On the Convergence of Autonomous Agent Communities

Hong Zhu
School Of Technology, Oxford Brookes University,
Oxford OX33 1HX, UK, Email: hzhu@brookes.ac.uk

Fang Wang
Pervasive ICT Research Centre, British Telecom,
Orion 1/12. Ipswich IP5 3RE, UK, Email: fang.wang@bt.comand

Shufeng Wang
National Lab. for Parallel and Distributed Processing,
Changsha, 410073, China, Email: shufeng.wang@gmail.com

Abstract

Community is a common phenomenon in natural ecosystems, human societies as well as artificial multi-agent systems such as those in web and Internet based applications. In many self-organizing systems, communities are formed evolutionarily in a decentralized way through agents' autonomous behavior. This paper systematically investigates the properties of a variety of the self-organizing agent community systems by a formal qualitative approach and a quantitative experimental approach. The qualitative formal study by applying formal specification in SLABS and Scenario Calculus has proven that mature and optimal communities always form and become stable when agents behave based on the collective knowledge of the communities, whereas community formation does not always reach maturity and optimality if agents behave solely based on individual knowledge, and the communities are not always stable even if such a formation is achieved. The quantitative experimental study by simulation has shown that the convergence time of agent communities depends on several parameters of the system in certain complicated patterns, including the number of agents, the number of community organizers, the number of knowledge categories, and the size of the knowledge in each category.

Keywords:
Adaptive systems, Self-organization, Autonomous agent, Community formation, Recurrence properties, Reachability, Stability, Convergence

1. Introduction

1.1. Motivation

Community formation is a common phenomenon in natural ecosystems and human societies. For example, a community can be defined as a group of people (or organisms) who have something in common to share [1]. The substance of shared elements varies widely, from a situation of interest to lives and values. In the past a community often refers to a group of people or entities that live together in the same area. Nowadays the concept of community is not limited by geographical locations, as the wide usage of Internet easily brings together people or entities situated at distributed locations to form a kind of virtual community. Consequently, the sizes of Internet-based communities may vary from tens to millions. For instance, the number of users with running machines in a file-sharing application, Gnutella, was reported to be 1,200,000 in March 2005[(1)], and the sheer number of indexable web pages was already over 10^9 in year 2000 [(2)]. It is not scalable for each peer to interact with all the others to discover appropriate resources, even to store information about all the others. To enable efficient accesses to internet resource as well as to facilitate various types of other service, communities are proposed and constructed in the context of peer-to-peer computing [2, 3].

The considerably increasing dimensions and complexity of contemporary Internet-based communities require substantial management work to organize and administrate proper groups in a large-scale distributed environment. A series of computational techniques have been proposed to automate this process. Among these techniques, self-organization has the advantages of being decentralized, evolutionary, autonomous and efficient.

Generally speaking, a self-organised agent communities consists of a number $n>0$ of autonomous peer agents A_1, A_2, …, A_n and a number $l>0$ of community organizers G_1, G_2, …, G_l. Each peer agent A_i has a subset $\{K_{i,1}, K_{i,2}, \ldots, K_{i,si}\}$ of knowledge, where each piece of knowledge K is classified into a one of r categories $\{C_1, C_2, \ldots, C_r\}$. Each peer agent is registered to an organizer with its set of knowledge. If an agent wants to access a particular piece of knowledge that it does not possess, it will search for the peers who have the knowledge and obtain the assistance from the peer. The search starts within the community by submitting a request to its organizer, which looks at its registry of the members. If the search fails within the community, the organizer of the community will contact other organizers for assistance. Peer agents may move from one community to

(1) See URL: http://www.limewire.com/english/content /netsize.shtml.
(2) See URL: http://www.inktomi.com/webmap/, accessed January 2000

another from time to time in order to be grouped with other agents that are interested and possess the knowledge of in the same category. A peer agent's movement is autonomous and based on a computational algorithm for its decision making using the local information about the communities involved.

However, such computational algorithms are difficult to understand because each member behaves autonomously without any central control and only based on the local knowledge while hoping the whole system evolves to form communities. It is unclear whether mature and optimal communities will always be formed and whether such formations will be stable. This paper studies these problems for a series of agent community formation algorithms by both a formal logic approach and an experimental simulation approach.

1.2. Related work

1.2.1. Research in community organization

A common method of community organization is to cluster entities according to their similarity, which can be back dated to 1970s [4, 5]. This method selects proper representations of entity features or patterns to calculate distance or proximity of entities based on feature differences. Communities are then created by grouping together those entities with sufficiently close proximity; see [6] for a comprehensive review on data clustering. Charikar *et al* [7] and Fisher [8] extended the clustering method to deal with new joining entities. More recently, Khambatti *et al* [9] and Ogston *et al* [10] further developed the method in a decentralized peer-to-peer environment. Because this method employs pre-defined features and computing models to generate communities, it may involve a significant amount of computation and re-clustering in dynamic situations, especially when entity features continuously change with time. Moreover, because entity features or patterns are usually difficult to extract and choose, inappropriate feature designs or representations would inevitably deteriorate the clustering results. The formed communities therefore may include mismatched entities with mistaken attributes.

In addition to entity proximity, communities can be formed based on entity associations. Iamnitchi et al [11] utilized data-sharing graphs to capture common user interests in data. Users that requested similar files had strong links in the graph so were formed into the same interest-based communities. By using relevant graph techniques such as maximum flow and minimum cut, in a recent study Iamnitchi et al [11] discovered a series of web communities in which members had more links to each other than to non-members. This method requires the full knowledge of the associations

between entities in order to perform community formation. This prerequisite sometimes made the community formation inapplicable in large-scale and dynamic applications.

An outstanding characteristic of communities is that entities working as a group usually exhibit behavior that goes beyond the simple addition of individual functions. This phenomenon in natural systems has attracted researchers' attention for a long time; c.f. [12, 13]. A similar behavior can also be found in artificial systems, such as in decentralized computer applications including Freenet [14] and Anthill [15]. Despite the absence of any centralized control point, clusters of specialized nodes in handling similar queries were gradually built in these applications, while the nodes simply cached a copy of query answers they had transferred. Flake et al. [16] discovered web communities on the Internet, though the web pages were written by independent creators. These systems possess a common feature of self-organization.

Based on the principle of self-organization, Wang [17] proposed a novel solution to community formation problem. It addressed community management in a decentralized way by taking advantage of autonomous agents. Cid-Sueriro and Wang [18] proved that the average formation time of the self-organizing communities increased linearly with the log of the number of users and also linearly with the number of middle agents.

However, the logic properties of such systems are unknown. For instance, will such a system always reach an optimal configuration in order to maximize its efficiency? If a system reaches such a state, will the state be stable? This paper aims to answer these open questions, which are essential for building sound and reliable online communities.

1.2.2. Research on Other Agent Organisation Paradigms

Agent communities can be regarded as a special type of multi-agent organizations. A wide range of organizational strategies and their combinations have emerged in the literature, which include hierarchies, teams, congregations, societies, federations, markets, and coalitions; see [19] for a recent survey on the research on multi-agent organizations and [20] for a collection of research papers that reflect the current research frontier on this topic. However, agent communities as a type of agent organizations have not been studied satisfactorily as discussed in the previous section. As Horling and Lesser stated, each type of agent organization is characterized by the collection of roles, relationships and authority structures which govern its behaviour. While the general theories of multi-agent systems applies to agent communities, the specific features and properties of agent communities cannot be directly derived from the results about other agent organization paradigms because of the differences between their characteristics. The following

discusses such differences.

A. Coalition.

Agent coalition is one of the organization paradigms that have similar features of agent communities studied in this paper. Agent coalition formation has been widely investigated [21, 22, 23, 24] in stable and dynamic environments [25], in software agents and embedded robots [26], and has even been applied to help artists control representations on a canvas [27], etc. Usually, agent coalitions are goal-directed and short lived: they are formed when a group of agents agree to cooperate in order to perform a task [24] and dissolve when the needs no longer exist. For example, in [23], the purpose of a coalition is represented as a global goal, which is decomposed into a number of subgoals, and each individual in the coalition has its own local goal(s) that matches one or more of the subgoals. The agents in a coalition are expected to coordinate their activities in a manner appropriate to the coalition's global goal and to maximize the group's and/or personal utilities. In this sense, agent communities differ from coalitions in that the agents to come to form a community with a rather simple goal of sharing information. More precisely, agents join in a community mainly because they are interested in a common topic and wish to share or exchange information or resources on this topic. Furthermore, agent coalition usually has strict membership requirement (for goal achieving) and often employ mechanisms that obtain consensus among agents before an agent is allowed to change its coalition membership. For example, in [28], the majority voting by the current coalition members is used to determine whether to allow an agent to join a coalition in the study of the convergence to stable coalitions. In the coalition formation algorithms proposed [23], the locally calculated weights of candidate coalitions are announced globally and a comparison of all announcements determines which coalition candidate is to be adopted by the all agents. In contrast, in the agent communities paradigm, an agent may join or quit a community autonomously mainly by the agent's own decision without approval by other community members. The main criterion for an agent to make the decision of joining or leaving a community is whether the agent can benefit from the community. By the term 'short lived', we meant that an agent organization is not permanent. In that sense we say that a coalition is usually short lived although it may last for a long time.

It is worth noting that, there is no widely accepted definition of the word 'coalition' in the literature of multi-agent systems. Some researchers regard coalition as more general and at a higher level of abstraction than a specific agent organizational structure. For example, in [27], a coalition is not just one kind of an organizational structure. Instead, it

is seen more as a framework in which many different organizational structures may be possible. From point of view, agent community is a much more concrete and specific kind of agent organizations in comparison to coalition in such a definition. Moreover, a coalition must have a set of rules for dealing with accepting additional agents and dissolving a coalition while the specific rules may vary [27]. One particular example of such rules is that an agent can join a coalition in order to gain happiness through the coalition's actions. Unfortunately, it may also lose happiness by taking coalition's actions. It is thus allowed to leave a coalition if its happiness falls below a threshold. This is very similar to agent communities studied in this paper. However, in the coalition framework, an agent's leaving may have much more profound impact than an agent's leaving from a community. As pointed out in [27], "if an agent selects to leave [a coalition], it can make sense to dissolve a coalition". In contrast, agent communities are not dissolved by any individual agent's leaving.

B. Team.

Similar to agent coalitions, agent teams also consist of a number of cooperative agents which have agreed to work together towards a common goal, but in comparison to coalitions, teams attempt to maximize the utility of the team rather than that of the individual members. Typically, members of a team are expected to take different roles to address different subtasks required to achieve the team goal. Agent communities differ from agent teams in that there is usually not a set of roles or subtasks of significant differences to be taken by the members. For example, STEAM [29] facilitates explicit specification of the relationship between a team operator and individual's or subteam's contributions to it based on the notion of roles. Here, a role is an abstract specification of the set of activities an individual or subteam undertakes in service of the team's overall activity. Role allocation and reallocation is one of the most challenging problems in multi-agent team organizations [30]. The convergence issues studied in the research on agent teams are focused on the formation of team decisions, which is known as multi-agent team decision problem rather than the formation of teams [29, 30, 31]. Although team formation, which is also known as team construction, is one of the key steps in the operation of agent team systems, in a static and reasonably sized agent population, team members can be determined off-line as a part of system design. In dynamic environments, team members can be dynamically discovered and assessed for selection using well-known discovery mechanisms such as the contract net protocol or matchmaker intermediaries [32].

C. Congregation.

Agent congregations are also groups of individuals who have banded together into a typically flat organization as coalitions and teams, but they are not formed with a single specific goal in mind. Instead, congregations are formed among agents with similar or complementary characteristics to facilitate the process of finding suitable collaborators. This is also one of the characteristics of agent communities. However, congregating agents are expected to be individually rational for maximizing their local long-term utility, which determines how agents select congregation [33]. In contrast, we do not assume the existence of such rationality, or an explicit utility for individual agents or implicit usage of such utility in agent community formation. Agent congregation, therefore, can be regarded as a subclass of agent communities. The convergence issue of the optimal congregation formation problem, i.e. how agents self-organize to find the correct congregation, has been investigated by Brookes and Durfee [33]. They regard a congregation system as a set of agents who simultaneously learn which other agents it wants to interact with and applied the CLRI model [34] to determine the complexity of congregation formation problem. They concluded that if agents are unable to describe congregations to each other, convergence problem is exponential in the number of agents. They then introduced labelers as a means of coordinating agent decisions, thus reduced the problem's complexity. They also used simulation experiments for congregations without labels, with flat labels and hierarchical labels. The agent communities paradigm studied in this paper differs from that of agent congregations in the way that how agents decide to move from one group to another. In particular, in the congregation paradigm, each agent has a payoff function to measure the value of a congregation that the agent is in, but can only estimate the payoff value of a congregation that it is not in when considering join a new congregation. The agent decides which congregation to join by maximizing the estimated payoff value. In our case of agent communities, we assume such a payoff value of a community, which called the strength of the community in this paper, is always available from the community organizer no matter whether the agent is a member of the community. This is proved to have a significant impact on the convergence of community formation as we will see later in the paper.

D. Society.

Agent societies are organizational paradigms that individuals of different stripes are free to come and go at will but must confined by the constraints imposed on their behaviours while remaining in the society, which are known as social laws, norms or conventions. The focus of research on agent societies has been on the social laws, such as dynamic norm formation; see, for example, [35 , 36 , 37]. In contrast, normative

behaviours are not the focus of research on agent communities, whose behaviours are relatively fixed.

