CAMLE: A Caste-Centric Agent-Oriented Modelling Language and Environment

Lijun Shan[1] and Hong Zhu[2]

[1] Department of Computer Science, National University of Defence Technology
Changsha, 410073, P.R. China
lijunshancn@yahoo.com
[2] Department of Computing, Oxford Brookes University, Oxford OX33 1HX, UK
hzhu@brookes.ac.uk

Abstract. This paper presents an agent-oriented modelling language and environment CAMLE. It is based on the conceptual model of multi-agent systems (MAS) proposed and formally defined in the formal specification language SLABS. It is caste-centric because the notion of caste plays the central role in its methodology. Caste is the classifier of agents in our language. It allows multiple and dynamic classifications of agents. It serves as the template of agents and can be used to model a wide variety of MAS concepts, such as roles, agent societies, etc. The language supports modelling MAS at both macro-level for the global properties and behaviours of the system and micro-level for properties and behaviours of the agents. The environment provides tools for constructing graphic MAS models in CAMLE, automatically checking consistency between various views and models at different levels of abstraction, and automatically transforming models into formal specifications in SLABS. The uses of the CAMLE modelling language and environment are illustrated by an example.

1 Introduction

One of the key factors that contribute to the progress in software engineering over the past two decades is the development of increasingly powerful and natural high-level abstractions with which complex systems are modelled, analysed and developed. In recent years, it becomes widely recognized that agents represent an advance in this direction that can unify data abstraction and operation abstraction. A number of agent-oriented software development methodologies have been proposed in the literature; see e.g. [1]. These proposals vary in how to describe agent and MAS at a higher abstraction level as well as how to obtain such a description. For example, Gaia [2] provides software engineers with the organization-oriented abstraction in which software systems are conceived as organized society and agents are seen as role players. Tropos [3] emphasizes the uses of notions related to mental states during all software development phases. The notions like belief, intention, plan, goals, etc., represent the abstraction of agent's state and capability.

Our work originates from formally specifying agent behaviour as responses to environment scenarios [4], developed into a formal specification language SLABS for engineering agent-based systems [5], and applied to a number of examples of MAS [6, 7]. In [8] and [9], a diagrammatic notation for modelling agent behaviours and

R. Choren et al. (Eds.): SELMAS 2004, LNCS 3390, pp. 144–161, 2005.

collaborations was respectively developed. This paper proposes a methodology of agent-oriented software engineering called CAMLE, which stands for Caste-centric Agent-oriented Modelling Language and Environment. Caste is the classifier of agents in our modelling and specification languages. It allows multiple classifications (i.e. an agent can belong to more than one caste) and dynamic classifications (i.e. an agent can change its caste membership at run time), as well as multiple inheritances among castes. It can be used to model a wide variety of MAS concepts, such as roles, agent societies, behaviour normality, etc. It provides the modularity language facility and serves as the template of agents in the design and implementation of MAS. The notion of caste plays a central role in the methodology. We consider behaviour rules as the basic abstraction for agent's behaviour while leaving out mental state notions such as belief and goal that are used in some other agent-oriented software researches, though such notions can be represented in our framework. Behaviour rules incorporating agent's perception to its environment represent the autonomy of agent's behaviour. With the CAMLE language, a software system can be modelled from three perspectives. The supporting tools help users to construct MAS models in graphical notations, to check the consistency between models from various views and at different abstraction levels, and automatically translate the graphic models into formal specifications.

The remainder of this paper is organized as follows. Section 2 reviews the underlying conceptual model. Section 3 presents the modelling language. Section 4 briefly reports the modelling tools. Section 5 concludes the paper with discussions on related work and directions for future work.

