition of a task : Decomposing a running task instance is explained in the following sub section using example workflow.

5.4.2 An illustrative example

Figure 5.13 shows an ad-hoc process. The goal of the process is to support execution of students' group project in virtual classroom. A workflow instance is created when a professor notifies a group project by filling in a group project notification form ("Initiate a group project"). After completion of the first task, a teaching assistant builds groups of students. After that, each student group prepares

report for the project. If due date of the project is over, the professor evaluates the reports and marks grades for the reports. Finally, each student gets feedback from the professor.

A special requirement for supporting the example workflow is that each student group should configure partial workflows for preparing report for the project. That is, the task “Execute group project” is assigned to manager of each group, and the group managers should define sub tasks and sequence among them and assign new tasks to their group members for executing the assigned task. In this case, each group manager dynamically configures its own workflow instance during the execution of workflow instance. For example, assume there are three groups and group project notification form is used to carry information for executing the group project. The original workflow



[Figure 5.13] An example process for group project in a virtual classroom

schema is as follows;

WF_{old} = {T, Re, Ro, A, U, Rp, Rr, Rm} where

T = {Initiate_Group_Project, Build_Groups, Execute_Group_Project,

Evaluate_Report, Get_Feedback},

Re = {Group_Project_Notification_Form, Report_Evaluation_Form},

Ro = {Professor, TA, Group_Manager, Student},

A = {Class_Member, Professor}, U = {∅}

Rp = {pr_0, pr_1, pr_2, pr_3, pr_4, pr_5}, Rr = {rr_1, rr_2, rr_3, rr_4, rr_5}, Rm = {∅}

A professor “Park” initiate a workflow instance by filling in some fields (class id, subject, due date etc.) of Group_Project_Notification_Form (assume its instance id is GPN00001). The teaching assistant “Lee” (determined by responsibility rule rr_2) of the class, then, builds groups for executing the project and assigns managers for the groups (i.e., fills in group session fields of the routed form). After the completion of the second task, the form is routed to group managers (“Cho”, “Ahn”, and “Kim” are determined by responsibility rule rr_3). The instance history of the example workflow is as follows;

H_I = < t_i, s, d_i, d_c, α, r_α, t_p, D, O_d >

<t_1, “COMPLETED”, “1998/10/5”, “1998/10/5”, “Park”, Professor, t_0, {GPN00001}, write>

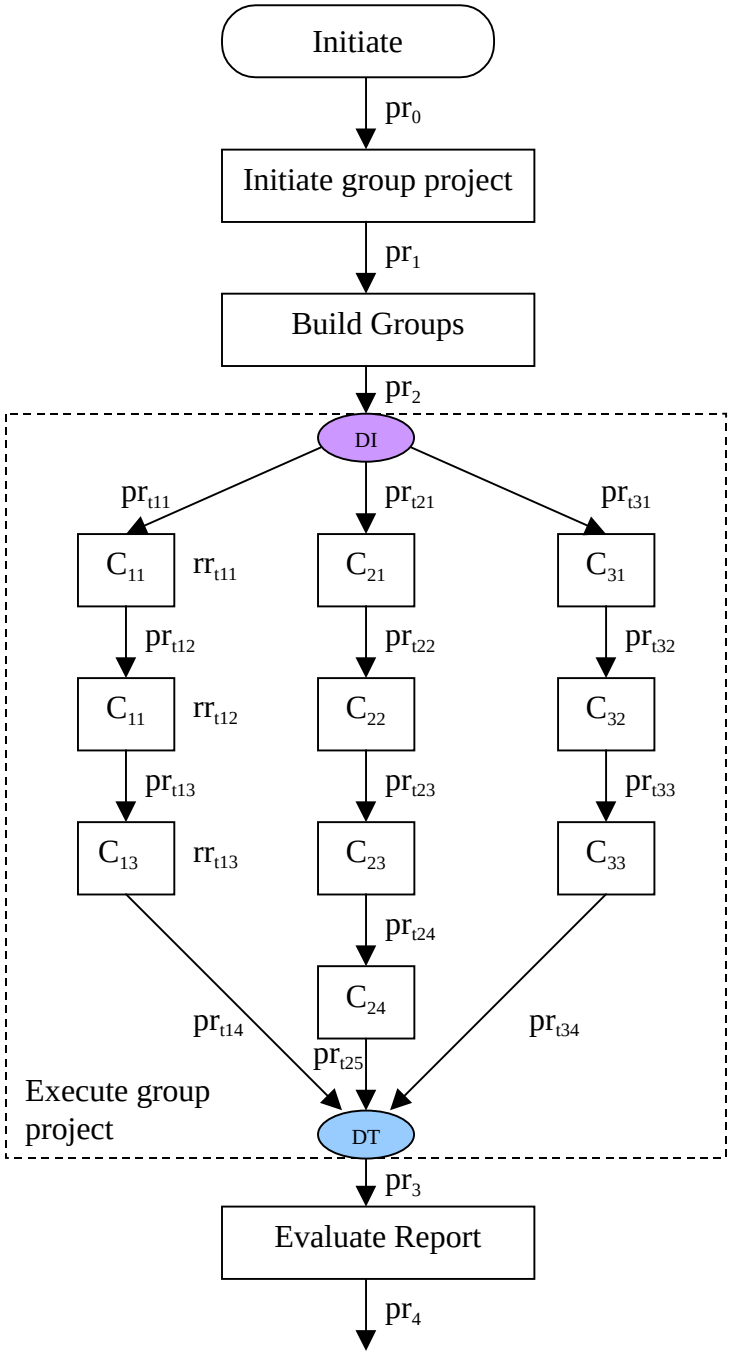
<t_2, “COMPLETED”, “1998/10/5”, “1998/10/6”, “Lee”, TA, t_1, {GPN00001}, write>

<t_3, “INITIATED”, “1998/10/6”, NULL, “Cho”, Group_Manager, t_2, {GPN00001}, read>

<t_4, “INITIATED”, “1998/10/6”, NULL, “Ahn”, Group_Manager, t_2, {GPN00001}, read>

<t_5, “INITIATED”, “1998/10/6”, NULL, “Kim”, Group_Manager, t_2, {GPN00001}, read>

At this time, each group manager should add new tasks to execute the group project according to the guide of Group_Project_Notification_Form. Assume that the group managers dynamically decompose the task “Execute group project” as shown in Figure 5.14. In this case, the history information for the task instances (t_3, t_4, and t_5) are updated as follows;



[Figure 5.14] An example process for group project in a virtual classroom (after dynamic decomposition)

<t_3, “DECOMPOSED”, “1998/10/6”, NULL, “Cho”, Group_Manager, t_2, {GPN00001}, read>

<t_4, “DECOMPOSED”, “1998/10/6”, NULL, “Ahn”, Group_Manager, t_2, {GPN00001}, read>

<t_5, “DECOMPOSED”, “1998/10/6”, NULL, “Kim”, Group_Manager, t_2, {GPN00001}, read>

By the decomposition of the group manager “Cho”, the workflow schema is changed as follows;

$WF_{new} = WF_{org} \cup \{WF_{t1}\}$ where

$WF_{t1} = \{T_{t1}, Rp_{t1}, Rr_{t1}, Rm_{t1}\}$ where

$T_{t1} = \{C_{11}, C_{12}, C_{13}, DI, DT\}$,

$Rp_{t1} = \{pr_{t11}, pr_{t12}, pr_{t13}, pr_{t14}\}$, $Rr = \{ rr_{t11}, rr_{t12}, rr_{t13}\}$,

$Rm_{t1} = \{rm_{t11}, rm_{t12}\}$ where rm_{t11}.SOURCE_RULE=$\varnothing \wedge$

rm_{t12}.SOURCE_RULE=$\varnothing \wedge rm_{t11}$.TARGET_RULE = $\{ pr_{t11}, pr_{t12}, pr_{t13}, pr_{t14}\}$

$\wedge rm_{t12}$.TARGET_RULE = $\{ rr_{t11}, rr_{t12}, rr_{t13}\}$

The tasks DI and DT are dummy tasks that represent initial and terminal task of decomposed task respectively. When a task is decomposed, the state of the dummy task DI is initialized as “COMPLETED”, which results in initialization of successor of DI. The state of the task instance t_3 is changed as “COMPLETED” when the dummy task DT is completed. The history information of the task decomposition is contained in sub history of the task instance. That is, a sub history is 10-tuple,

$H_S = < t_s, t_i, s, d_i, d_c, \alpha, r_\alpha, t_p, D, O_d >$ where t_s is super task instance of t_i.