1.3. Overview of the paper

In this paper, we will extend the work by Wang [17] and Cid-Sueriro and Wang [18] and formally investigate a variety of subtle variants of the algorithm that they proposed and studied. The main contributions of the paper include the following.

1. A formal specification of self-organizing agent communities and its variants is presented by using the formal specification language SLABS [39, 40]. The properties of agent community formation are formally expressed as scenarios in the dynamic executions of the system, such as maturity and optimal community formations. Their features are expressed as the recurrence properties of multi-agent systems, such as the reachability, stability and convergence of the scenarios. These recurrence properties have been proposed and formally defined and studied in the formal system of Scenario Calculus [46]. This enables the application of scenario calculus to the study of agent community formation problem.

2. The qualitative formal study proves that the formation of mature and optimal communities always occurs if agents behave based on collective knowledge of communities, and such formations are stable. In contrast, if agents behave solely based on individual agent knowledge or the information about community sizes, mature and optimal community formation cannot be guaranteed, and it may be instable even if such a formation is reached.

3. The quantitative experimental study by simulation shows that convergence time depends on all parameters of the agent community system, which include the number of agents in the system, the number of community organizers, the number of knowledge categories, and the size of knowledge in each category.

The remainder of the paper is organized as follows. Section 2 briefly reviews the SLABS language and Scenario Calculus. Section 3 formally specifies a variety of self-organizing agent communities in SLABS, formally defines the required emergent behaviours of agent community formation as scenarios and proves their common features by applying Scenario Calculus. Section 4 further formally study the recurrent properties of various types of agent community systems and prove or disprove their reachability, stability and convergence of agent community systems to mature and optimal scenarios. Section 5 reports the experimental study of the convergence speed of agent community formation using simulation. The impact of various parameters of agent community

systems on convergence time is investigated. Section 6 concludes the paper with a discussion of further work.

2. Overview of SLABS and Scenario Calculus

In this section, we briefly review the formal specification language SLABS and Scenario Calculus. Details of the language and Scenario Calculus can be found in [39, 46].

SLABS, which stands for *Specification Language for Agent-Based Systems*, is the first and only general purpose formal specification language that is designed for engineering agent-based systems [38, 39, 40]. It has been successfully used to specify various types of agent systems, including personal assistants [39], speech-act based interaction among agents, an evolutionary multi-agent ecosystem [41, 42], emergent behaviour of multi-agent systems [43], autonomous agent-based web services [44], etc.[3] More recently, Mao et al. [45] extended the meta-model underlying SLABS for more flexibility and expressiveness in the development of adaptive multi-agent systems. SLABS language's *caste* facility is particularly suitable for specifying systems that consists of a large number of agents classified into a number of types and interacting with each other through well-defined non-deterministic and probabilistic behaviour rules. The scenario calculus defined on the basis of SLABS [46] supports the reasoning about the dynamic behaviour of such multi-agent systems.

As in [39, 47], an agent is defined as a proactive computational entity that situates in its designated environment and takes actions autonomously according to its own behavior rules depending on its own view of the situation in the environment.

In particular, in addition to a unique identity, each *agent* $A=< S_A, \Sigma_A, R_A, E_A>$ contains a set S_A of variables to represent its state, a set Σ_A of actions that it is capable of performing, a set R_A of behavior rules that determines when and which action in Σ_A to take, and a set E_A of other agents or objects in the system that it is observing in order to know the situation in the environment.

Similar to that objects are created and declared as instances of classes, agents are defined or created as instances of *castes*. Each caste specifies a set of actions, a set of state variables, a set of behavior rules and a description of the environment in the form of a set of agents in the system. Therefore, every agent of the caste has the corresponding elements specified by the caste. A multi-agent system (MAS) consists of a number of agents that are classified by a number of castes. When an agent A is an instance of caste C, we write $A \in C$. Similar to the inheritance relationship between classes, an inheritance

[3] See http://cms.brookes.ac.uk/staff/HongZhu/SLABS/index.htm for more details.

relationship between castes is defined and we write $C_1 < C_2$ if C_1 inherits C_2, and we have that $A \in C_1$ and $C_1 < C_2$ implies that $A \in C_2$.

In the specification of caste in SLABS [39], the state variables are declared by giving the variable names and their corresponding data types. An action is declared in the form of an identifier plus a sequence of parameters and their types. An environment description is in the form of a sequence of clauses in the following form:

- `'Id: caste-name'` : where `Id` is an agent's name, the specific agent `Id` of the caste is in its environment and thus observed by the agent;
- `'All: caste-name'` : all the agents of the caste are in the environment, and thus observed;
- `'Var Id : class-name'` : where `Id` is a variable that ranges over the caste, the agent assigned to the variable `Id` is in the environment and thus is observed.

In SLABS, a behavior rule is written in the form of

[<*Rule Name*>*:*] *St* | [*Pr*] –> *Act,* IF *Sc* WHERE *Pre-cond*

where *Pr* is an optional real number expression with value between 0 and 1; *Sc* is a scenario of the system's state and *Act* is an action; *St* is the state of the agent; *Pre-cond* is a Boolean expression that represents the pre-condition of the rule. It means that when the environment is in the scenario *Sc* and the agent is in state *St*, the agent will take the action *Act* with the probability *Pr,* if the pre-condition is true. When there is more than one possible rule to apply and/or more than one way a rule can be applied, a random choice will be made according to the probability *Pr*. When the probability is omitted, the uniform distribution is assumed. Thus, the choice is non-deterministic.

Informally, a *scenario* is a runtime situation in the operation of the MAS. As in SLABS, it is defined by a predicate on the states of the agents and the actions taken by the agents. Formally, *scenario* expressions are defined inductively as follows. Here, we only give an informal semantics of the scenario expressions. The formal semantics can be found in [39].

Definition 2-1. (*Patterns and Scenarios*)

Let *A* and *B* be agent identifiers or variables that range over agents of a specific caste, *C* be a caste name, *Sc* and Sc_1, Sc_2 be well-formed scenario expressions. We define scenario expressions inductively as follows.

- A *Boolean expression* on an agent's state is a scenario expression. It means that the system is in the scenario when the corresponding agent's state makes the Boolean expression to be true;

- *A:P* is a scenario expression, if *A* is an agent and *P* is an activity pattern. It means that the system is in the scenario when agent *A*'s activity matches the pattern *P*;
- *A*=*B* (or *A*≠*B*) is a scenario expression, where *A* and *B* are agent variables or identifiers. It means that the system is in the scenario when the agents assigned to the variables/Ids *A* and *B* are the same agent (or not the same agent);
- *A*∈*C* is a scenario expression, where *A* is an agent and *C* is a caste. It means that the system is in the scenario if agent *A* is in caste *C;*
- ¬*Sc* is a scenario expression, where *Sc* is a scenario expression. It means that the system is in the scenario when it is not in scenario *Sc*;
- ∀*x*∈*C*.*Sc* is a scenario expression, where *C* is a caste and *Sc* is a scenario expression. It means that the system is in the scenario if *Sc*[*x*/*A*][(4)] is true for all agents *A* in caste *C*;
- $\exists_{[m]}x \in C.Sc$ is a scenario expression, where *C* is a caste and *Sc* is a scenario expression. It means that the system is in the scenario if there are at least *m* agents in caste *C* such that *Sc*[*x*/*A*] is true, where the default value of the optional expression *m* is 1;
- Sc_1 & Sc_2 is a scenario expression, where Sc_1 and Sc_2 are scenario expressions. It means that the system is in the scenario if both scenario Sc_1 and scenario Sc_2 are true;
- $Sc_1 \vee Sc_2$ is a scenario expression, where Sc_1 and Sc_2 are scenario expressions. It means that the system is in the scenario if either scenario Sc_1 or Sc_2 or both are true;

A *pattern P* is represented in the form of $[a_1, a_2, \ldots, a_k]$, $k \geq 0$, where a_i is an action, or an assignment to the state variables, or an action variable (which matches an action), or wild card \$, or silence τ. A pattern $[a_1, a_2, \ldots, a_k]$ means that the agent has take a sequence of $n \geq k$ actions and the last k actions match $a_1, a_2, \ldots, a_k$, respectively, where a_k is the most recent action. When $k=0$, the pattern [ ] is true if the agent has not taken any action since its creation. Another special pattern is [\$], which matches to all actions. □

Examples of scenario expressions can be found in [39].

The Scenario Calculus proposed in [46] is a formal system about the relations on scenarios. It defines the notion of scenario inclusion and orthogonality relations and transitions between scenarios in MAS. Informally, scenario Sc_1 include Sc_2 means that the system is in Sc_1 implies it is also in scenario Sc_2. Scenario Sc_1 is orthogonal to scenario Sc_2 means that the system will never be in scenario Sc_1 and Sc_2 at same time. A

(4) *Sc*[*x*/*A*] is the scenario expression obtained from *Sc* by systematically replacing free occurrences of variable *x* with *A*.

scenario Sc_1 transits into scenario Sc_2 means that the system can move from a state in scenario Sc_1 to a state in scenario Sc_2. The following formal definition of these notions is taken from [46]. (5)

Let r be an execution of a MAS $\boldsymbol{M}$, t be a time moment of the execution, Sc be a well-formed scenario expression of $\boldsymbol{M}$. We write $r\downarrow t|=Sc$ to denote that the system is in scenario Sc (i.e. Sc is true) at time t in the execution r.

Definition 2-2. (*Scenario inclusion and transition*)

Let Sc_1 and Sc_2 be two well-formed scenario expressions of a MAS $\boldsymbol{M}$.

A scenario Sc_1 *includes* scenario Sc_2 in $\boldsymbol{M}$, written $\boldsymbol{M} \models Sc_1 \Rightarrow Sc_2$ if and only if for all runs r and at all time moments $t \in T$, $r\downarrow t|=Sc_1$ implies that $r\downarrow t|=Sc_2$.

Scenario Sc_1 *transits to* Sc_2 in $\boldsymbol{M}$, written $\boldsymbol{M} \models S_1 \rightarrow S_2$, if and only if there is a run r of the system $\boldsymbol{M}$ and time moments $t_1 < t_2 \in T$ such that $r\downarrow t_1 \models S_1$ and $r\downarrow t_2 \models S_2$. □

Readers are referred to [46] for properties of these relations.

As discussed in [46], a recognizable phenomenon of the dynamic behavior of a MAS can be specified as a scenario. The recurrence properties of such a phenomenon can then be specified and proved as properties of scenarios. The following are the recurrence properties defined and studied in [46], which will also be used in this paper to study the properties of agent communities.

Definition 2-3. (*Recurrence properties of scenarios*)

Let $\boldsymbol{M}$ be any given MAS and Sc be a given well-formed scenario expression of $\boldsymbol{M}$.

System $\boldsymbol{M}$ *always reaches* scenario Sc, written $\boldsymbol{M} \rightarrowtail Sc$ if for all runs r there is a time moment t such that $r\downarrow t \models Sc$.

Scenario Sc is *stable* in MAS $\boldsymbol{M}$, written $\boldsymbol{M}@Sc$ if for all runs r and all time moments $t \in T$, $r\downarrow t \models Sc \Rightarrow \forall t' > t \in T.\ (r\downarrow t' \models Sc)$.

MAS $\boldsymbol{M}$ *always converges* to scenario Sc, written $\boldsymbol{M} \looparrowright Sc$ if for all runs r there is a time moment t such that $\forall t' \in T.(t' \geq t \Rightarrow r\downarrow t' \models Sc)$. □

It is worth noting that for each run r, as time t approaches infinity, the probability distribution of random chooses of applicable rules must satisfy the specified probability constraints in behavior rules. The word 'always' must be understood in this context.

The following lemma gives the relationships between reachability, stability and convergence.

(5) The orthogonal relation between scenarios is not used in this paper. Thus, it is not included in the formal definition.

Lemma 2-1.

For all MAS $\boldsymbol{M}$ and well-formed scenario Sc,

(a) $(\boldsymbol{M} \rightarrowtail Sc \ \& \ \boldsymbol{M}@Sc) \Rightarrow \boldsymbol{M} \looparrowright Sc$.

(b) $\boldsymbol{M} \looparrowright Sc \Rightarrow \boldsymbol{M} \rightarrowtail Sc$.

Proof.

(a) Assume that $\boldsymbol{M} \rightarrowtail Sc$ and $\boldsymbol{M}@Sc$. By Definition 2-3, from $\boldsymbol{M} \rightarrowtail Sc$ we have that for all runs r there is a time moment t_r such that $r{\downarrow}t_r \models Sc$. By Definition 2-3, from $\boldsymbol{M}@Sc$ we have that $\forall t' > t_r \in T.\ (r{\downarrow}t' \models Sc)$. Therefore, we have that $\boldsymbol{M} \looparrowright Sc$. Thus, statement (a) is true.

(b) It is straightforward from Definition 2-3. □

Note that, $\boldsymbol{M} \looparrowright Sc$ does not imply $\boldsymbol{M}@Sc$. Please see section 4 for a counterexample.

In addition to the above properties, we can also prove the following lemma about the relationship between recurrence properties and scenario inclusion. It is useful in this paper. The proof is straightforward, thus omitted for the sake of space.

Lemma 2-2.

(a) For all MAS M and well-formed scenarios Sc *and* Sc', $M \looparrowright Sc$ and $M \models Sc \Rightarrow Sc'$ imply that $\boldsymbol{M} \looparrowright Sc'$.