2 Conceptual Model

The conceptual model of MAS underlying our methodology is the same as that of the language SLABS [4, 5], which is a formal specification language designed for engineering MAS. It can be characterized by a set of pseudo-equations. Pseudo-equation (1) states that agents are defined as real-time active computational entities that encapsulate data, operations and behaviours, and situate in their designated environments.

$$\text{Agent}=<\text{Data, Operations, Behaviour}>_{\text{Environment}} \quad (1)$$

Here, data represent an agent's state. Operations are the actions that the agent can take. Behaviour is described by a set of rules that determine how the agent behaves including when and how to take actions and change state in the context of its designated environment. By encapsulation, we mean that an agent's state can only be changed by the agent itself, and the agent can decide 'when to go' and 'whether to say no' according to an explicitly specified set of behaviour rules. Therefore, there are two fundamental differences between objects and agents in our conceptual model. First, objects do not contain any explicitly programmed behaviour rule. Second, objects are open to all computation entities to call its public methods without any distinction of them.

In our conceptual model, the classifier of agents is called caste. Castes classify agents into various castes similar to that data types classify data into types, and classes classify objects into classes. However, different from the notion of class in object orientation, caste allows dynamic classification, i.e. an agent can change its

caste membership (called casteship in the sequel) at run time. It also allows multiple classifications, i.e. an agent can belong to more than one caste at the same time. As all classifiers, inheritance relations can also be specified between castes. As a consequence of multiple classifications, a caste can inherit more than one caste. As a modularity language facility, a caste serves as a template that describes the structure and behaviour properties of agents in the caste, and as the basic organizational units in the design and implementation of MAS. Pseudo-equation (2) states that a caste is a set of agents that have the same structural and behavioural characteristics at any time moment t in the execution of the system. The structure of caste descriptions in SLABS is shown in Fig.1.

$$\text{Caste}_t = \{\text{agents} \mid \text{structure \& behaviour properties}\} \quad (2)$$

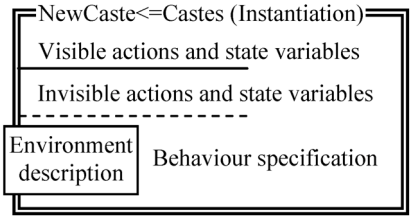


Fig. 1. Caste descriptions in SLABS

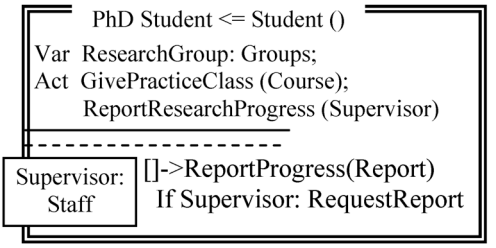


Fig. 2. An example of caste: PhD Student

For example, when modelling a university as an information system, each of the people in the university can be modelled as an agent. They can be grouped into a number of castes, such as the caste of students, the caste of faculty members and the caste of secretaries. The students can be further classified into undergraduates, graduates and Ph. D students. Fig. 2 is an example of the caste definition of Ph. D student.

In the real world, an undergraduate student can become a postgraduate student or alumni after graduation. To model this, the agents in an information system dynamically change their membership to castes. The weakness of static object-class relationship in current mainstream object-oriented programming has been widely recognized. Proposals have been advanced, for example, to allow objects' dynamic reclassification [10]. In [11], we suggested that agents' ability to dynamically change its roles is represented by dynamic casteship. In our model, dynamic casteship is an integral part of agents' behaviour capability. Agents can have behaviour rules that allow them to change their castes at run-time autonomously. To change its casteship, an agent takes an action to join a caste or retreat from a caste at run time. Therefore, which agents are in a caste depends on time even if agents can be persistent, hence the subscript of *t* in pseudo-question (2). We believe that this feature allows users to model the real world MAS naturally and to maximize the flexibility and power of agent technology.

In the research on agent-oriented methodologies, a number of notions have been proposed in the literature to model agent-based systems, which include role, agent society, organisation, normative behaviour, etc. The notion and language facility of caste can be used to represent these concepts as discussed in [6]. For example, the set of agents that play a specific role can be defined by a caste. The agents of a particular society or community that obey a specific set of normative behaviour rules and share

a set of resources can also be defined as a caste. However, the notions of societies and role etc. are too specific and restrictive to be used as a language facility. For example, all the people who speak a particular language, say Chinese, can be defined as a caste, but it would be unnatural to consider them as playing any specific role. Similarly, a set of software agents that follow a particular communication protocol can be defined by a caste, but it would be unnatural to model them by a role. The concept of society or community has a strong sense of membership. A person who speaks English does not necessarily belong to the society of English people. A society may also consists of agents playing different roles and obey different behaviour rules. Therefore, the concept of society is not suitable to be used as a language facility of code template.