Chapter 6. Implementation of K-WFMS

6.1 Introduction

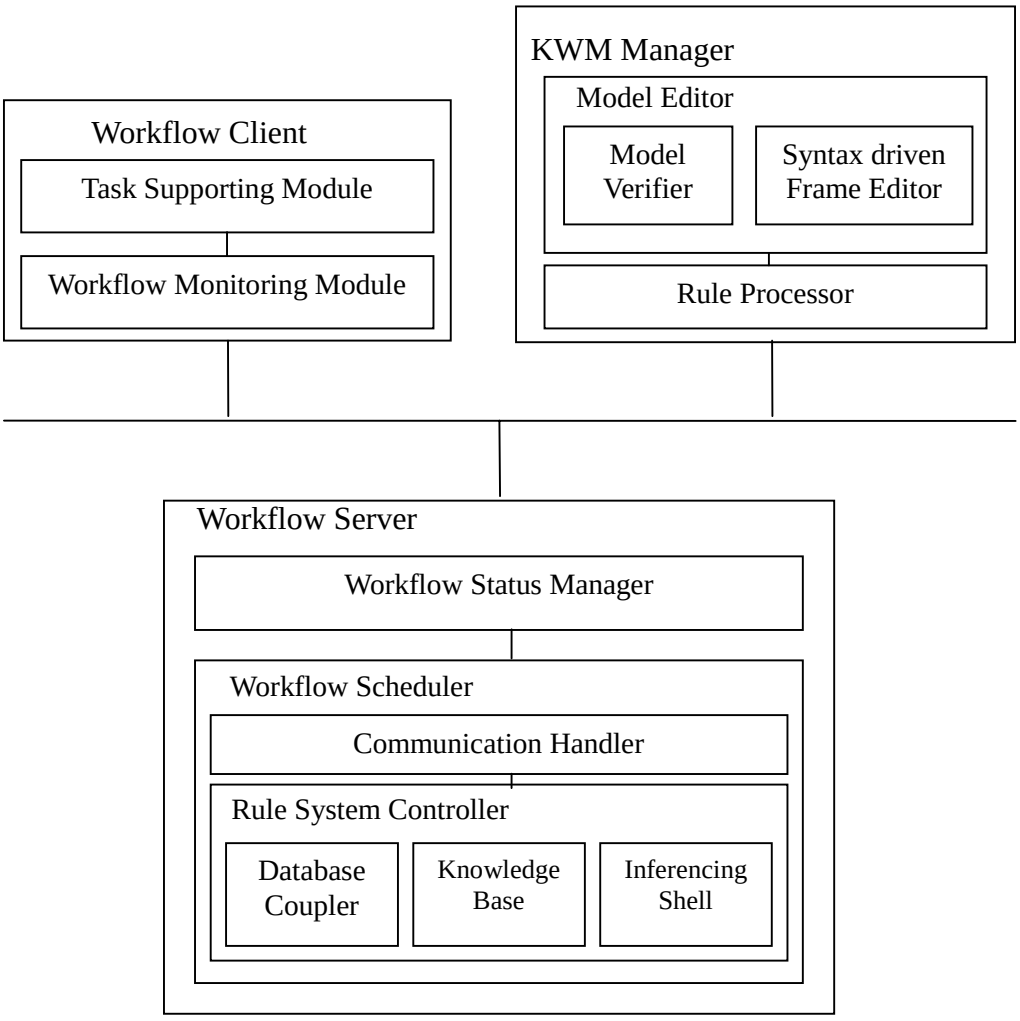
K-WFMS (Knowledge-based Workflow Management System) has been fully implemented using CLIPS and integrated as a component of a campus-wide information system called Intelligent Campus at KAIST (Park et al, 1995 & 1996). K-WFMS is integrated with other application information systems for executing tasks in business processes. In this chapter, overall architecture of K-WFMS and detailed implementation approach is explained. With the successful real application, KWM are proved to be a useful framework to implement the fully automated workflow under the agile environment.

6.2 Overall Architecture

Figure 6.1 illustrates the overall architecture of the K-WFMS. It has client-server architecture. Client part consists of KWM model manager and workflow client, and workflow server is composed of workflow status manager and workflow scheduler. We give detailed descriptions on each part in the following two sub sections.

K-WFMS is implemented under the SOLARIS operating system on a Sun SPARC 10. The workflow server has been implemented with OpenServer library of Sybase Inc., and the Sybase has been used as database server. OpenServer provides C libraries for building a server. Using the libraries, system developers access database using C functions and are able to integrate with other applications encoded with C language. We chose CLIPS (The C Language Integrated Production System) as the inference shell of

the rule system because it is appropriate tool to implement frames and easy to integrate with C language. All the frame constructs are compiled into CLIPS source codes before the inference is performed. The Workflow client has been implemented with PowerBuilder of PowerSoft Inc., and the KWM model manager has done with Visual C++ and CLIPS.



[Figure 6.1] Overall architecture of K-WFMS

6.3 K-WFMS Client

Client part of K-WFMS consists of KWM manager and workflow client. The former is used by workflow designer and enables the designer easy modeling of workflows. The other is used by workflow participants and supports the participants to manage and execute assigned tasks.

6.3.1 KWM Manager

KWM manager consists of model editor and rule processor. Model editor supports model builder's easy modeling of KWM by providing syntax-driven frame editor. Using the syntax-driven frame editor, workflow designer defines frames in KWM. All the frames of a workflow schema are inserted into workflow definition tables in a database. The editor provides metadata of the organizational database. The metadata includes database servers, databases, tables and fields information. When a workflow designer selects objects that should be used to control the target workflow, the frame editor inserts the information into workflow definition tables after translation of the objects into frames. The automatic translation prevents the model builder's syntactic error that might be occurred in the course of manual definition of frames. The model editor also provides query service on the information in KWM. Model builders can obtain the following information on the defined frames:

- List of all frames classified by frame styles
- List of frames that are defined for certain workflow
- All the rules expressed by verbal style
- Dependencies between rules including hierarchical relationships and horizontal relationships in the same level in the hierarchy
- Predecessors and pre-conditions of a task
- Successors and post-conditions of a task
- Documents and resources needed for a task execution

The model verifier verifies designer defined rules using algorithms proposed in chapter 4. Another functionality of model editor is to provide change propagation mechanism. According to the propagation rules in chapter 5, users are guided to modify workflow schema. The rule processor compiles frames in KWM into executable rule scripts (CLIPS program). The source codes that are generated from frames for example workflow in section 3.5 can be found in Appendix D.

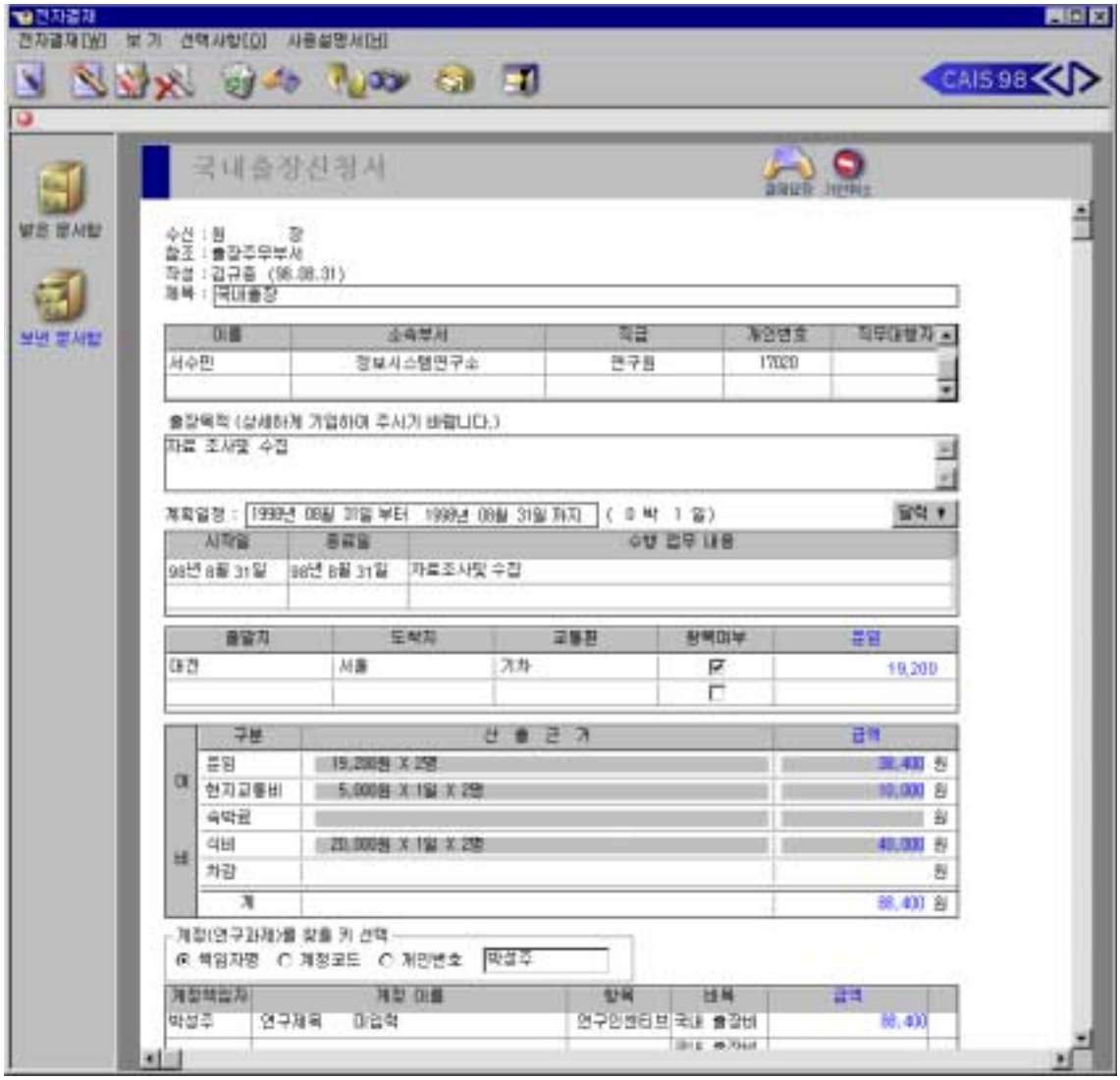
6.3.2 Workflow Client

On the other hand, the workflow client supports an actor's execution of assigned tasks and provides him/her workflow monitoring-service. The task supporting-module lists all the tasks assigned to him/her with information on the role-in-charge and attached forms. When the actor selects a task in order to execute it, the task supporting-module launches an application system automatically if the application system is