(b) $\boldsymbol{M}@Sc \Leftrightarrow$ for all well-formed scenarios Sc', $\boldsymbol{M} \models Sc \rightarrow Sc'$ implies $\boldsymbol{M} \models Sc' \Rightarrow Sc$. □

3. Self-Organised Agent Communities

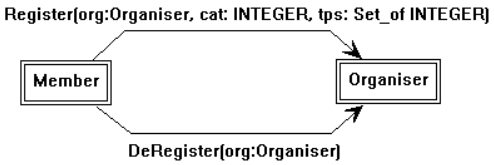
This section formally specifies various models of self-organizing agent communities in SLABS and studies their general properties using Scenario Calculus.

3.1. The Basic Model of Agent Communities

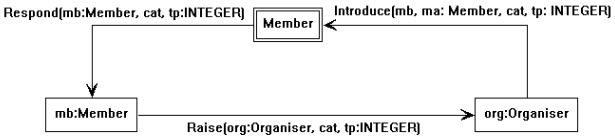
In the model of self-organizing communities [17, 18], there are two types of agents: members and organizers. Each organizer organizes a community by keeping a registry of the members of its community, handling the queries made by the members, and collaborating with other communities. Each member is registered only with one organizer at any time.

Each member is interested in a particular category of knowledge and has a certain set of knowledge of the same category. When a member registers with an organizer, it reports its interested category and the set of knowledge that it has; see Figure 3-1(a), where arrows represent the actions taken by an agent and observed by the other.

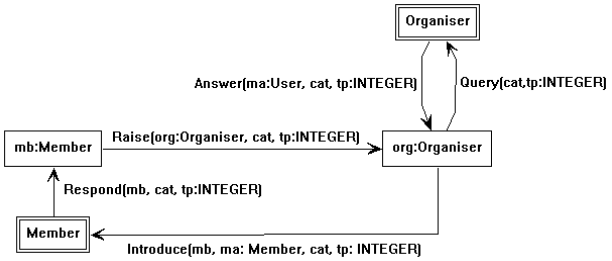
A member *R* may raise a query about a specific topic *tp* of its interested category *cat*. This query *tp* is submitted to the organizer, which will then search for a member in its community that knows this topic *tp*. If such a member is found, say member *S*, the organizer will introduce agent *R* to the member *S* together with the query on topic *tp*. Member *S* will then respond with an answer to the query *tp*; as shown in Figure 3-1(b). If the organizer *G* cannot find a member that is good at this topic within its community, it will ask for help from another organizer chosen at random, say *M*, by making a query on the topic *tp* of category *cat*. If *M* finds a member *T* in its community that knows the topic, it will answer the query and pass the identity of the member *T* to the organizer *G*. The organizer *G* will then introduce *T* to *R*. The member *T* will take the same action to answer *R*'s question; as shown in Figure 3-1(c).



(a) Registration



(b) Raise a question and find answers within a community



(c) Raise a question and find answers outside a community

Figure 3-1. Interaction between members and organisers

The structure and behaviors of members and organizers can be formally specified in SLABS as follows.

```
CASTE Member;
  ENVIRONMENT organiser: Organiser, All: Member;
  VAR
    Category: INTEGER;              (* the category of the agent *)
    Knowledge: SET_OF INTEGER;      (* a set of topics that it knows *)
  ACTION
```

```
    Raise(toWhom: Organiser, category: INTEGER, topic: INTEGER );
    Respond(Whom: Member, category: INTEGER, topic: INTEGER );
    Register(toWhom: Organiser, category: INTEGER,
              knowledge: SET OF INTEGER);
    DeRegister(toWhom: Organiser);
BEGIN
  <Initialisation>:
    [] |–> Category:=x; Knowledge:= S; Score:= y; organiser:= M;
           Register(organiser, Category, Knowledge),
               where (x>0)&(y>0)&(S≠∅)&(M∈Organiser);
  <Query>:
    [$] |–> Raise(organiser, Category, tp), where (tp ∉ Self. Knowledge);
  <Respond>:
    [$] |–> Respond(Raiser, cat, tp);
               if organiser:[Introduce(Raiser, Self, cat, tp)];
END Member;
```

Note that, the <*Query*> behavior rule specifies that a member will raise a question on any topic of the category at random as far as the member does not know the topic. We assume that the non-deterministic behavior is fair, thus it will ask all the questions in any order eventually. In other words, if a topic of the category is not in the agent's knowledge, the agent will raise queries on the topic at some time moments in all runs of the system although the specific time moments can be random from run to run.

```
CASTE Organiser;
  ENVIRONMENT All: Member, All: Organiser;
  VAR
     Registry: SET_OF <member: Member; category: INTEGER;
                       knowledge: SET OF INTEGER>;
  ACTION
     Query(org: Organiser; category: INTEGER; topic: INTEGER);
     Answer(org: Organiser; member: Member; cat, topic: INTEGER);
     Introduce(whom, toWhom: Member, cat, topic: INTEGER);
  BEGIN
   <Initialisation>:
      [] |–> Registry := {};
   <Register>:
     [$] |-> Registry := Registry + <mb,cat,kn>;
               if ∃mb∈ Member: [Register(Self, cat, kn)];
   <DeRegister>:
     [$] |–> Registry := Registry – <mb, cat, kn>;
               if ∃mb∈Member: [DeRegister(Self)];
               where cat = mb.Category & kn=mb.Knowledge;
   <Introduce>:
     [$] |–> Introduce(req, sv, cat, tp);
               if ∃req∈Member:[Raise(Self, cat, tp)],
               where ∃sv∈Member.((<sv, cat, knowledges> ∈ Registry)
                                 & (tp∈ knowledges));
   <Query another community>:
     [$] |–> Query(org, cat, tp);
               if ∃req∈Member:[Raise(Self, cat, tp)],
               where ¬(∃m∈Member.((<m,cat,kn>∈Registry) & (tp∈kn))
                        & (org∈Organiser);
   <Answer another community>:
     [$] |–> Answer(org, sv, cat, tp);
               if ∃org∈Organiser: [Query(Self, cat, tp)]
               where ∃m∈Member:(<m,cat,kn>∈Registry &(tp∈kn))
   <Introduction to another community>:
```

```
     [Query(org,cat,tp)]|->Introduce(mb,m,cat,tp);
               if ∃org∈Organiser:[Answer(Self,m,cat,tp)]
                  & ∃mb∈Member:[Request(Self,cat,tp)]
END Organiser.
```

Obviously, the performance of a world of self-organizing communities heavily depends on the configuration that members are grouped into communities. It is more efficient if a query raised by a member can be answered within the community. Assume that, at the beginning, members are registered with organizers at random. Therefore, the efficiency of the system cannot be guaranteed and thus reconfigurations of the communities are necessary.

In systems of self-organizing communities, in order to achieve optimized efficiency, agent communities reconfigure themselves through members' autonomous behaviors in moving from one community to another without global information. This is achieved by members changing their memberships to the communities. That is, a member deregisters from one organizer and then registers with another. In this model, members are autonomous to decide when or where to move. It is not controlled by the organizers or any global controller of the system.

Suppose that a member *R* raises a question on a topic, which is not known by any member of its community. While a member *T* of another community provides a successful service of answering *R*'s question. Then, members *T* and *R* will try to be in the same community. This can be achieved by either member *T* moving into member *R*'s community or member *R* moving into member *T*'s community. In this paper, we are only interested in such decision making rules that only use local information that are available from the involved member agents and their organizers rather than global information.

A simple rule to decide which member will move is that the agent who is in the more attractive community will stay while the one who is in the less attractive community will move. When the agents calculate a community's attraction in the same way, it is certain that one of them will stay and the other will move, thus they will be together after the actions. However, the complexity of the algorithms for self-organizing communities is due to the fact that each of these agents may move to another community subsequently. The situation is more complicated if agents calculate the attraction differently. In such cases, it may happen that both of the requester and the server move to the other community simultaneously, thus they may still be separated after taking the actions. A key question to be answered in the study of self-organizing communities is whether agents' moving between communities will lead to a globally optimal configuration even

if only local information is available.

The following subsection specifies some variants of agent communities.

3.2. Variants of Agent Communities

We classify two types of member agents according to the information used in their decision making rules.

Community attracted members (CAM) use information about the whole community, which is available from the organizer. We identify two further sub-types CAM agents as follows.

- *CAM-K (Community's amount of knowledge in the category)*: The agent measures a community's strength of attraction according to the total amount of knowledge in its interested category held by the agents registered with the community.
- *CAM-P (Community's number of agents in the specific category)*: The agent measures a community's strength of attraction according to its number of agents in the same category.

The second type of member agents is *personality attracted members* (PAM), which use information about the member agent only. There are also two sub-types of PAM agents.

- *PAM-R (Personal amount of knowledge of the service provider)*: The agent measures a community's strength of attraction according to the amount of knowledge that the specific service provider has in the category.
- *PAM-E (Personal attribute irrelevant to its knowledge)*: The agent measures a community's strength of attraction according to an attribute of the specific service provider, where the attribute is irrelevant to its knowledge.

Consequently, according to the types of members in the communities, we have the five different variants of self-organizing agent community systems as summarized in Table 3-1. The formal definitions of the variants will be given later in the paper.

TABLE 3-1. VARIANTS OF AGENT COMMUNITIES

Variants		Member type(s)	Main features
Type	Subtype		
CAM: Community attracted members	**CAM-K**: knowledge-based	CAM-K	An agent changes its community membership by moving into a community that collectively has more knowledge of the category.
	CAM-P: Population-based	CAM-P	An agent changes its community membership by moving into a community that has a larger number of agents of the category
PAM: Personality	**PAM-R:** Rational	PAM-R	An agent changes its community membership by moving into a community that has a member possessing more

attracted members	members		knowledge of the category than itself
	PAM-E: Emotional members	PAM-E	An agent changes its community membership by moving into a community that has a member that is more attractive than itself
Hybrid: Hybrid systems		CAM-K, CAM-P, PAM-R, PAM-E	The system contains members of more than one type. They may decide moving to another community according to difference rules.

3.3. Basic Properties of Agent Communities

It is desirable that a system of self-organizing communities demonstrates the dynamic behavior that its agents will gradually group into communities that members of the same category come together in one group and are registered with the same organizer. In order to formally define this phenomenon of system's dynamic behavior, we first introduce some notions and notations.

For the sake of simplicity, in the sequel, a community that is organized by organizer G will be referred to as community G.

Definition 3-1. (*Population*)

The *member population* of a category C in a community G at a time moment t is denoted by $P_t^G(C)$ and defined as follows.

$$P_t^G(C)=\{x\in_t Member \mid x.Category = C \ \&\ x.Organiser=G\}.$$

The *overall population* of the members of a category C in the whole world at time moment t is denoted by $P_t^*(C)$ and defined as follows.

$$P_t^*(C)=\{x\in_t Member \mid x.Category = C\}. \ \square$$

Because each member registers with one and only one organizer, we have the following lemma.

Lemma 3-1. For all categories C of knowledge, at all time moment t, we have that $P_t^G(C)$, $G\in Organiser$, is a disjoint partitioning of $P_t^*(C)$, i.e.

$$P_t^*(C)=\bigcup\{P_t^G(C)\,|\,G\in_t Organiser\},$$

and $$P_t^G(C)\cap P_t^{G'}(C)=\varnothing, \text{ if } G\neq G'. \qquad \square$$

Definition 3-2. (*Domain of knowledge*)

The *domain of knowledge* in category C in a community organized by G at time moment t is denoted by $D_t^G(C)$ and defined as follows.

$$D_t^G(C)=\bigcup\{x.Knowledge\,|\,x\in_t P^G(C)\}.$$

The domain of knowledge of category C in the whole system at time moment t is

denoted by $D_t^*(C)$ and defined as follows.

$$D_t^*(C)=\bigcup\{x.Knowledge \mid x \in_t P^*(C)\}$$

A category C of knowledge is *non-trivial*, if $D_t^*(C) \neq \varnothing$. □

Similar to Lemma 3-1, we have the following obvious property of the domains of knowledge.

Lemma 3-2. For all categories C of knowledge, at all time moment t, we have that

$$D_t^*(C)=\bigcup\{D_t^G(C) \mid G \in_t Organiser\} \text{ . } \square$$

However, $D_t^G(C)$, $G \in Organiser$, may have overlaps.

Definition 3-3. (*Closed world*)

A world of self-organizing communities is *closed* if no agent is added to or removed from the world during an execution. □

In a closed world, the overall population and the domain of knowledge in the whole system do not change with the time. Formally, we have the following lemma.

Lemma 3-3. In a closed world, we have that for all t, $t' \in T$,

$$P_t^*(C)= P_{t'}^*(C)$$

and

$$D_t^*(C)= D_{t'}^*(C). \qquad \square$$

Thus, in a closed world, the subscripts of t can be omitted. In the sequel, for the sake of simplicity, we assume that the world is closed.

In order to achieve optimal performance, it is desirable that an execution of a self-organizing community reaches the following scenarios.

Definition 3-4. (*Completeness w.r.t a category*)

At time moment t, a community organized by G is *complete* with respect to the knowledge category C if $D_t^G(C)= D_t^*(C)$, and written $Complete_t^C(G)$. □

Note that completeness is a well-formed scenario expression of self-organizing communities.

Definition 3-5. (*Maturity*)

A system of self-organizing communities is *mature*, if for every non-trivial category C of knowledge, there is a complete community with respect to C. Formally, a system is mature at time moment t, if the following scenario Mature is true at time t.

$Mature \equiv \forall C \in Category.(D^*(C) \neq \varnothing \Rightarrow \exists G \in Organiser.(Complete^C(G)))$. □

Definition 3-6. (*Optimal*)

A system of self-organizing communities is *optimal*, if every member is in a complete

community of its category. Formally, a system is optimal at time moment *t*, if the following scenario *Optimal* is true at time *t*.

$$Optimal \equiv \forall u \in Member.(u \in P^G(C) \Rightarrow Complete^G(C)),$$

where $G = u.Organiser$ and $C = u.Category$. □

The following lemma proves the relationship between *Mature* and *Optimal* scenarios.