The notion of role has been widely used to characterize agents' behaviour and interaction in agent-oriented methodologies, especially in the analysis and specification stage [13]. However, role is often used intuitively in system analysis. They are transformed into agent properties at design stage and eventually disappear in programming stage or represented indirectly as objects and classes. In contrast, caste not only overcomes the limitations and weakness of the informal notions of roles and societies at analysis and specification stage, but, as a language facility, can also be directly implemented in a programming language such as SLABSp [12].

Equation (3) states that in our model a MAS consists of a set of agents but nothing else. Our definition of agent implies that object is a special case of agent in the sense that it has a fixed rule of behaviour, i.e. "executes the corresponding method when receives a message".

$$\text{MAS} = \{\text{Agent}_{\,n}\},\ n \in \text{Integer} \tag{3}$$

Consequently, the environment of an agent in a MAS at time t is a subset of the agents, where some agents in the system may not be visible from the agent's point of view, as illustrated in pseudo-equation (4). Notice that, our use of the term 'visibility' is different from the concept of scope. In particular, from agent A's point of view, agent B is visible means that agent A can observe and perceive the visible actions taken by agent B or obtain the value of agent B's visible part of state at run time.

$$\text{Environment}_{\,t}\,(\text{Agent}, \text{MAS}) \subseteq \text{MAS} - \{\text{Agent}\} \tag{4}$$

Here, we take a 'designated environment' approach, i.e. the environment of an agent is specified when an agent is designed. The environment description of an agent or a caste defines what kinds of agents are visible. For example, it can be that the agents in a particular caste are visible. Note that a designated environment is neither closed, nor fixed, nor totally open. Since an agent can change its casteship, its environment may change dynamically. For example, an agent's environment changes when it joins a caste and hence the agents in the caste's environment become visible. The environment also changes when other agents join the caste in the agent's environment. Therefore, the set of agents in the environment of an agent depends on time, hence, the subscription t in pseudo-question (4).

The communication mechanism in our model is that an agent's actions and states are divided into the visible ones and internal ones. Agents communicate with each other by taking visible actions and changing visible state variables, and by observing other agents' visible actions and visible states, as shown in pseudo-equation (5). An agent taking a visible action can be understood as generating an event that can be perceived by other agents in the system, while an agent taking an internal action

means it generates an event that can only be perceived by its components. Similarly, the value of an agent's visible state can be obtained by other agents, while the value of the internal state can only be obtained by its components.

$$A \rightarrow B = A.Action \ \& \ B.Observation \quad (5)$$

This communication mechanism is different from message passing between objects where each message invokes a corresponding method of the object that receives the message. In our model, agents are active computational entities that execute concurrently. They are not invoked by messages. Instead, each agent observes the events happened in its environment and takes actions according to its behaviour rules. How an agent handles an event that it perceives is solely determined by the agent itself because agents are autonomous. In general, the agent that produces an event may not know which agent in the system will respond to the event or how the event will be handled. Therefore, the agent does not expect any agent to participate in the generation of an event. It may not even wait for the event to be handled to progress its own computation task. In this sense, the communication mechanism can be considered as asynchronous and non-blocking. Of course, this mechanism does not define the communication protocol and agent communication language. These issues should be addressed in the design and implementation of specific multi-agent system, rather than predefined by the modelling language or meta-model.

3 Modelling Language

CAMLE employs the multiple view principle. A MAS model contains three types of models: caste models, collaboration models and behaviour models. Each model consists of one or more diagrams.