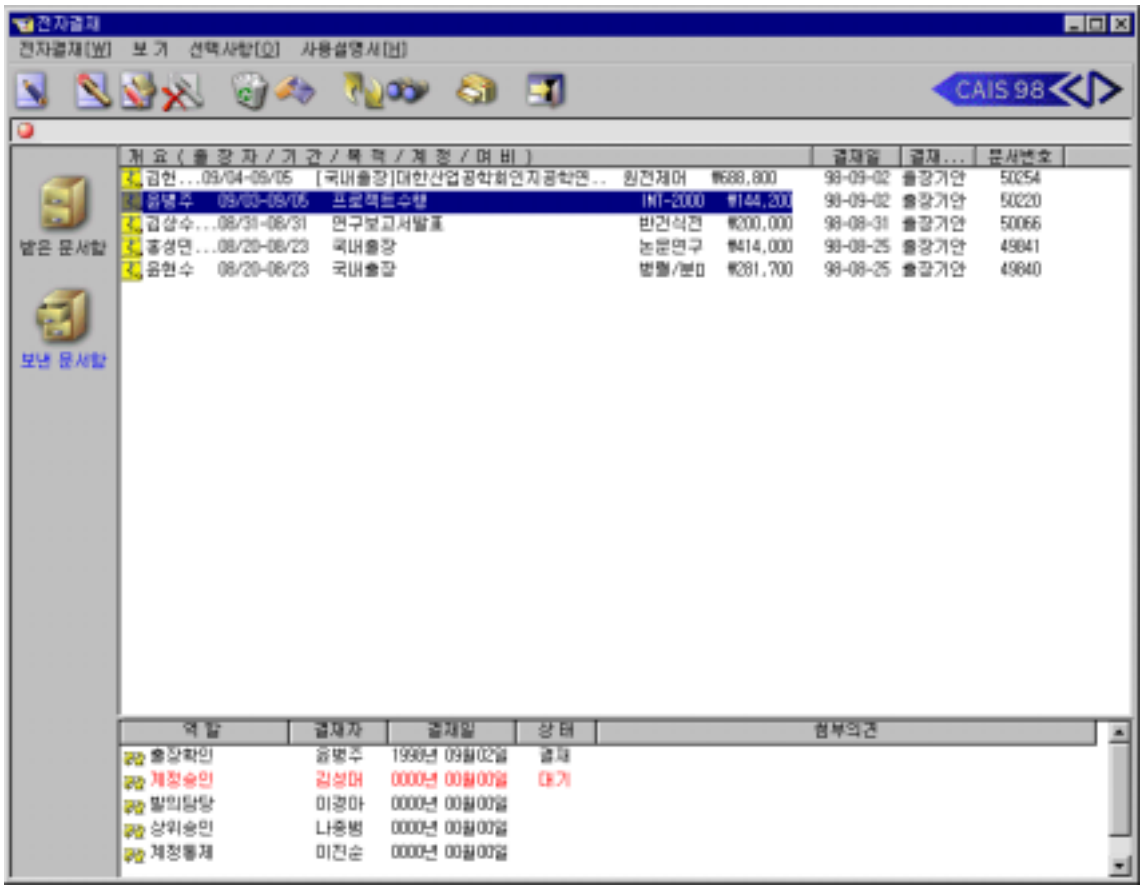
defined as supporting tool of the task execution. Figure 6.2 shows a screen example of form management supporting tool. Using smart form management system, Users could build a form easily without violating business rules that restrict filling-in the field values of the form. When the task is finished, the task supporting-module notifies workflow server the state of the task. The state of a task can be one of “initiated”, “completed”, “pending” and “canceled”. The workflow monitoring-module provides the information concerned with history of workflow instances. The information includes followings:

- Initiator of a workflow instance
- All participants of a workflow instance
- Tasks executed already
- The states of the workflow instance that the actor initiated
- All forms created in a workflow instance

Figure 6.3 shows a screen example of workflow monitoring service. In Figure 6.3, the upper window lists all the workflow instances a workflow participant created. When the participant selects an item, the history information of the workflow instance is listed in the lower window.



[Figure 6.2] Screen example of smart form



[Figure 6.3] Screen example of workflow monitoring service

6.4 Workflow Server

Workflow server is composed of two cooperative environments, Workflow Status Manager (WSM) dedicated to workflow status management and Workflow Scheduler (WSD) to workflow scheduling.

The WSM is responsible for updating workflow control data in database. When an actor finishes a task, the workflow client notifies the workflow server the state of the finished task through remote procedure call (RPC) protocol. The WSM updates the value of “state” field of “task” table in database and requests the WSD via RPC protocol to find next tasks and charged actors. On receiving the results from the WSD, the WSM inserts the results into the workflow instance table.

The WSD finds next tasks and actors by firing rules model builder defined. It has two sub modules, i.e., RPC handler and rule system controller. The RPC handler is responsible for the communication with the WSM. When a request from the WSM is occurred, the RPC handler parses the contents of message from the WSM. The message contains information needed for workflow scheduling, and includes workflow type, workflow instance identifier, and task number which has been finished. The RPC handler passes the rule system controller the message contents. The rule system controller configures rule base and fact base in inferencing shell to determine the next tasks that should be followed after finished task and actors who are in charge of the next tasks. The detailed description on the rule firing mechanism and the functionality of database coupler are given in section 6.4.2 because it would be more explanatory if we describe it by mentioning implementation tools.

6.4.1 Generation of CLIPS codes

All the frames in KWM are compiled into CLIPS codes. The compiled CLIPS codes are stored as files in knowledge base in the workflow server. Every change on a frame is automatically propagated into the CLIPS codes file by re-compiling the changed frame. Entity and relation frames are compiled into ‘deftemplate’ constructs of CLIPS, and rule frames into ‘defrule’ constructs. Figure 6.4 shows a translation example of an entity frame into CLIPS codes.

Ex-F1 :

```
FRAME Department
    IS_A : Entity;
    dept_id : number;
    parent : number;
    mnger_pos : number;
END-FRAME
```

Ex-C1 :

```
(deftemplate Department
                (field dept_id (type NUMBER))
        (field parent (type NUMBER))
        (field mnger_pos (type NUMBER)))
```

[Figure 6.4] Translation of an entity frame into CLIPS codes.

The compiling rule frames into ‘defrule’ constructs is more complicated. The compiler applies different rules for each rule type. For procedural rule type, the compiler uses four slots (“pre-task”, “pre-task-event”, “condition”, and “next-task”) of the rule frame. The “defrule” construct of CLIPS is composed of two parts, i.e., condition part and action part. The compiler uses three slots (“pre-task”, “pre-task-event” and “condition”) to compose condition part and “next-task” slot to compose action part. The compiler uses two slots of the responsibility rule frame to compose mapping rule between roles and real actors. The procedural_rule frame ‘rp8’ in Figure 3.8 and the responsibility_rule frame rr6-1 are translated into following CLIPS codes respectively:

```
(defrule rp8
        (Task (task_id “Approve_Subordinator’s_Trip”)(state "Completed"))
        (WF_DomTripMaster (Duration ?dur))
        (test (>= ?dur 7))
        =>
        (assert (Task (pretask_id “Approve_Subordinator’s_Trip”)
                (task_id “Inspect_Trip_Purpose”)(state "Initiated"))))
```

```
(defrule find-supervisor
    (Task (pretask_id ?pr_task)(task_id “Approve_Subordinator’s_Trip”)(state "completed"))
    (WF_DomTripTraveller (T_Id ?t_id)(T_Department ?dept))
    (Department (dept_id ?dept)(mnger_pos ?m-pos))
    (WorkFor (dept_id ?dept)(actor_id ?supervisor_id)(position ?m-pos))
    =>
    (assert (ActorMapping (pretask_no ?pr_task)(task_id “Approve_Subordinator’s_Trip”)
                                                (actor_id ?supervisor_id))))
```

The compilation of metarules are somewhat complicated. We use “undefrule” command of CLIPS to compile a substituting metarule. At the following CLIPS codes, the condition part represents the situation the metarule applied. In the action part “undefule” removes normal procedural_rule frames from inferencing shell and exceptional procedural_rule frames which are stored in another knowledge base substitute the normal rule frames.

```
(defrule undefrule-for-delegated-employee
        (declare (salience 200))
        (WF_DomTripTraveller (T_Id ?t-id))
        (WorkFor (actor_id ?t-id)(outside "y"))
        =>
        (undefrule from-traveller-to-account)
        (undefrule from-account-to-department)
        (undefrule from-account-to-supervisor)
        (undefrule from-supervisor-to-auditor)
        (undefrule from-supervisor-to-controller)
        (load "./wfdef/exception100-1.clp"))
```

6.4.2 Controlling Rule Execution

In this sub section, we describe the procedure for determining of next tasks and real actors who are in charge of the tasks, which is implemented in K-WFMS. First, The rule system controller in the WSD initializes CLIPS environment and automatically generates following initial facts;

(deffacts (WFinstance (instance_id “WF001”))

(deffacts (Task (task-name “Approve_Subordinator’s_Trip”) (state “approved”))

The fact can be inferred from the message contents received from the RPC handler by identifying the task that has been finished. The state of the task is extracted from the task instance table in database. The rule system controller, then, selects procedural rules and metarules that have to do with for scheduling tasks of the target workflow type and loads them into rule base in the inference shell. The selection of rules is performed by checking “pre-task” slot value of procedural rules. If a rule contains the name of the finished task in the slot, the rule is inserted into rule base. After inserting all the selected procedural rules into rule base, the controller selects all the metarules that contain the selected rules as value of the ‘target-rule’ slot, and again inserts them into rule base. When the rule selection is finished, the database coupler extracts data from database and inserts it into fact base after processing it as the format used in the inference shell. The next section explains the work of database coupler in detail.

When the configuration is completed, the controller fires the rule base. In the course of inferencing, metarules may retract some procedural rules if their condition parts evaluate true. The result of the firing is inserting new facts into fact base, and the new facts take following form;

(Next-Task (task-name “task-name”))

If the inserted fact is “(Next-Task (task-name NIL))”, this means that other task

should be executed before the next tasks are ready. Otherwise, the controller reconfigures the rule base and fact base to find real actors who are in charge of the next tasks using role concept. The controller inserts responsibility rules and metarules into rule base by loading CLIPS files that contain “defrule” constructs. After inserting new facts into fact base from database, the inference shell is executed and produces the following facts;

(Result (next-task “task-name”) (actor “actor-id”))

When all the procedures above finished, the rule system controller notifies the RPC handler the successful execution of inference shell. The RPC handler, then, parses the output file that created as a result of the execution and sends the information to the WSM.