Lemma 3-4.

In all worlds ***M*** of self-organizing agent communities, we have that

$$\boldsymbol{M} \models Optimal \Rightarrow Mature.$$

Proof.

Assume that at time moment *t*, the system *M* is in the *Optimal* scenario. Let C be any non-trivial category of knowledge, i.e. $D^*(C) \neq \emptyset$.

By Definition 3-2, there is a member agent *u* such that $u.Category=C$. By Lemma 3-1, there is an organizer agent G such that $u \in P^G(C)$. By Definition 3-6, we have that $Complete^G(C)$. Therefore, for any category *C* there is a community *G* such that *G* is complete w.r.t. *C*. By Definition 3-5, we have that at the same time moment *t*, the system is also in the Mature state. Thus, the statement of the lemma follows directly Definition 2-2. □

An important property of the self-organizing agent communities is the stability of the optimal scenario as proved in the following lemma.

Lemma 3-5.

In a closed world, for all systems *M* of self-organised communities, Optimal is a stable scenario. Formally, *M@Optimal*.

Proof. By the definition of the behavior rules of members, when a system is in the Optimal scenario, no agent will change its membership to its community because all queries can be answered locally within the community. □

Obviously, if a world of self-organizing communities is in the scenario of Optimal, all queries will be answered locally within the community. Thus, the performance of the system is optimal in this sense. It is desirable that an agent community always reaches the optimal state. Unfortunately, not every variant of agent community systems has this property. The following sections will study their properties.

4. Formal Analysis of Recurrence Properties

In this section, we study the recurrence properties of community formation by formally analysing the reachability, stability and convergence of the Mature and Optimal scenarios

in various types of agent community systems.

4.1. Worlds of Agents Attracted by Community Strengths

In this subsection, we study the model in which a member makes a decision about moving according to whether the other community is better than the current one.

4.1.1. Definition of the models

Suppose that a member *R* raises a query on topic *tp*, which is unknown to all other members of its community. Instead, a member *T* of another community provides a successful service to answer *R*'s query. Then, members *T* and *R* will decide whether it will move to the other community. In the CAM model, the rule to decide which member to move is that the member in the stronger community will stay while the one in the weaker community will move. When the strengths of the communities are equal, the member that raised the question will move. The specification of the caste CAM is given below. It is a sub-caste of Member; hence it inherits all state variables, actions, behaviour rules, and environment from the caste Member.

```
CASTE CAM <= Member;
BEGIN
  <Move To the Better Community>
    [$] |–>  DeRegister(organiser); Register(NewOrg); organiser:=NewOrg;
               if organiser:[Introduce(Self,Buddy,cat,tp)]
                  OR organiser:[Introduce(Buddy,Self,cat,tp)];
               where ((Buddy.organiser ≠ Self.organiser)
                  & (NewOrg = Buddy.organiser)
                  & (Buddy.Category = Self.Category)
                  &(Strength^Cat(Buddy.organiser)>Strength^Cat(Self.organiser))
  <Move To the Server's Community>
    [$] |–>  DeRegister(organiser); Register(NewOrg); organiser:=NewOrg;
               if organiser:[Introduce(Self,Buddy,cat,tp)];
               where ((Buddy.organiser ≠ Self.organiser)
                  & (NewOrg = Buddy.organiser)
                  & (Buddy.Category = Self.Category)
                  &(Strength^Cat(Buddy.organiser)=Strength^Cat(Self.organiser))
END CAM;
```

The sub-types of **CAM** systems differ in the way that the strength of a community is defined. For the **CAM-K** members, the strength is defined as the total amount of knowledge of category *C* known by the members of the community. Formally, the function $Strength^C(x)$, $x \in Organiser$, is defined as follows.

$$Strength^C(x) \triangleq \left\| \bigcup \{ y.Knowledge \mid y.Organiser = x \wedge y.Category = C \} \right\|. \qquad \text{Eq.(0.1)}$$

In other words, at any time moment *t*, we have that $Strength^C(x) = \left\| D_t^x(C) \right\|$.

For **CAM-P** systems, the attraction strength of a community is the population of members of the category *C*. Formally, the strength function is defined as follows.

$$Strength^C(x) \triangleq \left\| \{ y \mid y.Organiser = x \wedge y.Category = C \} \right\| \quad \text{Eq. (0.2).}$$

Equivalently, we have that for **CAM-P** systems, at all time moments t, $Strength^C(x) = \left\| P_t^x(C) \right\|$.

4.1.2. Convergence Properties

For **CAM-K** systems, we can prove that completeness with respect to a knowledge category is stable.

Lemma 4-1.

Let ***M*** be a closed world of CAM-K.

(a) When a community G is complete with respect to category C, its population of members of category C will never decrease after that. Formally,

$$\forall t \in T.(M{\downarrow}t| = Complete^G(C) \Rightarrow \forall t' \in T.(t' > t \Rightarrow P_{t'}{}^G(C) \supseteq P_t^G(C))).$$

(b) The state that there exists a community complete with respect to category C is stable. Formally, let C be any given category of knowledge. We have that

$$\textbf{\textit{CAM-K}}@(\exists G \in Organiser.(Complete^C(G))).$$

Proof.

(a) Assume that at time moment t, $A \in P_t^G(C)$ and $Complete^C(G)$ is true, i.e. community G is complete with respect to C. We prove by contradiction that it is impossible that at any time moment $t'>t$, $A \notin P_{t'}{}^G(C)$. Suppose that $A \notin P_{t'}{}^G(C)$. There are only two possibilities. First, agent A is no longer in the system. This is contradiction to the assumption that the system is closed. Second, agent A moved to another community H at time moment $t'>t$. By the definition of CAM-K, it can only be the result of applying either the rule <*Move to the better community*> or the rule <*Move to the server's community*>. The following proves by contradiction that both rules are not applicable.

Case 1: the rule <*Move to the better community*> was applied.

This means that $Strength(H) > Strength(G)$. By Definition 3-4, we have that $D^G(C)=D^*(C)$, because G is complete with respect to C. Hence, $Strength(G)=\|D^*(C)\| \geq Strength(H)$. This is in contradiction to the condition that $Strength(H) > Strength(G)$.

Case 2: the rule <*Move to the server's community*> was applied.

This means that an agent B of community $H \neq G$ provided a service to agent A. Thus, the query tp raised by agent A was unable to be answered by any agent in community G according to the behaviour rules of caste *Member*. This means that $tp \notin D^G(C)$. Since $tp \in B.Knowledge$, and $B.Category=A.Category=C$, $tp \in D^*(C)$. Therefore, $D^G(C) \neq D^*(C)$.

This is in contradiction to the condition that G is complete with respect to C.

Therefore, agent A does not move to any other community at any time moment $t'>t$. Thus, $P_{t'}^{G}(C) \supseteq P_t^{G}(C)$, for all $t'>t$.

(b) Directly follows the above proof of (a). □

Lemma 4-2.

In a closed world, for CAM-K systems, Maturity is stable. Formally, ***CAM-K*** $@Mature$**.**

Proof. Directly follows Lemma 4-1. □

The following lemma proves that ***CAM-K*** systems always reach the Optimal scenario.

Lemma 4-3.

In a closed world of ***CAM-K*** system, we have that

$$\textbf{\textit{CAM-K}} \rightarrowtail \forall u \in Member.(u \in P^{G}(C) \& Complete^{G}(C)).$$

Proof.

Assume that a member u moves from community G to G'. This must be the result of applying either the rule of <*Move to the Better Community*> or the rule <*Move To the Server's Community*>. In both cases, we prove that $Strenth_{t'}^{G}(C) < Strenght_{t'}^{G'}(C)$ after applying the rule.

Case 1: when the rule <*Move to the Better Community*> was applied.

By behavior rule <*Move to the Better Community*> of the *CAM-K* caste, when the rule is applied at time moment t, we have that $Strenth_t^{G}(C) < Strenght_t^{G'}(C)$. After application of the rule, i.e. at time moment $t'>t$, we have that

$Strenth_{t'}^{G}(C) = \| D_{t'}^{G}(C) \|$

$= \| \bigcup \{x.Knowledge \,|\, x \in P_{t'}^{G}(C)\} \|$

$= \| \bigcup \{x.Knowledge \,|\, x \in (P_t^{G}(C) - \{u\})\} \|$

$\leq \| \bigcup \{x.Knowledge \,|\, x \in P_t^{G}(C)\} \|$

$= Strenth_t^{G}(C)$

$< Strenght_t^{G'}(C)$

$= \| \bigcup \{x.Knowledge \,|\, x \in P_{t'}^{G'}(C)\} \|$

$\leq \| \bigcup \{x.Knowledge \,|\, x \in (P_{t'}^{G'}(C) \cup \{u\})\} \|$

$= \| D_{t'}^{G'}(C) \|$

$= Strenght_{t'}^{G'}(C)$.

Case 2: when the rule <*Move To the Server's Community*> was applied.

Similar to case 1, we can prove that after applying the rule, we have that $Strenth_{t'}^{G}(C)$

$< Strenght_t^{,G'}(C)$.

Therefore, the statement is true.

Note that the function *Strength* is upper bounded by $||D^*(C)||$ according to Lemma 3-2. Therefore, for each member agent, it can only change community for a finite number of times. Because in a closed world, the system can only have a finite number of agents, the whole system can only have a finite number of reconfigurations, which are caused by member agents changing community. Thus, the scenario transitions have finiteness property.

We now prove that the transitions can only terminate in the scenario Optimal. Suppose that community *G* is not complete with respect to *C* and there is $u \in P^G(C)$ and *u* is of category *C*. Let $tp \in D^*(C) - D^G(C)$. Eventually, there will be a member *u* in community *G* raising a question on topic *tp*. According to the behaviour rules, there will be a member, which is either *u* or a member *v* in another community *G*' that answers the question, move to the community *G* or *G*'. As proved above, the stronger community will increase its strength. Because the maximum value of $Strength_t^G(C)=||D^*(C)||$, every member of category *C* will eventually move to a community *G* that $Strength_t^G(C)= ||D^*(C)||$, i.e. the community *G* is complete by Definition 3-4. □

By Lemma 4-1 and Lemma 4-3, we have the following convergence and stability theorem of closed worlds of CAM-K organized communities.

Theorem 4-1.

In a closed world, we have that ***CAM-K*** $\vdash\!\!\!\!\rightarrow$ *Optimal*.

Proof. By Lemma 3-5 and Lemma 4-3, the statement follows Lemma 2-1. (a) immediately. □

Corollary. In a closed world, we have that ***CAM-K*** $\vdash\!\!\!\!\rightarrow$ *Mature*.

Proof. By Lemma 3-4, we have that *Optimal* $\Rightarrow$ *Mature*. The statement directly follows Theorem 4-1 and Lemma 2-2. . □

For ***CAM-P*** systems, unfortunately, the Mature scenario is not stable.

Lemma 4-4.

For **CAM-P** systems, the Mature state is not always stable.

Proof. Let community *G* be complete on category *C* and *G'* be incomplete on *C* but contain more members of category *C*. When a member of *G*' raises a query to be answered by a member *A* of *G*, agent *A* will move to community *G*' and breaks the completeness of *G*. Thus, the system becomes not mature. □

However, the Optimal state is still always reachable and stable for **CAM-P** systems.

Theorem 4-2.

In a close world, a **CAM-P** system will always converge to the *Optimal* scenario**.**

Proof.

The proof of the theorem is similar to the proof of the Lemma 4-3. The only difference is that we replace the function $Strength^C(G)= \| D_t^G(C) \|$ defined in Eq.(0.1) with the function $Strength^C(G)= \|P_t^G(C)\|$. The system will always reach state *Optimal*, because $u \in P^G(C) \rightarrow u \in P^{G'}(C)$ implies that $\|P_t^G(C)\| < \|P_{t'}^{G'}(C)\|$. The function $\|P_t^G(C)\|$ is also finitely upper bounded. Therefore, Optimal is always reachable.

Moreover, by Lemma 3-5, the Optimal scenario is stable. Therefore, by Lemma 2-1. , the theorem is true. □

Similar to the Corollary of Theorem 4-1, we have that CAM-P will always converge to the Mature scenario although in general the scenario is not stable.

4.2. World of agents attracted by individual strengths

In the CAM worlds of organized communities, members change their communities driven by the motivation of joining a stronger and better community. A variant of this model is that the members change their community because of being attracted by a particular member of another community. Such models can be formally defined by modifying the behavior rules of <*Move to a Better Community*> and <*Move to the Server's Community*>. The following specifies the caste PAM-R whose members decide whether to move according to the service provider's knowledge.

```
CASTE PAM-R <= Member;
BEGING
  <Attracted by the Buddy's knowledge>
    [$] |–> DeRegister(organiser); Register(NewOrg); organiser:=NewOrg;
          if organiser:[Introduce(Self, Buddy, cat, tp)] OR
             organiser:[Introduce(Buddy, Self, cat, tp)];
          where ((Buddy.organiser ≠ Self.organiser)
                 & (NewOrg = Buddy.organiser)
                 & (Buddy.Category = Self.Category)
                 & (Strength(Buddy)>Strength(Self))
  <Move to the Buddy's community>
    [$] |–> DeRegister(organiser); Register(NewOrg); organiser:=NewOrg;
          if organiser:[Introduce(Self, Buddy, cat, tp)];
          where ((Buddy.organiser ≠ Self.organiser)
                 & (NewOrg = Buddy.organiser)
                 & (Buddy.Category = Self.Category)
                 & (Strength(Buddy)=Strength(Self))
END;
```

where the function *Strength*(x) is defined as $Strength(x) = \|x.Knowledge\|$. By doing so, the strength of personal attraction is determined by the knowledge of the member. In comparison with the following caste PAM-E, this criterion for moving to a community

has some rationale, hence the name of the caste PAM-R.