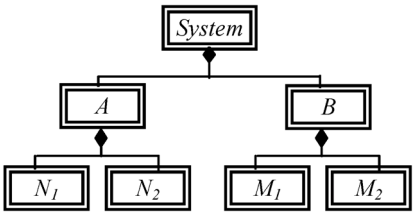
The caste model specifies the castes of the system and the relationships between them. A caste is a compound caste if its agents are composed of a number of other agents; otherwise, it is atomic. For example, as shown in Fig 3 (a), the System is directly composed of agents of caste A and B. Each of them can be further decomposed into smaller components N_1 and N_2, and M_1 and M_2, respectively. For each compound caste, such as the System, A and B, a collaboration model and a behaviour model are constructed. Atomic castes only have behaviour models because they have no components thus no internal collaboration.

The overall structure of a system's collaboration models and behaviour models can be viewed as a hierarchy, which is isomorphic to the whole-part relations described in the caste model; see e.g. Fig 3 (b).

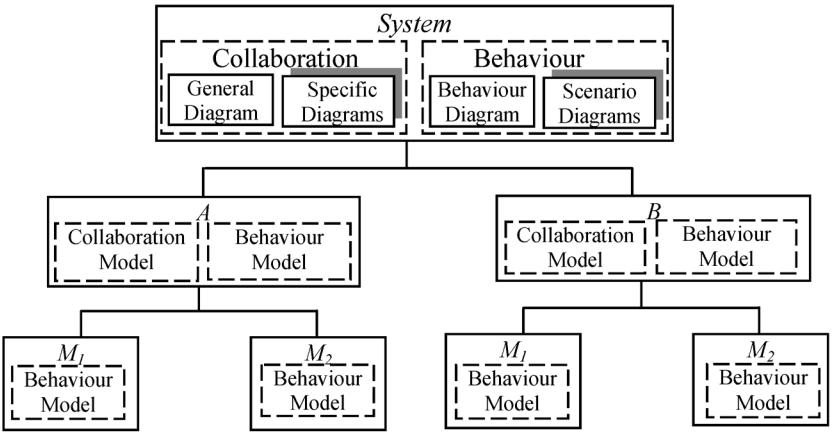
The following subsections describe each type of model and discuss their uses in agent-oriented software development, respectively; see [8, 9] for more details. Subsection 3. 4 discusses the consistency between various kinds of models.

3.1 Caste Model

We view an information system as an organization that consists of a collection of agents that stand in certain relationships to one another by being a member of certain groups and playing certain roles, i.e. in certain castes. They interact with each other by observing their environments and taking visible actions as responses to the environment scenarios. The behaviour of an individual agent in a system is determined by



(a) Example of Caste Model with Whole-Part Relations



(b) Collaboration Models and Behaviour models

Fig. 3. Overall Structure of CAMLE Models

the 'roles' it is playing. An individual agent can change its role in the system. However, the set of roles and the assignments of responsibilities and tasks to roles are usually quite stable [13]. Such an organizational structure of information systems is captured in our caste model.

A caste diagram identifies the castes in a system, indicates the inheritance, aggregation and migration relationships between them. Fig 4 shows the notation of caste diagrams and illustrates it with the university example introduced in section 2.

The inheritance relationship between castes defines sub-groups of the agents that have special responsibilities and hence additional capabilities and behaviours. Migration relations specify how agents in the castes can change their casteships. There are two kinds of migration relationships: migrate and participate. A migrate relation from caste A to B means that an agent of caste A can retreat from caste A and join caste B. A participate relation from caste A to B means that an agent of caste A can join caste B while retaining its casteship of A. For example, in Fig 4, an undergraduate student may become a postgraduate after graduation. A postgraduate student may become a PhD student after graduation or become a faculty member. Each student becomes a member of the alumni of the university after leaving the university. A faculty member can become a part time PhD student while remaining employed as a faculty member. From this model, we can infer that an individual can be both a student and a faculty member at the same time if and only if he/she is a PhD student.