6.4.3 Coupling KBS with DBS

The role of the database coupler in the WSD is to provide the inference shell organizational facts from organizational database. The database coupler parses the rule constructs in rule base, and makes a list of data to be extracted from database using meta-knowledge on the KWM. Then, it extracts data from database and inserts it into fact base of the rule system. This approach is loose coupling of KBS and DBS, that is, the DBS provides the KBS data before rules are fired, and no data is provided to KBS in the middle of KBS execution (Vassiliou, Clifford & Jarke, 1985).

The entity and relation frames in KWM are tightly connected with the tables in the database system. The connection information is kept in the database interface that defines mapping relationships between frames in the KWM and data schema in the database. The table is mapped with an entity or a relation frame, and the fields of the

table are mapped with the attribute slots of the frame.

One approach to couple KBS with DBS is to transfer all records of a table into fact base. The approach, however, has limits to apply in our system because the amount of records in some tables is too many. Currently, the number of students in KAIST exceeds 7,000. Furthermore, the existing database keeps all the data about graduated students, and the size of the table “Student” in database exceeds 20,000 records. The size of the table “Employee” and “Professor” is smaller than “Student” table, but exceeds 7,000 and 1,000 respectively. If we transfer all the data about organizational structure and resources into fact base, the performance of the inference shell might notably decrease.

To overcome the problem, we suggested an approach that filters data from database that is needed to fire rules in rule base. The database coupler makes a list of the rules that exist in the rule base. For each rule, the coupler checks conditions and makes SQL command for each condition using database interface. The database interface is to connect frames and data schema in a database. It contains the one-to-one mapping relationships between frame slots and fields of table in the database.

We explains the procedures of coupling KBS and DBS using the rule “Find_Supervisor” defined in Figure 5B. It has following three conditions in condition part;

(Trip_Request_Form (traveler_info (traveler_id ?t_id)))

(WorkFor (actor_id ?t_id) (department ?dept))

(Department (manager_id ?mng_id) (dept_id ?dept))

It is assumed that the names of the data variables in a rule definition are unique, and each condition constrains slot values of a frame. The conditions are, also, assumed having orders between them. At first, The database coupler checks the variables defined

in each condition. If two conditions have the same variable, the two frames are considered having a relationship, and the attribute that is constrained by the variable is used as foreign key. The database coupler generates SQL command for each condition from the one that has highest order, i.e. the condition that is declared on the top of the condition part. The first condition constraints on the slot value of a composite frame. In the case, the database coupler generates following SQL command;

SELECT form_id FROM WFinstnce_FormInstance WHERE wfinst_id = ?wfinstance

SELECT traveler_id FROM Traveler_Info WHERE form_id = ?form_id

The table ‘WFinstance_FormInstance’ has been defined to contain all the form identifiers that are used in a workflow instance. The database coupler fetches the form identifiers for the given workflow instance, and the result data is used to generate the second SQL command. The database coupler sends the SQL commands to the database management system (DBMS) and gets the results of them. Using the SQL commands, the following CLIPS code is generated, and loaded in CLIPS environment to insert facts into fact base (we assume that there are two travelers for the workflow instance);

```
(deffact (Traveler_Info (traveler_id 935291))
         (Traveler_Info (traveler_id 975211)))
```

Next, the database coupler tries to generate SQL commands for the second condition. In this time, the results (traveler identifiers) from the first SQL command are used to extract department information as the two conditions have the same variable “?t_id”.

SELECT department FROM WorkFor WHERE actor_id = 935291 OR actor_id=975211

If the database coupler receives any results of the SQL command from DBMS, it

inserts the following facts into fact base;

(WorkFor (actor_id 935291) (department 2100))

(WorkFor (actor_id 975211) (department 1500))

Lastly, the database coupler does the same procedure with the second condition for the third condition. The generated SQL command and new fact would be as follows;

SELECT manager_id, dept_id FROM Department

WHERE dept_id = 2100 OR dept_id = 1500

(Department (dept_id 2100) (mng_id 250))

(Department (dept_id 1500) (mng_id 342))

Chapter 7. Conclusion

7.1 Introduction

The emergence of new type of organization such as agile organization has led to the increased needs of adaptable WFMS. WFMS's appeal lies in the execution of business processes as a sequence of pre-planned tasks, so that the cost of coordinating different tasks is reduced. However, most of the existing WFMSs are rigid for supporting changing business processes under the turbulent organizational environments. Furthermore, providing workflow participants the mechanism to dynamically change their workflow instance is rarely addressed. Understanding changes on workflow under organizational context is important to support workflows. This, in turn, requires a

consideration of the organizational context in which workflow occur. These issues served as the nucleus for the work presented in this dissertation.

7.2 Contributions of the Dissertation

The contributions of this dissertation lie in four areas that have not been previously addressed in the literature:

(1) The concept of workflows is redefined so that it can be understood with organizational context. KWM defines a workflow as a set of business rules. Business rules contain rationale of workflow execution. They guide and restrict flow of works through ordered tasks, and assign responsibilities for tasks to actors using role concept. Organizational policies that reflect the organization's special characteristics are predefined as exceptional rules. KWM provides three basic rule frames, i.e. procedural-rule, responsibility-rule, and metarule frames, to represent the three types of business rules. This facilitates the introduction of organizational aspects into the domain of WFMS.

(2) A set theoretic formalism of KWM is developed and soundness properties based on the formal definition of KWM are developed and verified. Formalism is necessary for self-complacency, communication, extension and modification of model, and formal comparisons with other models. A formal property for correctness of KWM, soundness, is defined and analyzed using verification techniques. Knowledge-based approach of

KWM enables to check consistency and compactness of routing rules as well as terminality of workflows.

(3) A change propagation mechanism is proposed to enhance adaptability of WFMS. Changes are classified as schema-level change and instance-level change. The former is concerned with workflow schema evolution and the other is concerned with occurrence of exceptions in a workflow instance. Change propagation rules for schema modification primitives assure the soundness of new version. On the other hand, change propagation rules for modification primitives for instance-level change create temporal rule frames without changing the original workflow schema. The temporal rule frames are added when the target workflow instance progresses. This approach simplifies version management of workflow schema and undoing temporal changes, which increases the adaptability of WFMS.

(4) Based on KWM, K-WFMS is developed. The implementation of K-WFMS integrated two heterogeneous technologies. For effective management of workflow schema, a knowledge based system, CLIPS, is adopted. On the other hand, database system, SYBASE, is used to manage huge volume of workflow instances. Integrating the two technologies is proved to be appropriate for implementing adaptive WFMS.

7.3 Further Research Directions

Further research areas are as follows:

(1) Development of graphic based KWM manager.

The current implementation of KWM manager is based on syntax-driven frame editor. Development of graphic based KWM manager provides workflow designer easy modeling work. The graphic based KWM manager should be able to generate KWM frames from workflow diagram. The workflow verifier and change propagation mechanism proposed in this dissertation can be easily integrated with the graphic based KWM manager.

(2) Development of facility for workflow transaction management

One of main issues for development of database-based WFMS is providing transaction management mechanism which performs consistency and concurrency control or recovery from failure. The results from researches on transactional workflow (Eder and Liebhart, 1995; McCarthy and Sarin, 1993; and Rusinkiewicz and Shet, 1993) can be extended to K-WFMS.

(3) Integration with groupwares for supporting ad-hoc workflows

The current implementation of K-WFMS is applied to application systems for supporting execution of office tasks. The application systems for office tasks are usually used by an actor. On the other hand, many of tasks in an ad-hoc workflow such as software process are usually performed by multiple actors and the use of groupware is needed. Integration of groupwares with WFMS will increase the performance of group works in ad-hoc workflows.

References

Aielo, L., Nardi, D., and Panti, M. (1984). Structural office modeling: A first step toward the office expert system. In *Proceedings 2nd ACM Conference on Office Information Systems (Toronto, June 1984),* ACM New York.

Agostini, A., Michelis, G.D., Grasso, M.A., Prinz, W., and Syri, A. (1996). Contexts, work processes, and workspaces. *Computer Supported Cooperative Work: The Journal of Collaborative Computing,* 5, 223-250.

Appleton, D. S. (1984). Business Rules: The Missing Link", *Datamation*, October, pp.145-150.

Bailey, A., Gerlach, J., McAfee, P., and Whinston, A. (1983). An OIS model for internal accounting control evaluation. *ACM Trans. Office Inf. Syst.* 1 (1), 25-44.

Barber, G. (1983). Supporting organizational problem solving with a work station. *ACM Trans. Officce Inf. Syst.* 1 (1), 45-67.

Bogia, D. P. and Kaplan, S.M. (1995). Flexibility and control for dynamic workflows in the worlds environment. *In Proc. Of the Conference on Organizational Computing Systems* (ACM 1995).

Bose, Ranjit (1996). Intelligent agents framework for development knowledge-based

decision support system for collaborative organizational processes. *Expert System With Applications*, 11(3), 247-261.

Bose, Ranjit (1995). CMS: An intelligent knowledge-based tool for organizational procedure modeling and execution. *Expert System With Applications,* 8 (1), 1-21.

Bracchi, G., and Pernici, B. (1983). SOS: A conceptual model for office information systems. In *Proceedings of ACM SIGMOD Database Week Conference* (San Jose, Calif., May), ACM, New York, 108-116.

Bracchi, G., and Pernici, B. (1984). The design requirements of office systems. *ACM Trans. On Inf. Syst.,* 2 (2), 151-170.

Casati, F., Ceri, S., Pernici, B., and Pozzi, G. (1996). Deriving active rules for workflow enactment, *Proceedings of DEXA 96*, Zurich (CH), 1996.

Casati, F., Ceri, S., Pernici, B., and Pozzi, G. (1998). Workflow evolution. *Data & Knowledge Engineering*, 24, 211-238.

Chang, J.W. and Scott, C.T. (1996). Agent-based workflow: TRP Support environment (TSE). *Computer Networks and ISDN Systems.* 28, 1501-1511.

Cook, C. (1980). Streamlining office procedures – an analysis using the information control net model. In *Proceedings AFIPS National Computer Conference* (May 1980), 555-565.

Curtis, B., Kellner, M. I., and Jim, O. (1992). Process Modeling. *CACM,* 25 (9), 75-90.