An alternative definition of personal attractive strength is to introduce an integer valued attribute of the member to denote its strength. The value assigned to the attribute can be independent of other attributes. The structure and behavior rules of such members are formally defined as follows.

```
CASTE PAM-E<=Member;
  VAR AttractiveStrength: INTEGER;
BEGIN
  <Attracted by the Buddy's Beauty>
    [$] |–>  DeRegister(organiser); Register(NewOrg); organiser:=NewOrg;
     if (organiser:[Introduce(Self,Buddy,cat,tp)]
      OR organiser:[Introduce(Buddy,Self,cat,tp)])
         & (Buddy∈EmotionalMember);
     where ((Buddy.organiser ≠ Self.organiser)
            & (NewOrg = Buddy.organiser)
            & (Buddy.Category = Self.Category)
            &(Buddy.AttractiveStrength > Self.AttractiveStrength))
  <Move to the Buddy's community>
    [$] |–>  DeRegister(organiser); Register(NewOrg); organiser:=NewOrg;
     If (organiser:[Introduce(Self, Buddy, cat, tp)])
         & (Buddy∈EmotionalMember);
     where ((Buddy.organiser ≠ Self.organiser)
            & (NewOrg = Buddy.organiser)
            & (Buddy.Category = Self.Category)
            & (Buddy.AttractiveStrength=Self.AttractiveStrength))
END.
```

For both PAM-R and PAM-E worlds of self-organizing communities, it is possible that the system does not converge to the optimal scenario. The following is a situation of PAM-R systems where the optimal scenario is not reachable.

Example 1. (Three Gurus)

Let *A*, *B* and *C*∈PAM-R be agents shown in Table 4-1.

Note that, in this system, the domain of knowledge of category 1 is $D^*(1) = \{1, 2, 3\}$. Therefore, agents *A* and *B* will only raise questions about topic 3. Only agent *C* can answer this. When agent *C* is in the same community of agent *A*, agent *A*'s question will be answered by *C* without causing reconfiguration of the communities. If *C* is not in the same community of agent *A*, agent *C* will move to the community of agent *A* since *A*'s attractive strength is greater than *C*. The same is true for agent *B*. Therefore, if agents *A* and *B* are initially registered with different organizers, agent *C* will keep moving between two communities, while agents *A* and *B* will not change their registration at all. Table 4-2 shows a concrete example of a sequence of events and the reconfigurations of the communities to illustrate the above analysis.

TABLE 4-1. THE THREE GURUS AND THE INITIAL STATE

Agent	Category	Knowledge	Initial organiser	Strength= ‖Knowledge‖	Domain $D^*(1)$
A	1	{1, 2}	*A.organiser = a*	2	{1, 2, 3}

B	1	{1, 2}	*B.organiser = b*	2	{1, 2, 3}
C	1	{3}	*C.organiser = c*	1	{1, 2, 3}

In Table 4-2, column 1 is a sequence of events. Column 2 and 3 show the corresponding effects on the configuration of the communities and the states of the system after the events, respectively, where *A@a* stands for that agent *A* is registered with organizer *a*. □

TABLE 4-2. EXAMPLE SEQUENCE OF RECONFIGURATIONS

Event	Consequence	State after the event
A: *raise*(orgA,1, 3)	*C serves; C move to A's community*	*A,C@a, B@b,*
B: *raise*(orgB,1, 3)	*C serves; C move to B's community*	*A@a, B,C@b*
C: *raise*(orgC,1, 1)	*B serves*	*A@a, B,C@b*
A: *raise*(orgA,1, 3)	*C serves; C move to A's community*	*A,C@a, B@b*
B: *raise*(orgB,1, 3)	*C serves; C move to B's community*	*A@a, B,C@b*
C: *raise*(orgC,1, 2)	*B serves*	*A@a, B,C@b*
…		…

Theorem 4-3. (Non-convergence properties of PAM-R)

There exists a PAM-R system ***M*** of agent communities such that

(a) ***M*** does not always reach the Optimal scenario. Formally, ***M***$\rightarrowtail$*Optimal* is not true.

(b) ***M*** does not converge to the Optimal scenario. Formally, ***M*** $\not\rightarrowtail$*Optimal* is not true.

(c) The scenario Mature is not stable in ***M***. Formally, ***M****@Mature* is not true.

(d) ***M*** does not converge to the Mature scenario. ***M*** $\not\rightarrowtail$*Mature* is not true.

Proof.

(a) As shown in Example 1, the Three Gurus system cannot reach the Optimal scenario.

(b) The Three Gurus system does not converge to the Optimal scenario. If the system Three Gurus always converges to the Optimal scenario, by Lemma 2-1. (b) it always reaches the Optimal scenario. This contradicts (a). Therefore, statement (b) is true.

(c) To prove this statement, add agent *C*' into the Three Gurus system, where *C'.Knowledge* ={4}. Similar to agent *C*, *C'* will keep moving between *A*'s

community and *B*'s community. When *C* and *C'* are in the same community of either agent *A* or agent *B*, the system is in the Mature scenario because, say, *A.Knowledge* $\cup$ *C.Knowledge* $\cup$ *C'.Knowledge* is complete. However, agent *C* and *C'* will move to the other community when agent *B* raises a query on topic 3 or 4. The system will then be in a scenario that is not Mature since none of the communities are complete with respect to category 1. Therefore, scenario Mature is not stable in this system.

(d) The system constructed in the proof of (c) above gives a counter-example of convergence.

□

It is worth noting that, according to Lemma 2-1, the reachability to a scenario *Sc* is a necessary but not sufficient condition of the convergence to *Sc*. Therefore, in Theorem 4-3, statement (b) is a consequence of statement (a), but not visa versa. The stability of a scenario *Sc* is neither a necessary nor a sufficient condition of the convergence to the scenario *Sc*. Therefore, in Theorem 4-1, statement (c) and (d) are logically independent of each other.

For PAM-R systems, the Mature scenario is reachable although it is not stable.

Lemma 4-5.

Let ***M*** be a PAM-R system of agent communities and *C* be any given non-trivial category of knowledge in ***M***. The system ***M*** will always reach a scenario that ***M*** has a community *G* that is complete with respect to category *C*. Formally, $\boldsymbol{M} \rightarrowtail \exists G.(Complete^{G}(C))$.

Proof .

To prove that a scenario *Sc* is always reachable, it is sufficient to construct a sequence of actions that will always lead to the scenario *Sc*. Based on the assumption that the non-deterministic choices of actions by the agents are fair, the basic theory of probability implies that the sequence of actions will eventually happen with probability approaches 1 as the time increases. The following proves the lemma by constructing such a sequence of actions that will lead to the scenario Mature.

Let *Agents*(*C*) be the set of member agents of category *C* in the system ***M***. Assume that at time moment *t*, G_1, G_2, …, G_k are the communities that have at least one member of category *C*. We say that member *A* of category *C* in G_i is a *leader* of G_i for category *C*, if

$$\forall x \in P_t^{Gi}(C).(||x.Knowledge|| \leq ||A.Knowledge||).$$

Note that at any time moment *t*, for a community G_i that $P_t^{Gi}(C) \neq \varnothing$, there may be

many leaders of category *C*, but there will always at least one leader. Moreover, at different time moments, the leaders of a community may be different because a leader may move to another community or an agent of high volume of knowledge may join the community and becomes a new leader. In the sequel, we will discuss with respect to a given category *C*. Thus, without any risk of confusion, we omit the reference to category *C*.

Let $L_{i,t}$ be a leader of G_i at time moment *t*. Define the leadership strength of a community *G* at time moment *t*, written $LS_t(G)$, to be the volume of knowledge held by a leader. Formally,

$$LS_t(G) = ||L_{G,t}.Knowledge||,$$

where $L_{G,t}$ is a leader of *G* at time moment *t*.

We define that, at time moment *t*, a community *G* is the *strongest community* with respect to its leadership strength is the community such that for all communities *G'*, $LS_t(G) \geq LS_t(G')$.

Note that the strongest community with respect to its leadership strength may be not unique. The following proof will proceed in two cases. The first case is when the strongest community is unique. In this case we proof that the strongest community can always reach completeness. The second case is when the strongest community is not unique, i.e. when there are two or more strongest communities. We proof that the number of strongest communities can always be reduced until there is only one strongest community, or one of the strongest community will reach the completeness scenario in the attempt to reduce the number of strongest communities.

Case 1: *when the strongest community is unique*.

Let *G* be the strongest community and let *L* be a leader of *G*. By definition of strongest community, we have that all other community's leadership is weaker than *G*'s leader. Therefore, for all agents *A* of category *C* in any other community *G*', we have that

$$||L.Knowledge|| > ||A.Knowledge||. \qquad (*)$$

Let *q* be any topic of category *C* that is not in the domain of knowledge in community *G*, i.e. $q \notin D_t^G(C)$ and $q \in D_t^*(C)$. When *L* makes a query on *q*, there must be an agent *A* from another community *G'* that answers the query. Because of (*), according to the <*Attracted by the Buddy's knowledge* > rule, agent *A* will join community *G* after answering query *q*.

Therefore, after *L* makes queries on all topics that are not known by agents in its community, *G* becomes complete on category *C*. Therefore, the system will always reach the mature scenario.

Case 2: *when the strongest community is not unique*.

Let *G* be any one of the strongest communities and *L* be a leader of *G*. The same as in the proof of Case 1, *L* will make queries on topics *q* that are unknown by members in community *G*.

If a query on topic *q* is answered by an agent *A* that has lower volume of knowledge than *L*, agent *A* will joint community *G* according to the rule <*Attracted by the Buddy's knowledge* >. If the query is answered by an agent *A* that has the same volume of knowledge as *L*, *L* will join *A*'s community according to the rule <*Move to the Buddy's community*>. Then, a new leader *L'* for community *G* will be selected.

If the volume of knowledge held by the new leader *L'* is less than *L*, *G* will no longer be a strongest community. Thus, the number of strongest community will be reduced by one. If the volume of knowledge held by *L'* is equal to that of the old leader *L*, G is still a strongest community, but the number of *G*'s leaders reduces by one. The new leader *L'* then carries on make queries until either there is no more knowledge that is unknown by community *G* or the leader is changed to someone that has less volume of knowledge. In the former situation, we reach a complete community, thus the statement of the lemma is true. In the later case, the number of strongest community reduces by one.

If the strongest community in the system is still not unique, selecting one of the strongest communities and repeating the process above will further reduce the number of strongest community until there is only one strongest community. Thus, eventually Case 1 applies and then the strongest community will reach completeness.

It is worth noting that in the above proof when a leader *L* of a strongest community moves to another community, the total number of strongest communities in the system will not increase. According to the rule <*Move to the Buddy's community*>, the agent *A* that answers *L*'s query must have the same volume of knowledge as the *L*. By the definition of leaders, the leader of *A*'s community must have at least the same volume of knowledge as *L*. Thus, the leader of *A*'s community must have at least the same volume of knowledge as *L*. Therefore, since *L* is a leader of a strongest community, *A*'s community must also be a strongest community already.

Therefore, the lemma is true.

□

Theorem 4-4. *(Reachability of Mature Scenario in PAM-R systems)*

For all PAM-R systems ***M*** of agent communities, scenario Mature is always reachable. Formally, we have that ***M***$\rightarrowtail$*Mature.*

Proof. The theorem follows directly from Lemma 4-5 and the fact that for all categories C and C' that $C \neq C$', the actions taken by agents of category C' will not affect the completeness of communities with respect to category C. □

It is worth noting that PAM-R is a special case of PAM-E when the following condition holds.

$$\forall X \in PAM\text{-}E.(X.AttractiveStrength = ||X.Knowledge||)$$

Therefore, the non-convergence property proved in Theorem 4-3 is also true for PAM-E systems in general. Moreover, the none-convergence property is also true even if the attractive strengths are inconsistent with the amount of knowledge possessed by the agents.

Definition 4-1. (*Consistency between attractiveness and knowledge*)

In a PAM-E system, we say that the attractive strength is *consistent* with the volume of knowledge, if for all $A, B \in Member$ caste, we have that

$$(A.AttractiveStrength > B.AttractiveStrength) \Leftrightarrow (||A.Knowledge|| > ||B.Knowledge||).$$

Otherwise, we say that the attraction strength is *irrelevant* to the volume of knowledge. □

The PAM-E system given in Table 4-3 is a system that has irrelevant attraction strengths. It is a counterexample of the convergence to the Optimal scenario. A similar counterexample of the stability of the Mature scenario can be constructed as in the proof of Theorem 4-3(c).

Theorem 4-5. (*Non-convergence properties of PAM-E***)**

There exists a PAM-E agent community system $\boldsymbol{M}$ whose agents' attraction strengths are irrelevant to the volume of knowledge such that the following statements are true.

(a) $\boldsymbol{M}$ does not always reach the scenario Optimal. Formally, $\boldsymbol{M} \rightarrow\!\!\!\!\!\rightarrow Optimal$ is not true.

(b) $\boldsymbol{M}$ does not converge to the Optimal scenario. Formally, $\boldsymbol{M} \not\rightarrow Optimal$ is not true.

(c) The scenario Mature is not stable in $\boldsymbol{M}$. Formally, $\boldsymbol{M}@Mature$ is not true.