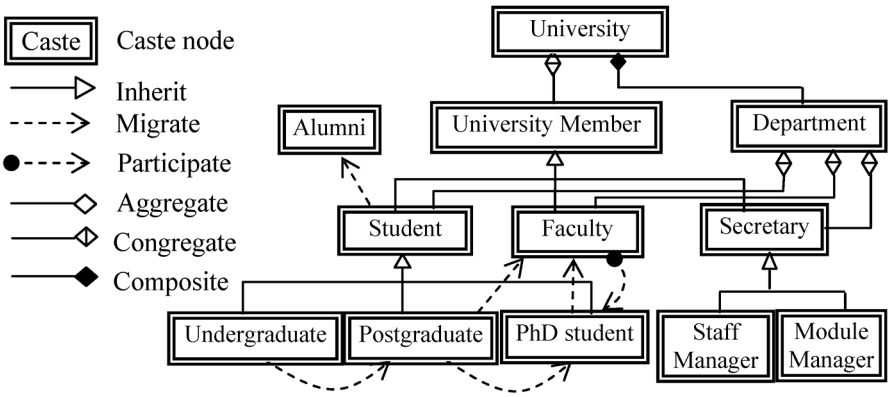


Fig. 4. Caste diagram: notations and example

An agent may contain a number of components that are also agents. The former is called compound agent of the latter. In such a case, there exists a whole-part relationship between the compound agent and the components. We identify three types of whole-part relationships between agents according to the ways a component agent is bound to the compound agent and the ways the compound agent controls its components. The strongest binding between a compound agent and its components is *composition* in which the compound agent is responsible for creation and destruction of its components. If the compound agent no longer exists, the components will not exist. The weakest binding is *aggregation*, in which the compound and the component are independent to each other, so that the component agent will not be affected for both its existence and casteships when the compound agent is destroyed. The third whole-part relation is called *congregation*. It means if the compound agent is destroyed, the component agents will still exist, but they will lose the casteship of the component caste.

The composition and aggregation relation is similar to the composition and aggregation in UML, respectively. However, congregation is a novel concept in modelling languages introduced in by CAMLE. There is no similar counterpart in object oriented modelling languages, such as UML. It has not been recognized in the research on object-oriented modelling of whole-part relations [14]. We believe that it is important for agent-oriented modelling because of agents' basic feature of dynamic casteship. For example, as shown in Fig 4, a university consists of a number of individuals as its members. If the university is destroyed, the individuals should still exist. However, they will lose the membership as the university member. Therefore, the whole-part relationship between University and University Member is a congregation relation. This relationship is different from the relationship between a university and its departments. Departments are components of a university. If a university is destroyed, its departments will no long exist. The whole-part relationship between Department and University is therefore a composition relation.

The semantics of the whole-part relations at modelling level given above has a number of implications on the operations on agents at implementation level , especially the creation and destroy of agents. For example, a composition relation implies that a component agent can be destroyed when the compound agent is destroyed.

In contrast, a component agent must be kept intact even if the compound agent is destroyed if the whole-part relation is aggregate. In object-oriented systems, a component object can be destroyed or garbage collected if there is no more reference to the component object. Therefore, to support garbage collection, a reference count to a component object should be maintained and the reference count must be decreased if the compound object is destroyed. However, in agent-oriented systems, because agents are active computation entities, an agent cannot be destroyed unless explicitly instructed by the user even if it is not a component of any compound agent. For congregation relation, when a compound agent is destroyed, the component agents should not be destroyed, but their casteship must be changed so that they are no longer members of the component castes of the compound agent. It is an open question whether or not an agent should be destroyed if it no longer belongs to any caste.

3.2 Collaboration Model

While caste model defines the static architecture of MAS, collaboration model implicitly defines the dynamic aspect of the MAS organization by capturing the collaboration dependencies and relationships between the agents.

Agents in a MAS collaborate with each other through communication, which is essential to fulfil the system's functionality. Such interactions between agents are captured and represented in a collaboration model. In CAMLE, a collaboration model is associated to each caste and consists of a set of collaboration diagrams.

A collaboration diagram specifies the interaction between the agents in the system or in a compound agent. Fig. 5 gives the notations.