Dellen B., Maurer, F., and Pews, G. (1997). Knowledge-based techniques to increase the flexibility of workflow management. *Data & Knowledge Engineering*, 23, 269-295.

Eder, J. and Liebhart, W. (1995). The workflow activity model: WAMO, *Proceedings of the third international conference on the cooperative information systems*, 87 - 98.

Ellis, C., and Nutt, G. (1980). Office information systems and computer science. *ACM Computing Survey.* 12 (1), 27-60.

Ellis, C., and Bernal, M. (1982). OFFICETALK-D: An experimental office information system. In *Proceedings ACM SIGOA Conference on Office Systems* (Philadelphia, June 1982), ACM, New York, 131-140.

Ellis,C., and Nutt, G. (1993). Modeling and enactment of workflow systems, *Application and Theory of Petri Nets 1993, 14th International Conference Proceedings*, Chicago, Illinois, USA, 1-16.

Flores, F., Graves, M., Hartfield, B., and Winograd, T. (1988). Computer systems and the design of organizational interaction. *ACM Transactions on Office Information Systems*, 6 (2), 153-172.

Ganapathy, B. K. (1996). *The representation of organizational workflows as knowledge*

management episodes. Ph.D Dissertation, Department of Decision Sciene and Information Systems, University of Kentucky, Lexington, Kentucky.

Grasso, A., Meunier, J., Pagani, D., and Pareschi, R. (1997). Distributed coordination and workflow on the world wide web, *Computer Supported Cooperative Work: The Journal of Collaborative Computing, 6, 175-200.*

Gruhn, Volker(1995). Business process modeling and workflow management. *International Journal of Cooperative Information Systems*. 4(2&3), 145–164.

Hammer, M., and Kunin, J. (1980). Design principles of an office specification language. In *Proceedings AFIPS National Computer Conference* (May 1980), 541-547.

Halle, B. V. (1993). Business Rules in Practice. *Database Programming & Design*, July, pp. 15-18

Hurts, K. and de Greef, P. (1994). Cognitive ergonomics of multi-agent systems: observations, principles and reseach issues. *Lecture Notes in Computer Science* (B. Blumenthal, J. Gornostaev, and C. Unger, eds), 876: 164-180, Springer-Verlag.

Jarke, M., Jeusfeld, M.A., Peters, ., and Pohl, K. (1997). Coordinating distributed organizational knowledge. *Data & Knowledge Engineering*, 23, 247-268.

Jennings, N.R., et al. (1996). Agent based business process management. *International*

Journal of Cooperative Information System. 5 (2), 105-130.

Kang, H. W. (1995). *A Modeling Framework for Manufacturing System Integration: Process-based Approach.* Ph.D Dissertation, Department of Industrial Mangement, Korea Advanced Institute of Science and Technology, Taejon, South Korea, November, 1995.

Kang, H. W., Kim, J. W., and Park, S. J. (1998). Integrated Modeling Framework for Manufacturing Systems: A Unified Representation of the Physical Process and Information Systems. *International Journal of Flexible Manufacturing System,* 10 (3).

Kappel, G., Lang, P., Rausch-Schott, S., Retschitzegger, W. (1995). Workflow management based on objects, rules, and roles. *IEEE Bulletin of the Technical Committee on Data Engineering*, 18(1).

Kaplan, S., Tolone, W., Bogia, D., and Bignoli, C. (1992). Flexible, active support for collaborative work with ConversatoinBuilder. In *Proceedings of the Conference on Computer Supported Cooperative Work (Toronto, Canada),* ACM/SIGCHI and SIGOIS, NY, 1992, 378-385.

Kim, J. W. (1994). *A modeling framework for integrated decision models and information systems*, Ph.D Dissertation, Department of management science, Korea

Advanced Institute of Science and Technology, Taejon, South Korea, November 1994.

Kirn, S. (1994). Supporting human experts' collaborative work: Modeling organizational context knowledge in cooperative information system. In *CSCW and Artificial Intelligence*, edited by Connolly, John H. and Edmonds, Ernest A. Springer-Verlag. 127-139.

Konsynski, B., Bracker, L., and Bracker, W. (1982). A model for specification of office communications. *IEEE Trans. Commun. COM-30,* 1, 27-36.

Loucopoulos, P. and Layzell, W. (1989). Improving information system development and evolution using a rule-based paradigm. *Software Engineering Journal,* September, pp. 259-267.

Lubich, H. P. (1995). *Towards a CSCW Framework for Scientific Cooperation in Europe.* Lecture Notes in Computer Science, 889, Spring-Verlag, Berlin.

Lum, V., Choy, D., and Shu, N. (1982). OPAS: An office procedure automation system. *IBM Syst. J.*, 21 (3), 327-350.

Mahling, D.E., Woo, C., Bluementhal, R., Sclichter, H., and Horstman, T. (1995), Workflow = OIS? A report of a workshop at the CSCW '94 Conference. *SIGOIS Bulletin, 16 (1),* 59-64.

Marshak,R.T. (1993). Action technologies' workflow products. *Workgroup Computing*

Report, 16 (5).

Martin J. (1993) *Principles of Object-Oriented Analysis and Design*, Prentice-Hall.

McCarthy, D. R. and Sarin, S. K. (1993). Workflow and Transactions in InConcert. *Bulletin of the Technical Committee on Data Engineering*, 16 (2).

Medina-Mora, R., Winograd, T., Flores, R., and Flores, F. (1992). The action workflow approach to workflow management technology. *Proceedings of Computer Supportive Cooperative Work '92*, Toronto, Canada, 281-288.

Michael, J. B., Sibley, E. H., Baum, R. F., and Li, F. (1992). On the Axiomatizaion of Security Policy: Some Tentative Observations about Logic Representation. *Proceedings of the sixth IFIP TC 11.3 Working Conference on Database Security*, pp. 401-429.

Michelis, G. D. and Grasso, M. A. (1994). Situating conversations within the language/action perspective: The milan conversation model. *In Proceedings of Computer Supported Cooperative Work '94*, Chapel Hill, NC, USA., 89 – 100.

Moriarty, T. (1993a). The Next Paradigm, *Database Programming & Design*, February, pp. 66-69.

Moriarty, T. (1993b). Business Rule Analysis. *Database Programming & Design*, April, pp. 66-69.

Nguyen, T.A., Perkins, W.A., Laffey, T.J., and Pecora, D. (1985). Checking an expert systems knowledge base for consistency and completeness. *In Proc. 9th International Joint Conference on Artificial Intelligence (IJCAI 85)* , 1, 375-378.

Nilsson, N. J. (1980). *Principles of artificial intelligence.* Springer-Verlag.

Odell, J. J. (1993a). Specifying requirements using rules. *Journal of object-oriented programming,* May, pp. 20-24.

Odell, J. J. (1993b). Using business rules with diagrams. *Journal of object-oriented programming,* July/August, pp. 10-16.

Ong, K. and Lee, R. M. (1996). A decision support system for bureaucratic policy administration: An abductive logic programming approach. *Decision Support System,* 16, 21-38.

Park, S. J., Kim, J. W., and Kang, H. W. (1996). Heuristic Knowledge Representation of Production Scheduling: An Integrated Modeling Approach, *Expert System with Applications,* 10 (3/4), pp. 325 – 339.

Park, J. Y. and Park, S. J. (1997). A Process Modeling Framework for Change Management in Business Re-engineering. *Working paper*, Korea Advanced Institute of Science and Technology.

Peterson, J. L. (1981). *Petri Net Theory and the Modeling of Systems*. Prentice-Hall Inc.

Puuronen, S. (1987). A tabular rule-checking method. *In Proc. 7th International Workshop on Expert Systems and their Applications* (Avignon, May 13-15, 1987), 257-268.

Ramaswamy, M., Sarkar, S., and Chen, Y.S. (1997). Using directed hypergraphs to verify rule-based expert systems. *IEEE Trans. On Knowledge and Data Engineering*, 9 (2), 221-237.

Richter, G. (1981). IML – inscribed nets for modeling text processing and data(base) management systems. In *Proceedings Very Large Data Base Conference* (Cannes, Sept. 1981), 363-375.

Rusinkiewicz, M. and Shet, A. (1993). On transactional workflows. *Bulletin of the Technical Committee on Data Engineering,* 16 (2).

Sachs, P. (1995). Transforming work: collaboration, learning, and design. *Communication of the ACM,* 38 (9), 36-44.

Scott-Morton, M. S. (1994). The 1990s research program: Implications for management and the emerging organization. *Decision Support Systems,* 12 (2), 251-256.

Searle, J.R. (1969). *Speech Acts : An essay in the philosophy of language.* Cambridge University Press, Cambridge, UK.

Sibley, E. H., Michael, J. B., and Wexelblat, R. L. (1992a). An approach to formalizing

policy management. In *Economics & Cognitive Science,* P. Bourgine and B. Walliser, Ed. Pergamon Press, 155-169.

Sibley, E. H., Michael, J. B., and Wexelblat, R. L. (1992b). Use of Experimental Policy Workbench: Description and Preliminary Results. In *Database Security, V: Status and Prospects,* C.E. Landwehr and S. Jajodia, eds. Elsevier Science Publishers, pp. 47-76.

Sirbu, M., Schoichet, S., Kunin, J., and Hammer, M. (1981). OAM: An office analysing methodology. MIT Office Automation Group Memo OAM-016.

Suwa, M., Scott, A.C., and Shortliffe, E.H. (1982). An approach to verifying completeness and consistency in a rule-based expert system. *AI Magazine,* 3(4), 16-21.

Ter Hofstede, A.H.M., Orlowska, M.E., and Rajapakse, J. (1998). Verification problems in conceptual workflow specifications. *Data & Knowledge Engineering,* 24, 239-256.

Tsichritzis, D. (1982). Form management. *Commun. ACM,* 25 (7), 453-478.

Van der Aalst, W.M.P. (1997), Verification of Workflow Nets . In P. Azema and G. Balbo , editors, *Application and Theory of Petri Nets 1997*, volume 1248 of Lecture Notes in Computer Science, pages 407--426. Springer-Verlag, Berlin, 1997.