(d) $\boldsymbol{M}$ does not converge to the Mature scenario. Formally, $\boldsymbol{M} \not\rightarrow Mature$ is not true.

Proof. The proof is similar to the proof of Theorem 4-3 with the Three Gurus example replaced by the system given in Table 4-3. Details are omitted for the sake of space. □

TABLE 4-3. THREE GURUS OF PAM-E

Agent	*Category*	*Knowledge*	*Initial organiser*	*Attractive Strength*	||Knowledge||
A	1	{1}	*A.organiser = a*	2	1

B	1	{1}	*B.organiser = b*	2	1
C	1	{2, 3}	*C.organiser = c*	1	2

Similar to PAM-R system's reachability of Mature scenario, we can prove that PAM-E systems are always reachable to the Mature scenario.

Theorem 4-6. (Reachability of Mature Scenario in PAM-E systems)

For all PAM-R systems ***M*** of agent communities, scenario Mature is always reachable. Formally, we have that ***M***$\rightarrowtail$*Mature.*

Proof. The proof is very similar to the proofs of Lemma 4-5 and Theorem 4-6. The only change is to replace the definition of leadership strength by the following formula $LS_t(G) = ||L_{G,t}.AttractionStrength||$, and to replace the phrase 'volume of knowledge' by 'attraction strength'. □

4.3. Hybrid worlds

In this subsection, we study the worlds of organized communities that contain more than one type of members.

In order to enable *PAM-E* agents to compare their attractive strengths with agents of other types, we modify the definition of CAM-K, CAM-P and PAM-R agents by adding an Integer type constant state attribute *AttractiveStrength*.

The following theorems state that hybrid systems have the same convergence properties as PAM-R and PAM-E systems if there is at least on of such members.

Note that, as proved in Lemma 3-5, The Optimal scenario is stable for all kinds of member agents. Therefore, it is still true for hybrid systems. The following theorem proves the none-convergence properties of hybrid systems.

Theorem 4-7. (None-convergence properties of hybrid systems)

There exists a hybrid system ***M*** of agent communities that contains one PAM-R or PAM-E agent such that the following statements are true.

(a) ***M*** does not always reach the scenario Optimal. Formally, ***M***$\rightarrowtail$*Optimal* is not true.

(b) ***M*** does not converge to the Optimal scenario. Formally, ***M*** $\not\hookrightarrow$*Optimal* is not true.

(c) In ***M***, the scenario Mature is not stable. Formally, ***M***@*Mature* is not true.

(d) ***M*** does not converge to the Mature scenario. Formally, ***M*** $\not\hookrightarrow$*Mature* is not true.

Proof.

(a) Let's construct a counterexample of the reachability. Consider the Three Gurus

system of Example 1. By replacing the caste of agent *C* in the system with *CAM*, we obtain a proper hybrid system. In this system, once *C* joins a community of agent *A* or *B*, it will not change its membership to the community because *C* is attracted to its community which is complete. However, both agents A and B will not change their community memberships as proved in Example 1. The system will keep in this state rather than further evolves into the optimal scenario. Therefore, the system cannot reach the optimal scenario.

(b) By Lemma 2-1. and statement (a), such a hybrid system may not converge to the Optimal scenario.

(c) To prove that the scenario Mature is not stable, add the same agent *C*' in the proof of Theorem 4-3(b) to the counterexample constructed in the proof of (a) above. Assume that agents *A*, *C* and *C*' are registered with the same organiser, they form a complete community. The system is therefore in the mature scenario. However, when agent *B* raise a query on topic 4, *C*' will move to the community of agent *B*. the agents *A*, *B*, *C* and *C*' split into two communities {*A*, *C*} and {*B*, *C*'}. None of these communities are complete. Hence, the system is no longer in the scenario mature. Therefore, scenario mature is not stable.

(d) The counterexample given in the proof of (c) is also a counterexample for the convergence. □

There are some phenomena that may happen in a hybrid system but impossible in any of the homogenous systems. For example, in a hybrid system, it is possible that two agents swap their communities. That is, it is possible that after inter-community communication between agents *A* of community *G* and *B* of community *H*, agent *A* moves to the community *H* and *B* moves to community *H*. This is impossible in homogenous systems because of the design of the rules. The following is a concrete example of community swap.

Example 2. (*Swap communities***)**

Let agent *A* be a *CAM-K* and *B* be a *PAM-E* with the initial state given in Table 4-4.

TABLE 4-4. DEFINITION OF A HYBRID SYSTEM

Agent	*Category*	*Knowledge*	*Initial organiser*	*Attractive Strength*	‖Knowledge‖
A	1	{1, 2}	*A.organiser = G*	2	2
B	1	{3, 4, 5}	*B.organiser = H*	1	3

When agent *A* raises a question on topic 3, *B* answers the question. Then, *A* as a *CAM*

agent decides to move to community *H* since $D^H(1)=3>D^G(1)=2$. Meanwhile, agent *B* as a *PAM-E* agent decides to move to community *G* because

$$AttractiveStrength(A) = ||A.Knowledge|| = 2 > B.AttractiveStrength = 1.$$

Therefore, it leads to the state where *A* is registered with organizer *H* and *B* registered with organizer *G*. The system will carry on swapping communities. Agents *A* and *B* will never get into the same community. □

The above example gives another counterexample of hybrid systems' reachability to the Mature and Optimal scenarios. Thus, we have the following theorem.

Theorem 4-8. (*Non-reachability of hybrid system*)

There exists a hybrid system that does not reach the Mature and Optimal scenarios.

Proof. The system given in Example 2 does not reach Maturity and Optimal scenarios in all executions. □

Now, we have proved or disproved all the recurrence properties of the Mature and Optimal scenarios for all five types of agent community systems. The results are summarized below in Table 4-5.

TABLE 4-5. RECURRENCE PROPERTIES OF AGENT COMMUNITY FORMATION BEHAVIORS

Variant	Maturity			Optimality		
	Reachability	Stability	Convergence	Reachability	Stability	Convergence
CAM-K	Yes	Yes	Yes	Yes	Yes	Yes
CAM-P	Yes	No	Yes	Yes	Yes	Yes
PAM-R	Yes	No	No	No	Yes	No
PAM-E	Yes	No	No	No	Yes	No
Hybrid	No	No	No	No	Yes	No

5. Experimental Study of Convergence Speed

To further validate the results obtained above and to study the dynamic behaviours of agent communities quantitatively, an experimental study through simulations was also conducted. The preliminary results of the experiment have been reported in [48]. This section reports the further experiment results.

5.1. Experiment environment

To enable the experiments, we developed an experiment environment to simulate the execution of agent communities. The tool also provides a graphical user interface to setup simulation experiments and to collect data for statistical analysis. **Figure 5-1** is the

snapshot of its GUI interface.

Figure 5-1. Interface of the experiment environment

As shown in Figure 5-1, the tool takes four parameters as the input to the agent community simulation:

- **k:** the number of organizers,
- **m:** the total number of member agents in the system,
- **c:** the number of categories of knowledge, and
- **s:** the size of the knowledge for each category.

For each given set of parameters, an initial setting of the agent communities is generated at random according to the uniform distribution. Thus, each agent is initially assigned at random with a category, a non-empty set of topics and an organizer. An initial setting is called *non-trivial* if every organizer has at least one member and every category has at least one member. The trivial settings are avoided in random generation of initial settings, if possible. Figure 5-2 is a screen snapshot showing on the left an initial setting and on the right the setting after several iterations in an execution of the same system.

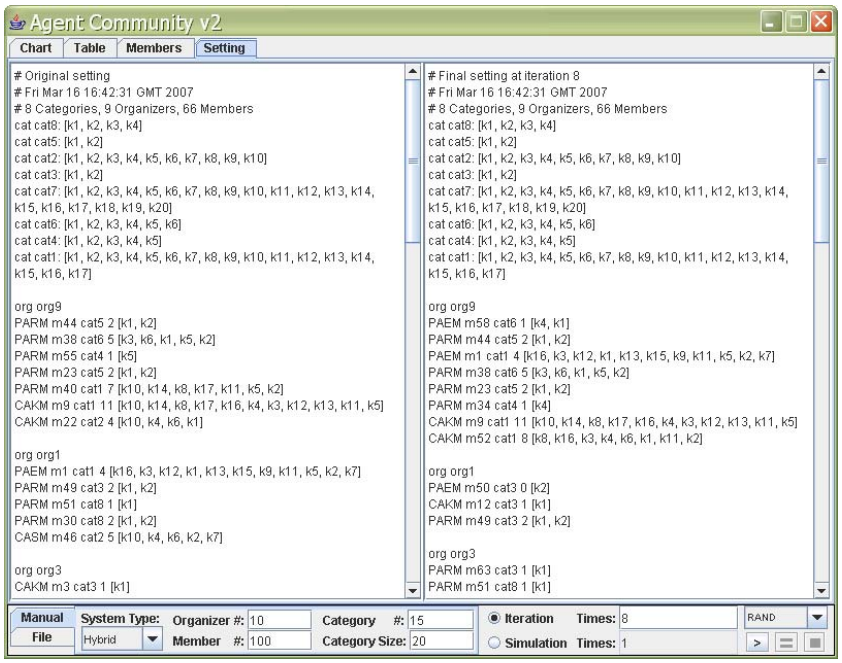


Figure 5-2. Example of initial setting and final setting

The user can select one of the five types of agent community systems to perform an experiment. Two basic types of experiments are supported by the tool. The first is to run a system for once and to collect the data of the execution and display them in various windows. The second is to repeat the execution on an initial setting for a number of times and to collect the data, display the data in a graphic user interface and perform statistical analysis of the data.

In each execution, the algorithm is run for a certain number of iterations determined by the user. In each iteration, there is a random number of members to raise questions, get their answers if exist, and then to make movement according to the members' behavior rules. Each member only raises one question in one iteration cycle. After each iteration cycle, the system's global state is checked to see if it matches of the scenarios Mature or Optimal. The tool also allows the user to set the number of iterations in each execution and the number of executions to be repeated on each initial setting. **Figure 5-1** is a screen snapshot that displays the statistical data in the middle of an execution for repeating simulation for 100 times.

5.2. Experimental validation of theoretical results

To validate the theoretical results presented in section 4, simulation experiments were conduected for each of the five types of agent community systems. For each type of the systems, 1000 initial settings were generated at random with parameters in the range

shown in Table 5-1. The system was run repeatedly for 100 times on each initial setting, where each execution iterates for 500 cycles.

TABLE 5-1. PARAMETER RANGES IN THE SIMULATIONS

Parameter	**Range**
k: the number of organizers	2≤k≤100
m: the total number of member agents in the system	2≤m≤100
c: the number of categories of knowledge	1≤c≤30
s: the size of each category of knowledge	2≤s≤30

The experiments confirmed the theoretical results as follows.

- For a scenario that is theoretically always reachable in a type of agent community systems, it is confirmed by experiments if and only if in all runs of the agent community system on every initial setting the system is in the scenario after a number of iterations.
- For a scenario that is theoretically not always reachable, the property is confirmed if and only if there is an initial setting in which there is at least one run of the agent community system that does not reach the scenario in all iterations within the set number of iterations.
- For a scenario that is theoretically stable, the property is confirmed by the experiments if and only if, for all initial settings and every execution of the agent community system in the setting, the system reaches the scenario after a certain number of iterations implies that the system is in the scenario for all iterations after that in the same execution.
- For a scenario that is theoretically not stable, the experiments confirm the property if and only if there is at least one initial setting and at least one execution of the agent community system in the setting such that the system reaches the scenario at a certain iteration and breaks the scenario at an iteration afterwards in the same execution.

It is worth noting that, by systematically conducting repeated experiments using a large number of randomly generated initial settings and a large number of random executions of the system in each initial setting with a large number of iterations, we can gain a high confidence in the results. However, the experimental results are not as conclusive as the theoretical proofs for reachability, unreachability, and stability. Only non-stability can be conclusively confirmed by observing a witness of the non-stability.

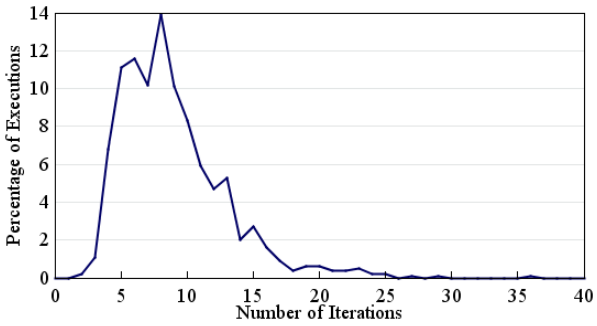
Thus, simulation experiments cannot replace formal proofs.

5.3. Quantitative study of convergence speed

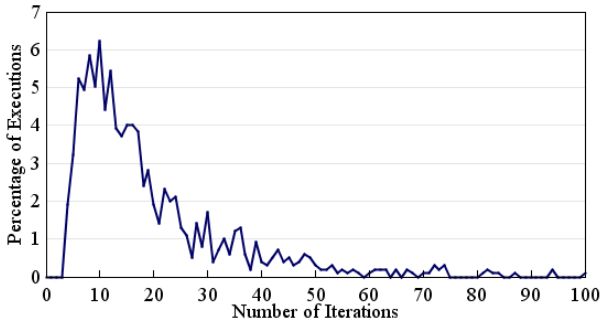
However, simulation experiments do have their advantages in discovering phenomena that were unknown. For example, the simulation experiments with CAM-K systems revealed a number of interesting relations between the parameters of agent communities and the speed for a community to reach a desired scenario. This section reports the main findings of the experiments.

B. Distributions of convergence time

A question raised in the experiments is when to stop the execution of an agent community. A practical approach is to set a number of iterations the agent community will execute. Thus, an experiment is carried out aiming at discovering how fast a CAM-K agent community system converges to a scenario. The number of iterations that a system needs to reach a scenario is called the *convergence time* in the sequel.



(a) Average convergence time to reach the Mature scenario



(b) Average convergence time to reach the Optimal scenario

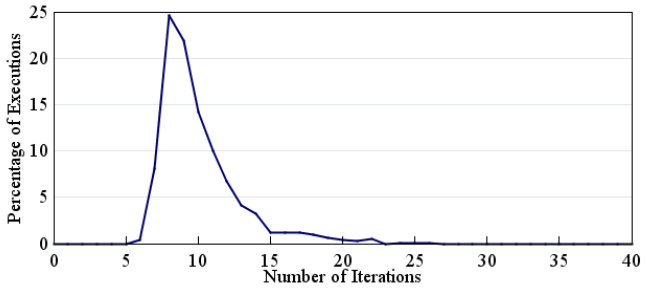
Figure 5-3. Distributions of convergence times on the same initial setting

(Parameters: k=50, m=50, c=20, s=30)

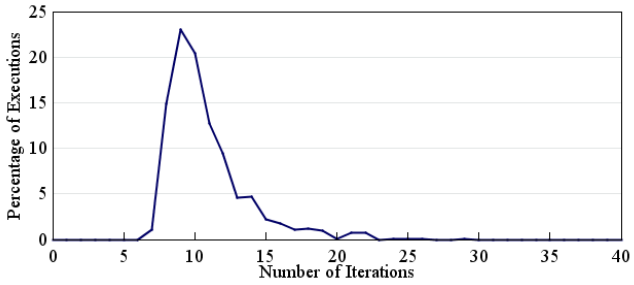
The first experiment is on fixed initial settings generated at random with repeated executions for 1000 times. This experiment investigates the impact of random behaviours

of agents on the convergence time. Figure 5-3 shows the distributions of convergence times for the Mature and Optimal scenarios on fixed initial settings with parameters k=50, m=50, c=20, and s=30, respectively, where the *x* axis is the number of iterations the system needs to converge; the *y* axis is the percentage of the number of executions in which the system converges to the scenario in the number of iterations.

The second experiment is on different initial settings generated at random with the same fixed parameters. This experiment investigates the impact of initial settings on convergence time. Figure 5-4 shows the distribution of average convergence times on each initial settings that were generated with parameters k=50, m=50, c=20, and s=30, where the *x* axis represents the average iterations in which a system converges to the scenario, and the *y* axis is the same as in Figure 5-3. The same pattern of the distributions of the average convergence times were observed as shown in Figure 5-4.



(a) Average convergence time to reach the Mature scenario



(b) Average convergence time to reach the Optimal scenario

Figure 5-4. Distribution of average convergence times on variable initial settings

(Parameters: k=50, m=50, c=20, s=30)

The overall average convergence times on fixed and variable initial settings with the same parameters k=50, m=50, c=20 and s=30 is presented in Table 5-2.

TABLE 5-2 AVERAGE CONVERGENCE TIMES

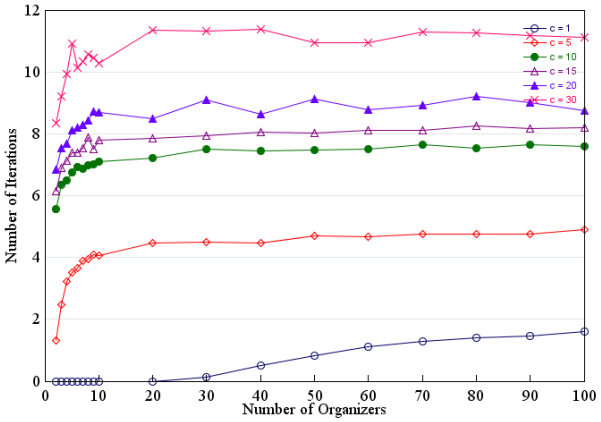
	Maturity		Optimality	
Experiment	Average Value	Standard Deviation	Average Value	Standard Deviation
Fixed initial setting	8.90	4.03	19.11	15.01
Variable initial settings	9.85	3.10	10.54	3.28

From the experiment results, we conclude that with very high probability an agent community will converge within 100 iterations, if it will converge.

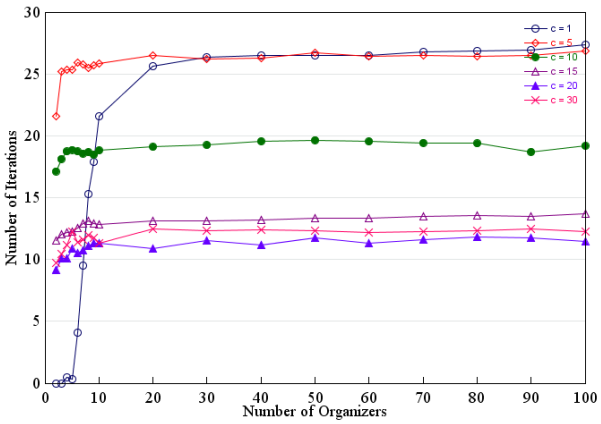
To explore the relationships between convergence time and various parameters of agent communities, we carried out further experiments with CAK-M systems. The results are reported below.

C. The effect of number of organizers

In the experiment that aims at understanding the effect of the number of organizers on convergence time, we fixed the parameters m (the number of members), c (the number of categories) and s (the size of each category). The parameter k (number of organizers) varied from 2 to m, because when k>m, there must be organizers associated with no members, which has no effect on the operation of the system, and thus the initial setting is trivial. For each value of k, 100 initial settings were generated at random with the uniform distribution, and on each initial setting the agent community system was executed repeatedly for 100 times. The average convergence time was then calculated. The experiment was carried out for several different the number of categories, i.e. c=1, 5, 10, 15, 20, and 30.



(a) Average convergence time to reach the Mature scenario



(b) Average convergence time to reach the Optimal scenario

Figure 5-5. Impact of the number k of organizers on convergence time

(Parameters: m=100, c=1,5,10,20,30, s=30, k varies from 2 to m=100)

The results are shown in Figure 5-5 shows how the average convergence time depends on the number of organizers when the parameters are m=100, s=30, and c=1, 5, 10, 15, 20 and 30, respectively. In general, when the number of organizers increases, the agents in the initial setting are spread to more and more communities. Thus, it takes longer to form mature communities and to reach optimal scenarios. This pattern is observed for all different numbers of categories.

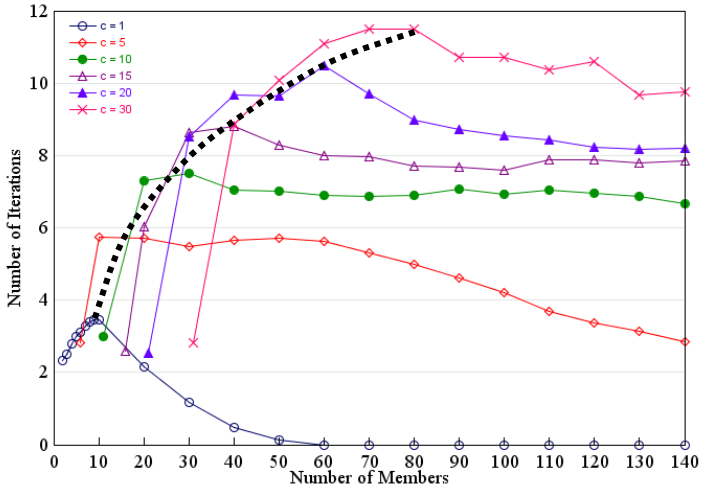
D. The effect of number of members

To study the effect of the number of members on convergence time, we fixed the parameters of k (number of organizers), c (the number of categories) and s (the size of each category) and let the parameter m (the number of members) vary.

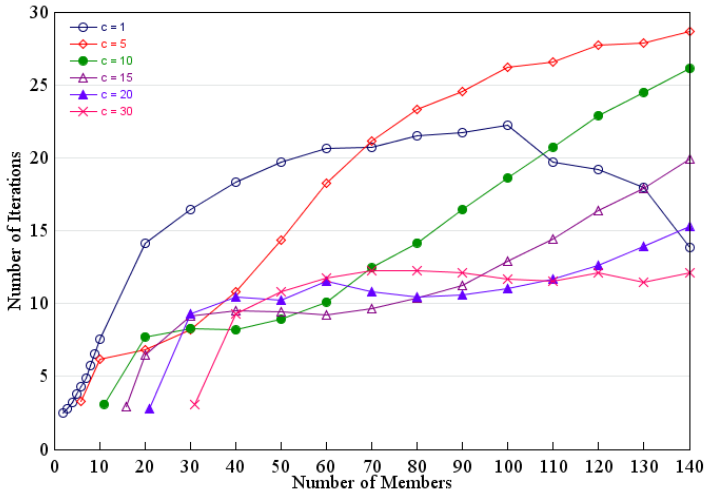
The particular parameters used in the experiments were: k=100 organizers and the size of each category s=30. This experiment is also repeated for several different numbers of categories, i.e. for c = 1, 5, 10, 15, 20 and 30. The number of iterations set in the experiments was 500. Similar to the experiment reported in sub-section C, 100 initial settings were generated at random and on each initial setting the CAKM agent community system was executed for 100 times and average convergence time were calculated.

As noted above, an organizer that is initially assigned with no member has no effect on the operation of the system. In order to minimize the effect of empty initial organizers on statistical results, two strategies were used in the random generation of the initial settings according to the value of m. When m>k, at least one member is initially assigned to each organizer. When m≤k, each organizer was initially assigned with at most one

member.



(a) Average convergence time to reach the Mature scenario



(b) Average convergence time to reach the Optimal scenario

Figure 5-6. Impact of the number m of members on convergence time

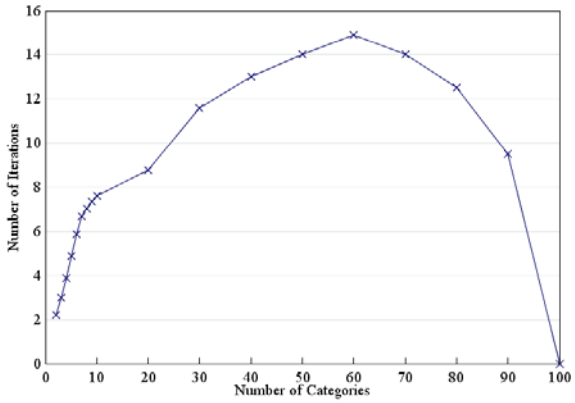
(Parameters: k=10, c=1, 5, 10, 15, 20, 30, and s=30, m varies from 1 to 140).

Figure 5-6 shows how the average convergence time depends on the numbers of members in the system when k=10, s=30, and c=1, 5, 10, 15, 20, and 30. In general, the experiments demonstrated that, for both the Mature and Optimal scenarios, the convergence time first increases then decreases as the number m of members increases. The turning point of a curve on which it changes from increase to decrease depends on the number of categories. As shown by the dashed line in Figure 5-6, the larger the value of c, the later the turning point.

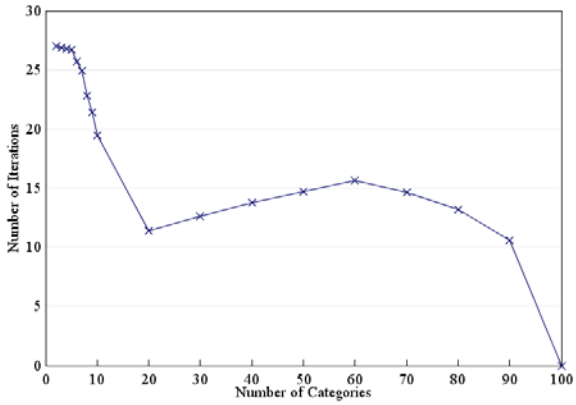
E. The effect of number of categories

In the experiments to investigate the effect of the number of categories on convergence speed, we varied the number c of categories while kept the other parameters fixed. Note that, when c>m, there must be a category of knowledge that no agent is interested in. Such categories have no effect on the operation of the system. Thus, in the experiment, the variable c varied in the range of 1<c<m.

The experiment shows that, to reach the Mature scenario, with the number c of categories increasing, the convergence time first increases, and then decreases. In contrast, for the Optimal scenario, that with the number c of categories increasing, the convergence time decreases sharply, then increases gently, and at last decreases sharply again. Figure 5-7 shows how the average convergence time depends on the number of categories when k=100, m=100 and s=30, respectively.