Van der Aalst, K.W.M. (1998). The application of Petri nets to workflow management.

The Journal of Circuits, Systems and Computers, forthcoming.

Van der Aalst, K.W.M., van Hee, K.M., and Houben, G.J. (1994). Modeling workflow management systems with high-level Petri nets. *Proceedings of the second Workshop on Computer-Supported Cooperative Work, Petri Nets and Related Formalisms,* 31-50.

Vernadat, F.B. (1996). Enterprise integration: On business process and enterprise activity modeling. *Concurrent Engineering*, 4 (3), 219 - 228.

Victor, F. and Sommer, E. (1991), Supporting the design of office procedures in the DOMINO system. In *Studies in Computer Supported Cooperative Work,* Bowers and Benford (eds.), New York: Elsevier Science, 1991, 119-131.

WfMC (1996). *Workflow management coalition terminology and glossary (WfMC-TC-1011).* Technical Report, Workflow Management Coalition, Brussels, 1996.

Winograd, T. (1987). A language/action perspective on the design of cooperative work. *Human-Computer Interaction*, 3 (1), 3-30.

Winograd, T. and Flores, F. (1986). *Understanding computers and cognition: A new foundation for design.* Ablex, Norwood, N.J., 1986 and Addison-Wesley, Reading, Mass., 1987.

Wolf, M. and Reimer, U. (1996). *Proceedings of the International Conference on*

Practical Aspects of Knowledge Management (PAKM '96), Workshop on Adaptive Workflow, Basel, Switzerland, Oct 1996.

Zisman, M. (1978). Use of production systems for modeling asynchronous, concurrent processes. In *Pattern Directed Inference Systems,* Waterman and Hayes-Roth, Eds., Academic Press, New York, 1978, 53-68.

Zloof, M. (1982). Office-by-Example: A business language that unifies data, word processing, and electronic mail. *IBM Syst. J.*, 21 (3), 272-304.

Appendix A. Specification of rule frames for example workflow

A.1. Procedural-rule frames

```
FRAME rp0
    PRE_TASK: Initiate;
    PRE_TASK_STATE: ‘Completed’;
    NEXT_TASK: Create_Trip_Request_Form;
    CONDITION : NULL;
END-FRAME
```

```
FRAME rp1
    PRE_TASK: Create_Trip_Request_Form;
    PRE_TASK_STATE: ‘Completed’;
    NEXT_TASK: Confirm_Travel;
    CONDITION : NULL;
END-FRAME
```

```
FRAME rp2-1
    PRE_TASK: Confirm_Travel;
    PRE_TASK_STATE: ‘Completed’;
    NEXT_TASK: Confirm_Task_Delegation;
    CONDITION : (Traveler (T_Id ?t-id))
          (Professor (P_Id ?t-id) (Position ?pos))
          (test (> ?pos 0));
END-FRAME
```

```
FRAME rp2-2
    PRE_TASK: Confirm_Travel;
    PRE_TASK_STATE: ‘Completed’
    NEXT_TASK: Confirm_Task_Delegation;
    CONDITION: (Traveler (T_Id ?t-id))
                    (Employee (E_Id ?t-id));
END-FRAME
```

```
FRAME rp3-1
    PRE_TASK: Confirm_Travel;
    PRE_TASK_STATE: ‘Completed’;
    NEXT_TASK: Approve_Traveling_Allowance;
    CONDITION : (Traveler (T_Id ?t-id))
            (Student (S_Id ?t-id));
END-FRAME
```

```
FRAME rp3-2
    PRE_TASK: Confirm_Travel;
    PRE_TASK_STATE: ‘Completed’
    NEXT_TASK: Approve_Traveling_Allowance;
    CONDITION: (Traveler (T_Id ?t-id))
            (Professor (P_Id ?t-id)(Position ?pos))
            (test (=    ?pos    0));
END-FRAME
```

```
FRAME rp4
    PRE_TASK: Confirm_Task_Delegation;
    PRE_TASK_STATE: ‘Completed’;
    NEXT_TASK: Approve_Traveling_Allowance;
    CONDITION : NULL;
END-FRAME
```

```
FRAME rp5
    PRE_TASK: Approve_ Traveling_Allowance;
    PRE_TASK_STATE: ‘Completed’
    NEXT_TASK: Check_Trip_Request_Form;
    CONDITION: (Traveler (T_Id ?t-id))
                (Student (S_Id ?t-id));
END-FRAME
```

```
FRAME rp6
    PRE_TASK: Approve_ Traveling_Allowance;
    PRE_TASK_STATE: ‘Completed’;
    NEXT_TASK: Approve_Subordinator’s_Trip;
    CONDITION : (Traveler (T_Id ?t-id))
                (not (Student (S_Id ?t-id));
END-FRAME
```

```
FRAME rp7
    PRE_TASK: Check_Trip_Request_Form;
    PRE_TASK_STATE: ‘Completed’
    NEXT_TASK: Approve_Subordinator’s_Trip;
    CONDITION: NULL;
END-FRAME
```

```
FRAME rp8
    PRE_TASK: Approve_Subordinator’s_Trip;
    PRE_TASK_STATE: ‘Completed’;
    NEXT_TASK: Inspect_Trip_Purpose;
    CONDITION : (Trip_Request_Form (Duration ?dur))
                (test (> ?dur 7));
END-FRAME
```

```
FRAME rp9
    PRE_TASK: Approve_Subordinator’s_Trip;
    PRE_TASK_STATE: ‘Completed’
    NEXT_TASK: Grant_Traveling_Allowance;
    CONDITION: (Trip_Request_Form (Duration ?dur))
               (test (<= ?dur 7));
END-FRAME
```

```
FRAME rp10
    PRE_TASK: Inspect_Trip_Purpose;
    PRE_TASK_STATE: ‘Completed’;
    NEXT_TASK: Grant_Traveling_Allowance;
    CONDITION : NULL;
END-FRAME
```

```
FRAME rp11
    PRE_TASK: Grant_Traveling_Allowance;
    PRE_TASK_STATE: ‘Completed’
    NEXT_TASK: Update_Traveler_Ledger;
    CONDITION: NULL;
END-FRAME
```

```
FRAME rp12
    PRE_TASK: Update_Traveler_Ledger;
    PRE_TASK_STATE: ‘Completed’;
    NEXT_TASK: Terminate;
    CONDITION : NULL;
END-FRAME
```

```
FRAME rp-s1
    DESCRIPTION: “exceptional rule for
                 delegated employee”;
    PRE_TASK: Confirm_Travel;
    PRE_TASK_STATE: ‘Completed’;
    NEXT_TASK: Confirm_Task_Delegation;
    CONDITION : NULL;
```

```
FRAME rp-s2
    DESCRIPTION: “exceptional rule for
                 delegated employee”;
    PRE_TASK: Confirm_Task_Delegation;
    PRE_TASK_STATE: ‘Completed’;
    NEXT_TASK: Approve_Subordinator’s_Trip;
    CONDITION : NULL;
```

```
END-FRAME
FRAME rp-s3
    DESCRIPTION: “exceptional rule for
                        delegated employee”;
    PRE_TASK: Approve_Subordinator’s_Trip;
    PRE_TASK_STATE: ‘Completed’;
    NEXT_TASK: Approve_Traveling_Allowance;
    CONDITION : NULL;
END-FRAME

FRAME rp-s5
    DESCRIPTION: “exceptional rule for
                        delegated employee”;
    PRE_TASK: Approve_Traveling_Allowance;
    PRE_TASK_STATE: ‘Completed’;
    NEXT_TASK: Grant_Traveling_Allowance;
    CONDITION : (Trip_Request_Form (Duration ?dur)) (test (<= ?dur 7));
END-FRAME
```

```
END-FRAME
FRAME rp-s4
    DESCRIPTION: “exceptional rule for
                        delegated employee”;
    PRE_TASK: Approve_Traveling_Allowance;
    PRE_TASK_STATE: ‘Completed’;
    NEXT_TASK: Inspect_Trip_Purpose;
    CONDITION : (Trip_Request_Form (Duration ?dur))
                    (test (> ?dur 7));
END-FRAME
```

A.2. Responsibility-rule frames

```
FRAME rr1
    ROLE: Form_Creator;
    ACTOR: Trip_Request_Form.Form_Builder;
    CONDITION: NULL;
END-FRAME

FRAME rr3
    ROLE: Task_Mandatory;
    ACTOR: Traveler.Mandatory;
    CONDITION: NULL;
END-FRAME

FRAME rr5
    ROLE: Department_Officer;
    ACTOR: Task_Charge.Actor;
```

```
FRAME rr2
    ROLE: Trip_Applicant;
    ACTOR: Traveler.Traveler_Id;
    CONDITION: NULL;
END-FRAME

FRAME rr4
    ROLE: Account_Manager;
    ACTOR: Account.Manager_Id;
    CONDITION: (Trip_Request_Form (Account ?a-id))
            (Account (A_Id ?a-id) (Manager_Id ?m-id));
END-FRAME
```

CONDITION: (Traveler (Department ?dept-id)) (Task_Charge (Task_Id) (Dept_Id ?dept-id) (Actor ?a-id));
END-FRAME

FRAME rr6-1
ROLE: Supervisor;
ACTOR: WorkFor.actor_id;
CONDITION: (Traveler (Department ?dept-id)) (Department (dept_id ?dept) (mnger_pos ?m-pos))
(WorkFor (dept_id ?dept)(actor_id ?supervisor_id)(position ?m-pos));
END-FRAME

FRAME rr6-2
ROLE: Supervisor;
ACTOR: WorkFor.actor_id;
CONDITION: (Traveler (Department ?dept-id)) (Department (dept_id ?dept) (super-dept ?s-dept))
(Department (dept_id ?s-dept) (mnger-pos ?m-pos))
(WorkFor (dept_id ?dept)(actor_id ?supervisor_id)(position ?m-pos));
END-FRAME