(a) Average convergence time to reach the Mature scenario



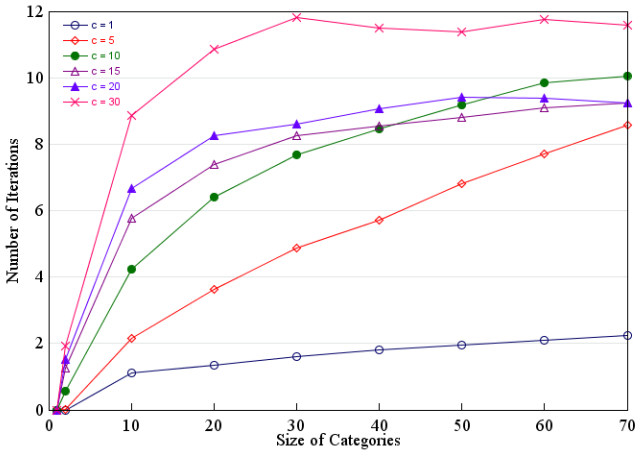
(b) Average convergence time to reach the Optimal scenario

Figure 5-7. Impact of the number of categories on average convergence time

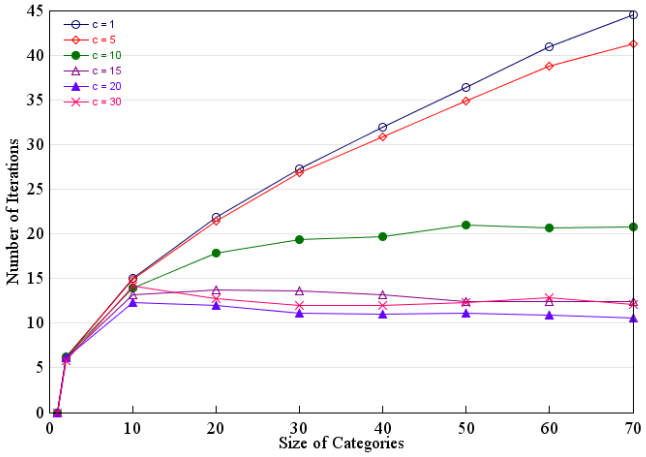
(Parameters: k=100, m=100 and s=30, c varies from 1 to 100)

F. The effect of the size of categories

In the experiments about the size of categories, we varied the size of categories s while kept other parameters fixed. This is again repeated for several different values of c. The experiment demonstrated that the average convergence time for both Mature and Optimal scenarios increases with the size of the category increasing although the speed is different. Figure 5-8 shows how the average convergence time depends on the category size when k=100, m=100, and c=1, 5, 10, 15, 20, 30, respectively.



(a) Average convergence time to reach the Mature scenario



(b) Average convergence time to reach the Optimal scenario

Figure 5-8. Impact of category size on convergence time

(Parameters: k=100, m=100, c=1, 5, 10, 15, 20, 30, s varies from 1 to 70)

The main findings of the experiments discussed above are summarized in Table 5-3, where ↑ indicates that the convergence time increases with the variable, and ↓ for decreasing.

TABLE 5-3. RELATIONSHIPS BETWEEN SYSTEM PARAMETERS AND CONVERGENCE TIMES

Parameter	Condition	Convergence Time	
		Maturity	Optimality
#Organizers		↑	↑
#Members	m≤k	↑	↑
	m>k	↑ ↓	↑ ↓
#Categories		↑ ↓	↓ ↑ ↓
Size of Category		↑	↑

6. Conclusion

In this paper, we formally analyzed the properties of various types of self-organizing agent community systems. For each variant, we studied two scenarios. The Maturity scenario of community formation is the situation that for each category of knowledge, there is a community that contains all the knowledge of the category. The Optimal scenario of community formation is the situation where every agent in the system is in a community that contains all the knowledge that it is interested in. We formally proved or disproved the stability and the reachability of these scenarios in self-organizing community formation, and their convergence to such scenarios.

The properties we have proved in this paper for PAM-R, PAM-E and hybrid systems are negative. It is an interesting topic for future work to find the conditions in which a system is reachable, stable and convergent to the Mature and/or Optimal scenarios. We will also study other variants of community formation algorithms by applying the same techniques presented in this paper.

In the experimental study of convergence speed using simulation, we observed interesting, and sometime complicated, patterns of convergence time depending on the parameters of the agent community systems. Some of these patterns are difficult to explain. It is an interesting topic for future work to provide formal proofs of such patterns.

Our investigation of agent community formation demonstrated that formal methods and simulation techniques are complementary to each other. Simulation helps to discover recurring phenomena while formal methods help to prove their recurrences with certainty. We believe that this approach can be applied to all studies of emergent behaviours of multi-agent systems.

ACKNOWLEDGEMENT

The work reported in this paper was done while Hong Zhu was on sabbatical leave from Oxford Brookes University and visiting the Pervasive ICT Research Centre of British Telecom at Ipswich.

REFERENCES

[1] United Nations Joint Programme on Aids. 1997. *Community mobilization and AIDS: UNAIDS Technical Update*, Geneva: UNAIDS.

[2] Akram, A. and Rana, O. F. 2003. Structuring peer-2-peer communities. *Proc. of IEEE International Conference on Peer-2-Peer Computing*, Linkoping, Sweden.

[3] Wang, Y. and Vassileva, J. 2004. Trust-based community formation in peer-to-peer file sharing networks. *Proc. of 2004 IEEE/WIC/ACM International Conference on Web Intelligence* (WI 2004), Beijing, China, pp341-348.

[4] Duda, R.O. and Hart. P. E. 1973. *Pattern Classification and Scene Analysis*. John Wiley & Sons.

[5] Lu S. Y. and Fu, K.S. 1978. A sentence-to-sentence clustering procedure for pattern analysis. *IEEE Transactions on Systems, Man and Cybernetics,* Vol. 8, pp381-389.

[6] Jain, A.K., Murty, M.N., and Flynn, P.J. 1999. Data clustering: a review. *ACM Computing Survey, Vol.* 31, No.3, pp264-323.

[7] Charikar, M., Chekuri, C., Feder, T. and Motwani, R. 1997. Incremental clustering and dynamic information retrieval. *Proc. of the Conference on Theory of Computation*, pp626-635.

[8] Fisher, D. 1987. Knowledge acquisition via incremental conceptual clustering. *Machine Learning* 2, pp139-172.

[9] Khambatti, M., Ryu, K., and Dasgupta P. 2003. Structuring peer-to-peer networks using interest-based communities. *Proc. of International Workshop On Databases, Information Systems and Peer-to-Peer Computing*, pp48-63.

[10] Ogston, E, Overeinder, B., Van Steen M. and Brazier F. 2004. Group formation among peer-to-peer agents: learning group characteristics. *Proc. of the Second International Workshop on Agents and Peer-to-Peer Computing (AP2PC)*, Springer Lecture Notes in Computer Science Vol. 2872, pp59-70.

[11] Iamnitchi, A, Ripeanu, M and Foster I. 2004. Small-world file-sharing communities. *Proc. of INFOCOM 2004*, Hong Kong, Vol. 2, pp952- 963.

[12] Camazine, S., Deneubourg, J.-L., Franks, N.R., Sneyd, J., Theraulaz, G. and Bonabeau, E. 2001. *Self-Organization in Biological Systems*. Princeton University Press.

[13] Green, D.G. 1994. Emergent Behaviour in Biological Systems. *Complexity International,* Vol. 1.

[14] Clarke, I. and Sandberg, O. 2000. Freenet: A distributed anonymous information storage and retrieval system. *Proc. of ICSI Workshop on Design Issues in Anonymity and Unobservability*, California, USA, pp25-26.

[15] Babaoglu, O., Meling, H. and Montresor, A. 2003. Anthill: A Framework for the Development of Agent-Based Peer-to-Peer Systems. *Proc. of the 22nd International Conference on Distributed Computing Systems*, Vienna, Austria, pp15-22.

[16] Flake, G.W., Lawrence, S. and Giles, C.L. 2000. Efficient identification of web communities. *Proc. of the Sixth International Conference on Knowledge Discovery and Data Mining*, pp150-160.

[17] Wang, F. 2002. Self-organising communities formed by middle agents. *Proc. of 1st International Joint Conference on Autonomous Agents and Multi-Agent Systems* (*AAMAS'02*), Italy, pp1333-1339.

[18] Cid-Sueirro, J. and Wang, F. 2002. A scalability analysis of self-organizing agent communities. *Proc. of International Conference on Learning*, Madrid, Spain.

[19] Bryan Horling and Victor lesser. 2005. A survey of multi-agent organizational paradigms. *The knowledge Engineering Review*, Vol. 19:4, pp281-316.

[20] Virginia Dignum, (Ed.). 2009. *Handbook of Researches on Multi-Agent Systems:*

Semantics and Dynamics of Organizational Models. IGI Global.
[21] Ketchpel, S. P. 1995. Coalition formation among autonomous agents, *From Reaction to Cognition*, LNCS 957, Springer, pp. 73-88.
[22] Shechory O. and Kraus, S. 1995. Coalition formation among autonomous agents: Strategies and complexity, *From Reaction to Cognition*, LNCS 957, Springer, pp. 57-72.
[23] Shehory, O. and Kraus, S. 1998. Methods for task allocation via agent coalition formation. *Artificial Intelligence*, Vol. 101, pp. 165-200.
[24] Shehory, O. 2004. Coalition formation: towards feasible solutions. Fundamenta Informaticae, Vol. 63, pp. 107-124.
[25] Klusch, M. and Gerber, A. 2002. Dynamic Coalition formation among rational agents, IEEE Intelligent Systems, Vol. 17, pp. 42-47
[26] Vig, L. and Adams, J. A. 2007. Coalition formation: from software agents to robots. J Intell Robot Syst., Vol. 50, pp. 85–118
[27] Mason, K., Denzinger, J. and Carpendale, S. 2005. Negotiating Gestalt: Artistic expression by coalition formation between agents. LNCS 3638, Springer, pp. 103-114.
[28] Merida-Campos C. and Willmott, S. 2007. Stable Collaboration Patterns of Self-Interested Agents in Iterative Request for Proposal Coalition Formation Environments. *International Transactions on Systems Science and Applications*, Vol. 2, No. 1, pp40-45.
[29] Milind Tambe. 1997. Towards Flexible Teamwork. *Journal of Artificial Intelligence Research*, Vol. 7, pp83-124.
[30] Nair, R., Tambe, M. and Marsella, S. 2003. Role allocation and reallocation in multiagent teams: Towards a practical analysis, *Proc. of AAMAS'03*, Melbourne, Australia, pp552-559.
[31] Pynadath D. V. and Tambe, M. 2002. Multiagent teamwork: analyzing the optimal and complexity of key theories and models. *Proc. of AAMAS'02*, Bologna, Italy, pp873-880.
[32] Sycara, K., Decker, K. and Williamson, M. 1997. Middle-Agents for the Internet, *Proc. of IJCAI-97*, pp578-583.
[33] Brooks C. and Durfee, E. 2003. Congregation Formation in Multiagent systems, *Journal of Autonomous Agents and Multi-Agent Systems*, Vol. 7, pp145-170.
[34] Vidal, J. M. and Durfee, E.H. 1998. The Moving Target Function Problem in Multi-Agent Learning. *Proc. of the Third International Conference on Multiagent Systems* (ICMAS 1998), Paris, France, pp317-324.
[35] Shoham Y. and Tennenholtz, M. 1997. On the emergence of social conventions: modeling, analysis and simulation. *Artificial Intelligence*, Vol. 94, No.1-2, pp139-166.
[36] Savarimuthu B. T. R. and Purvis, M. 2007. Mechanisms for norm emergence in multi-agent societies. *Proc. of AAMAS'07*, pp1104–1106, 2007.
[37] Mukherjee, P., Sen S., and Airiau, S. 2008. Norm emergence under constrained interactions in diverse societies, *Proc. of AAMAS'08*, Estoril, Portugal, Vol. 2, pp779-786.
[38] Zhu, H. 2000. Formal Specification of Agent Behaviour through Environment Scenarios. *Formal Approaches to Agent-Based Systems*: *Proc. of First Goddard Workshop on FAABS 2000,* NASA Goddard Space Flight Center, Springer LNCS 1871, pp263-277.
[39] Zhu, H. 2001. SLABS: A Formal Specification Language for Agent-Based Systems. *International Journal of Software Engineering and Knowledge Engineering,* Vol. 11 No.5, pp529-558.
[40] Zhu, H. 2003. A Formal Specification Language for Agent-Oriented Software

Engineering. *Proc. of AAMAS'2003*, Melbourne, Australia, pp1174 – 1175.

[41] Zhu, H. 2002. Formal Specification of Evolutionary Software Agents. *Proc. of ICFEM'2002*, Shanghai, China, Oct. pp249-261.

[42] Zhu, H. 2002. Developing Formal Specifications of Multi-Agent Systems in SLABS -- A Case Study of Evolutionary Multi-Agent Ecosystem. *Proc. of AOIS'2002 at AAMAS'02*, Bologna, Italy, pp20-34.

[43] Randles, M., Zhu, H. and Taleb-Bendiab, A. 2007. A Formal Approach to the Engineering of Emergence and its Recurrence. *Proc. of The Second International Workshop on Engineering Emergence in Decentralised Autonomic Systems* (EEDAS 2007), Jacksonville, Florida, USA. Greenwich University Press, London, UK, pp12-21.

[44] Zhu, H. and Shan, L. 2007. Agent-Oriented Modelling and Specification of Web Services. *The International Journal of Simulation and Process Modelling*, Special Issue on Trustworthy Web Services, Vol. 3, No.1&2, pp26 – 44.

[45] Mao, X., Shan, L., Zhu, H. and Wang, J. 2008. An Adaptive Casteship Mechanism for Developing Multi-Agent Systems. *International Journal of Computer Application in Technology*, Vol. 31, Nos. 1/2, pp17-34.

[46] Zhu, H. 2005. Formal reasoning about emergent behaviours of MAS. *Proc. of Software Engineering and Knowledge Engineering*, Taipei, pp280-285.

[47] Jennings, N. R. 2000. On agent-based software engineering. *Artificial Intelligence* Vol. 117, pp277-296.

[48] Wang, S. and Zhu, H. 2007. An Experimental Study of the Emergent Behaviors of Self-organizing Agent Communities. *Proc. of the 2007 IEEE Congress on Evolutionary Computation* (CEC 2007), Singapore, pp3239-3246.