FRAME rr7
ROLE: Auditor;
ACTOR: Task_Charge.Actor;
CONDITION: (Task_Charge (Task ‘Inspect_Trip_Purpose’) (Actor ?a-id));
END-FRAME

FRAME rr8-1
ROLE: Account_Controller;
ACTOR: Task_Charge.Actor;
CONDITION: (Traveler (T_Id ?t-id) (T_Department ?dept)) (Student (S_Id ?t-id))
(Task_Charge (Task ‘Grant_Traveling_Allowance’) (Department ‘Academic_&_Student_Service’) (Actor ?a-id));
END-FRAME

FRAME rr8-2
ROLE: Account_Controller;
ACTOR: Task_Charge.Actor;
CONDITION: (Traveler (T_Id ?t-id) (T_Department ?dept)) (Professor (P_Id ?t-id) (Position NIL))
(Task_Charge (Task ‘Grant_Traveling_Allowance’) (Department ‘Academic_&_Student_Service’) (Actor ?a-id));
END-FRAME

```
FRAME rr8-3
    ROLE: Account_Controller;
    ACTOR: Task_Charge.Actor;
    CONDITION: (Traveler (T_Id ?t-id) (T_Department ?dept)) (Trip_Request_Form (Account ?acc))
      (not (Professor (P_Id ?t-id) (Position NIL))) (not (Student (S_Id ?t-id)))
      (Account (Account_Id ?acc) (Type 'project'))
  (Task_Charge (Task 'Grant_Traveling_Allowance') (Department 'research_management') (Actor ?a-id));
END-FRAME

FRAME rr8-4
    ROLE: Account_Controller;
    ACTOR: Task_Charge.Actor;
    CONDITION: (Traveler (T_Id ?t-id) (T_Department ?dept)) (Trip_Request_Form (Account ?acc))
      (not (Professor (P_Id ?t-id) (Position NIL))) (not (Student (S_Id ?t-id)))
      (Account (Account_Id ?acc) (Type 'general'))
  (Task_Charge (Task 'Grant_Traveling_Allowance') (Department 'finance') (Actor ?a-id));
END-FRAME

FRAME rr9
    ROLE: Personnel_Department;
    ACTOR: Task_Charge.Actor;
    CONDITION: (Task_Charge (Task 'Update_Traveler_Ledge') (Department 'personnel') (Actor ?a-id))
END-FRAME

FRAME rr-s1
    ROLE: Vice_President;
    ACTOR: WorkFor.Actor;
    CONDITION: (WorkFor (Actor ?a-id) (Position 'Vice_President'))
END-FRAME
```

A.3. Metarule frames

```
FRAME rm1
    SOURCE_RULE: {rp2-1, rp3-1};
    TARGET_RULE: {rp2-1};
    CONDITION: (rp2-1 (CONDITION TRUE))
```

```
FRAME rm2
    SOURCE_RULE: {rp2,rp3,rp4,rp5,rp6,rp7,rp8,rp9};
    TARGET_RULE: {rp-s1, rp-s2, rp-s3, rp-s4, rp-s5};
    CONDITION: (Traveler (T_Id ?t-id))
```

(rp3-1 (CONDITION TRUE)); (WorkFor (Actor_Id ?t-id) (Delegated ‘Yes’));
END-FRAME END-FRAME

FRAME rm3
SOURCE_RULE: {rr6-1, rr6-2};
TARGET_RULE: {rr6-2};
CONDITION:
(Traveler (T_id ?t-id) (Department ?dept-id)) (Department (dept_id ?dept) (mnger_pos ?m-pos))
(WorkFor (Actor_Id ?a-id) (Dept_Id ?dept) (Position ?m-pos)) (test (= ?a-id ?t-id));
END-FRAME

FRAME rm4
SOURCE_RULE: {rr6-2};
TARGET_RULE: {rr-s1};
CONDITION:
(Traveler (T_id ?t-id)
(WorkFor (Actor_Id ?t-id) (Position ‘Manager-of-research-institute-in-affiliation’));
END-FRAME

Appendix B. Algorithms for checking soundness properties

B.1. Occurrence Of Circularity

Given a set of rule frames, Ru={Rp, Rr},

1. Set $T_1 = T_2 = \varnothing$,

add tasks ts to T_2 satisfied that

$\exists$ pr $\in$ Rp such that pr.PRE_TASK = INITIATE and pr.NEXT_TASK = t.

2. While (TERMINATE $\notin$ T) and (T_1 != T_2)

Set $T_1 = T_2$

for each t $\in$ T_1,

for each t' satisfied that $\exists$ pr $\in$ Rp such that pr.PRE_TASK = t and pr.NEXT_TASK = t',

if pr.CONDITION = NULL,

print "There is a cycle which is started from t' to t'.

else add task t' to T_2.

B.2. Missing Rules

Given a set of rule frames, Ru={Rp, Rr},

1. Set $T_1 = T_2 = \varnothing$,

add tasks ts to T_2 satisfied that

$\exists$ pr $\in$ Rp such that pr.PRE_TASK = INITIATE and pr.NEXT_TASK = t.

2. While (TERMINATE $\notin$ T) and T_1 != T_2

set $T_1 = T_2$

for each t $\in$ T_1,

if !$\exists$ t' satisfied that $\exists$ pr $\in$ Rp such that pr.PRE_TASK = t and pr.NEXT_TASK = t',

print "There is missing rule to proceed workflow from task t"

add all t' to T_2 satisfied that $\exists$ pr $\in$ Rp such that pr.PRE_TASK = t and pr.NEXT_TASK = t'

B.3. Missing Values

Given a set of rule frames, Ru={Rp, Rr},

for each task t $\in$ T,

Rp(t) = {pr $\in$ Rp | pr.PRE_TASK = t}

let O(t) = { o $\in$ O | o is restricted in pr.CONDITION and pr $\in$ Rp(t)}

for each o $\in$ O(t),

$\vee_{for\ all\ pr \in Rp(t)}$ *pr.CONDITION|o != dom(o)*

where pr.CONDITION|o is a projected condition of Pr.CONDITION,

which is restricted as a condition of object o.

B.4. COMPLETENESS

Given a set of rule frames, Ru={Rp, Rr},

For each t ∈ T, !∃ pr∈ Rp s.t. t = pr.PRE_TASK

B.5. COMPACTNESS

Given a set of rule frames, Ru={Rp, Rr},

For each pair t, t' ∈ T satisfied that ∃ pr ∈ Rp s.t. t = pr.PRE_TASK and t' = pr.NEXT_TASK

set Rp(t,t') = {pr| t = pr.PRE_TASK and t'=pr.NEXT_TASK}

if ∃ Pr1, Pr2 in Rp(t,t') s.t. pr1.CONDITION∧pr2.CONDITION = pr1.CONDITION

print "procedural rule pr2 is duplicated with procedural rule pr1"

B.6. STABILITY

Given a set of rule frames, Ru={Rp, Rr},

For each t ∈ T,

Set Rp(t) = {pr|pr.PRE_TASK = t}

For each pair pr1, pr2 in Rp(t),

If pr1.CONDITION ∧ pr2.CONDITION = NULL,

Print "The two procedural rule pr1 and pr2 may infer conflicting hypothese" (?)

Appendix C. Algorithms that generate dependencies between frames in KWM.

C.1. DEPENDENT

If IS_A(o1,o2), then Add Dependent(o1,o2).

If SUBPART_OF(o1,o2), then Add Dependent(o1,o2).

If user_defined_rel(o1,o2), then Add Dependent(o1,o2) and Dependent(o2,o1).

C.2. XOR-FIRING

Given a set of rule frmes, Ru = {Rp, Rr},

set Rp(t1) = {pr∈Rp| pr.PRE_TASK = t1}

for each pr1∈Rp(t1),

set O(pr1) = {o | o is an object whose domain is restricted in pr1.CONDITION}

set Rp(pr1) = {pr | pr∈Rp(t1), O(pr)=O(pr1), where O(pr) is defined as similar with O(pr1)}

set XOR(pr1) = {pr1}

for each pr∈Rp(pr1),

$pr \wedge (\wedge_{pri \in XOR(pr1)} pri.CONDITION) = \varnothing$

pr∈*XOR(pr1)*

if $(\vee_{pri \in XOR(pr1)} pr_i.CONDITION = X_{oi \in O(pr1)} dom(O_i))$

exit

if $\vee_{pri \in XOR(pr1)} pr_i.CONDITION = X_{oi \in O(pr1)} dom(O_i)$

Add predicate XOR-firing(pr1, pr2, …, prn) for all pr1, pr2, … , prn ∈ XOR(pr1)

C.3. AND-FIRING

Given a set of rule frmes, Ru = {Rp, Rr},

for each t ∈ T satisfied that ∃ pr ∈ Rp s.t. pr.PRE_TASK = t

set Rp(t) = {pr∈Rp| pr.PRE_TASK = t }

for each pair pr1 and pr2 in Rp(t1),

if pr1.CONDITION = pr2.CONDITION,

add AND-firing(pr1,pr2)

C.4. PRECEDENCE

For each pr∈Rp,

if pr.PRE_TASK = t1, pr.NEXT_TASK = t2, add Precedence(t1,t2,pr)

C.5. ROLE-CHARGE

For each rr ∈ Rr,
if rr.ROLE = ro, add Role-charge(ro,rr)

Appendix D. CLIPS source codes for frames in business trip approval workflow.

D.1 Entity & Relationship Frames

```
(deftemplate WF_DomTripMaster
        (field Form_No (type NUMBER))
        (field Writer_Id (type NUMBER))
        (field Writer_Name (type STRING))
        (field Theme (type STRING))
        (field Submit_Date (type STRING))
        (field Start_Date (type STRING))
        (field End_Date (type STRING))
        (field Duration (type NUMBER))
        (field Purpose (type STRING))
        (field Status (type NUMBER))
        (field Total_Cost (type NUMBER))
        (field Control_No (type NUMBER))
        (field Comment (type STRING)))

(deftemplate WF_DomTripTraveller
        (field Form_No (type NUMBER))
        (field T_Id (type NUMBER))
        (field T_Name (type STRING))
        (field T_Type (type NUMBER))
        (field T_Position (type NUMBER))
        (field T_Department (type NUMBER))
        (field T_Class (type NUMBER))
        (field A_Id (type NUMBER))
        (field A_Name (type STRING)))
```

```
(deftemplate WF_DomTripAccount
        (field Form_No (type NUMBER))
        (field Account_No (type STRING))
        (field Account_Name (type STRING))
        (field Account_Type (type NUMBER))
        (field Account_Own_Id (type NUMBER))
        (field Account_Own_Name (type STRING))
        (field Item_Id (type STRING))
        (field Total_Cost (type NUMBER))
        (field Control_No (type NUMBER)))

(deftemplate Department
        (field dept_id (type NUMBER))
        (field parent (type NUMBER))
        (field mnger_pos (type NUMBER)))

(deftemplate w_dept_gyohak
        (field dept_id (type NUMBER))
        (field gyohak_id (type NUMBER)))

(deftemplate w_acti_charge
        (field task_no (type NUMBER))
        (field dept_id (type NUMBER))
        (field worker_id (type NUMBER)))

(deftemplate WorkFor
        (field actor_id (type NUMBER))
        (field dept_id (type NUMBER))
        (field position (type NUMBER)))

(deftemplate Task
        (field pretask_no (type NUMBER))
        (field task_no (type NUMBER))
```

```
        (field state (type STRING)))

(deftemplate ActorMapping
        (field pretask_no (type NUMBER))
        (field task_no (type NUMBER))
        (field actor_id (type NUMBER)))
```

D.2 Source Codes for Procedural-Rule Frames

```
;;
;; exception handling for delegator's travel
;;
(defrule undefrule-for-delegated-employee
        (declare (salience 200))
        (WF_DomTripTraveller (T_Id ?t-id))
        (WorkFor (actor_id ?t-id)(outside "y"))
        =>
        (undefrule from-traveller-to-account)
        (undefrule from-account-to-department)
        (undefrule from-account-to-supervisor)
        (undefrule from-supervisor-to-auditor)
        (undefrule from-supervisor-to-controller)
        (load "./wfdef/exception100-1.clp"))
;;
;; Routing rules for normal workflow instances
;;
(defrule from-kian-to-traveller
        (Task (task_no 1)(state "completed"))
        (WF_DomTripTraveller (A_Id ?agent))
        (test (= ?agent 0))
        =>
        (assert (Task (pretask_no 1)(task_no 3)(state "completed"))))
(defrule from-kian-to-agent
        (declare (salience 100))
        (Task (task_no 1)(state "completed"))
        (WF_DomTripTraveller (A_Id ?agent))
        (test (<> ?agent 0))
        =>
        (undefrule from-kian-to-traveller)
        (assert (Task (pretask_no 1)(task_no 2)(state "completed"))))
```

```
(defrule from-agent-to-traveller
        (Task (task_no 2)(state "completed"))
        =>
        (assert (Task (pretask_no 2)(task_no 3)(state "completed"))))
(defrule from-traveller-to-account
        (Task (task_no 3)(state "completed"))
        =>
        (assert (Task (pretask_no 3)(task_no 4)(state "completed"))))
(defrule from-account-to-department
        (Task (task_no 4)(state "completed"))
       (WF_DomTripTraveller (T_Department ?d_id))
        (w_acti_charge (task_no 5)(dept_id ?d_id))
        =>
        (assert (Task (pretask_no 4)(task_no 5)(state "completed"))))
(defrule from-account-to-supervisor
        (Task (task_no 4)(state "completed"))
       (WF_DomTripTraveller (T_Department ?d_id))
        (not (w_acti_charge (task_no 5)(dept_id ?d_id)))
        =>
        (assert (Task (pretask_no 4)(task_no 6)(state "completed"))))
(defrule from-department-to-supervisor
        (Task (task_no 5)(state "completed"))
        =>
        (assert (Task (pretask_no 5)(task_no 6)(state "completed"))))
(defrule from-supervisor-to-auditor
        (Task (task_no 6)(state "completed"))
        (WF_DomTripMaster (Duration ?dur))
        (test (>= ?dur 7))
        =>
        (assert (Task (pretask_no 6)(task_no 7)(state "completed"))))
(defrule from-auditor-to-controller
        (Task (task_no 7)(state "completed"))
        =>
```

```
        (assert (Task (pretask_no 7)(task_no 8)(state "completed"))))
(defrule from-supervisor-to-controller
        (Task (task_no 6)(state "completed"))
        (WF_DomTripMaster (Duration ?dur))
        (test (< ?dur 7))
        =>
        (assert (Task (pretask_no 6)(task_no 8)(state "completed"))))
(defrule from-controller-to-personel-end
        (Task (task_no 8)(state "completed"))
        =>
        (assert (Task (pretask_no 8)(task_no 9)(state "completed")))
        (assert (Task (pretask_no 8)(task_no 0)(state "completed"))))
(defrule from-personel-to-end
        (Task (task_no 9)(state "completed"))
        =>
        (assert (Task (pretask_no 9)(task_no 0)(state "completed"))))
```

D.3 Source Codes for Responsibility-Rule Frames

```
(defrule find-taskagent
        (Task (pretask_no ?pr_task)(task_no 2)(state "completed"))
        (WF_DomTripTraveller    (T_Id    ?u_id) (A_Id ?agent_id))
       (test (<> ?agent_id 0))
        =>
        (assert (ActorMapping (pretask_no ?pr_task)(task_no 2)(actor_id ?agent_id))))
(defrule find-trveller
        (Task (pretask_no ?pr_task)(task_no 3)(state "completed"))
        (WF_DomTripTraveller (T_Id    ?traveler_id)(A_Id ?agent_id))
        =>
        (assert (ActorMapping (pretask_no ?pr_task)(task_no 3)(actor_id ?traveler_id))))
(defrule find-accountowner
        (Task (pretask_no ?pr_task)(task_no 4)(state "completed"))
        (WF_DomTripAccount (Account_Own_Id ?owner_id))
        =>
        (assert (ActorMapping (pretask_no ?pr_task)(task_no 4)(actor_id ?owner_id))))
(defrule find-department-officer
        (Task (pretask_no ?pr_task)(task_no 9)(state "completed"))
        (WF_DomTripTripTraveller (T_Department ?dept_id))
       (w_acti_charge (task_no 9)(dept_id ?dept_id)(worker_id ?officer_id))
        =>
        (assert (ActorMapping (pretask_no ?pr_task)(task_no 9)(actor_id ?officer_id))))
(defrule find-supervisor
        (Task (pretask_no ?pr_task)(task_no 5)(state "completed"))
        (WF_DomTripTraveller (T_Id ?t_id)(T_Department ?dept))
        (Department (dept_id ?dept)(mnger_pos ?m-pos))
        (WorkFor (dept_id ?dept)(actor_id ?supervisor_id)(position ?m-pos))
        =>
        (assert (ActorMapping (pretask_no ?pr_task)(task_no 5)(actor_id ?supervisor_id)))
        (assert (traveler ?t_id ?dept)))
(defrule manager's-supervisor
```

```
        (declare (salience 5))
        ?f1<-(ActorMapping (pretask_no ?pr_task)(task_no 5)(actor_id ?supervisor_id))
        ?f2<-(traveler ?supervisor_id ?dept)
        (Department (dept_id ?dept)(parent ?sup_dept))
        (Department (dept_id ?sup_dept)(mnger_pos ?sm_pos))
        (WorkFor (dept_id ?dept)(actor_id ?sm_id)(position ?sm_pos))
        =>
        (retract ?f1 ?f2)
        (assert (ActorMapping (pretask_no ?pr_task)(task_no 5)(actor_id ?sm_id))))
 (defrule president-of-KAIST
        (declare (salience 10))
        (Task (pretask_no ?pr_task)(task_no 5)(state "completed"))
        (WF_DomTripTraveller (T_Id ?president_id))
        (WorkFor (actor_id ?president_id)(position 100))
        =>
       (undefrule find-supervisor)
       (assert (ActorMapping (pretask_no ?pr_task)(task_no 5)(actor_id ?president_id))))
(defrule find-auditor
        (Task (pretask_no ?pr_task)(task_no 6)(state "completed"))
        (w_acti_charge (dept_id 2003)(task_no 6)(worker_id ?auditor_id))
        =>
        (assert (ActorMapping (pretask_no ?pr_task)(task_no 6)(actor_id ?auditor_id))))
 (defrule find-account-controller-gyohak
        (declare (salience 100))
        (Task (pretask_no ?pr_task)(task_no 7)(state "completed"))
        (WF_DomTripTraveller (T_Department ?dept))
        (w_dept_gyohak (dept_id ?dept)(gyohak_id ?gyohak))
        (w_acti_charge (dept_id ?gyohak)(task_no 7)(worker_id ?controller_id))
        =>
       (undefrule find-account-controller-jaemoo)
        (assert (ActorMapping (pretask_no ?pr_task)(task_no 7)(actor_id ?controller_id))))
(defrule find-account-controller-jaemoo
        (declare (salience 10))
```

```
        (Task (pretask_no ?pr_task)(task_no 7)(state "completed"))
        (w_acti_charge (dept_id 2093)(task_no 7)(worker_id ?controller_id))
        =>
        (assert (ActorMapping (pretask_no ?pr_task)(task_no 7)(actor_id ?controller_id))))
 (defrule find-personel
        (Task (pretask_no ?pr_task)(task_no 8)(state "completed"))
        (w_acti_charge (dept_id 2102)(task_no 8)(worker_id ?personel_id))
        =>
        (assert (ActorMapping (pretask_no ?pr_task)(task_no 8)(actor_id ?personel_id))))
(defrule find-final
        (Task (pretask_no ?pr_task)(task_no 0)(state "completed"))
        =>
        (assert (ActorMapping (pretask_no ?pr_task)(task_no 0)(actor_id 0))))
```