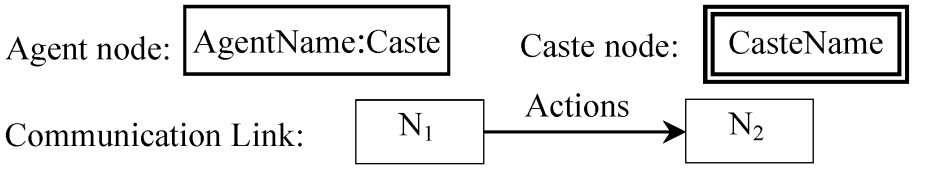


Fig. 5. Notation of Collaboration Diagram

There are two types of nodes in a collaboration diagram. An agent node represents a specific agent. A caste node represents any agent in a caste. An arrow from node A to node B represents that the visible behaviour of agent A is observed by agent B. Therefore, agent A influences agent B. When agent B is particularly interested in certain activities of agent A, the activities can also be annotated to the arrow from A to B. Although this model looks similar to collaboration diagrams in UML, there are significant differences in the semantics. In OO paradigm, what is annotated on the arrow from A to B is a method of B. It represents a method call from object A to object B, and consequently, object B must execute the method. In contrast, in CAMLE the action annotated on an arrow from A to B is a visible action of A. Moreover, agent B is not necessarily to respond to agent A's action. The distinction indicates the shift of modelling focus from controls represented as method calls in OO paradigm to collaborations represented as signalling and observation of visible actions. It fits well with the autonomous nature of agents.

3.2.1 Scenarios of Collaboration. One of the complications in the development of collaboration models is to deal with agents' various behaviours in different scenarios. By scenario, we mean a typical situation of the operation of the system. In different scenarios, agents may pass around different sequences of messages and may communicate with different agents. Therefore, it is better to describe them separately. The collaboration model supports the separation of scenarios by including a set of collaboration diagrams. Each diagram represents one scenario. In such a scenario specific collaboration diagram, actions annotated on arrows can be numbered by their temporal sequence. Fig. 6 below gives an example of scenario-specific collaboration diagram. It describes the collaborations of an undergraduate student with his/her personal tutor, the faculty members who give lectures and the PhD students who are practical class tutors.

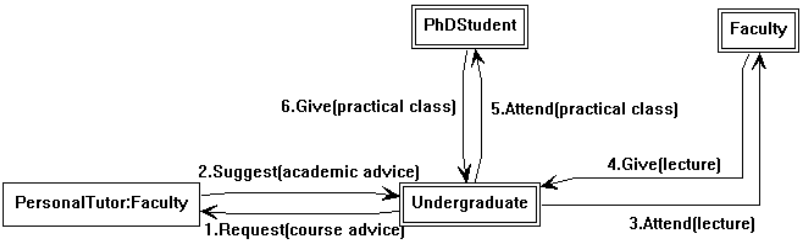


Fig. 6. An example of Scenario-Specific Collaboration Diagram

In addition to such specific diagrams, a general collaboration diagram is also associated to the caste to give an overall picture of the communication between all the component agents by describing all visible actions an agent may take and all possible observers of the actions. Fig. 7 describes the communications within a department between various agents.

3.2.2 Refinement of Collaboration Models. The modelling language supports modelling complex systems at various levels of abstraction, and to refine from high-level models of coarse granularity to more detailed fine granularity models. At the top level, a system can be viewed as an agent that interacts with users and/or other systems in its external environment. This system can be decomposed into a number of subsystems interacting with each other. A sub-system can also be viewed as an agent and further decomposed. As analysis deepens, a hierarchical structure of the system emerges. In this way, the compound agent has its functionality decomposed through the decomposition of its structure. Such a refinement can be carried on until the problem is specified adequately in detail. Thus, a collaboration model at system level that specifies the boundaries of the application can be eventually refined into a hierarchy of collaboration models at various abstraction levels. Of course, the hierarchical structure of collaboration diagrams can also be used for bottom-up design and composition of existing components to form a system.

Fig. 8 gives an example of general collaboration diagram that refines the caste Dept Office. In this diagram, the agents in the castes of Student and Faculty as well as a specific agent called Dept Head in the caste of Faculty form the environment of the caste Dept Office. Therefore, they are visible for the component agents of the caste.

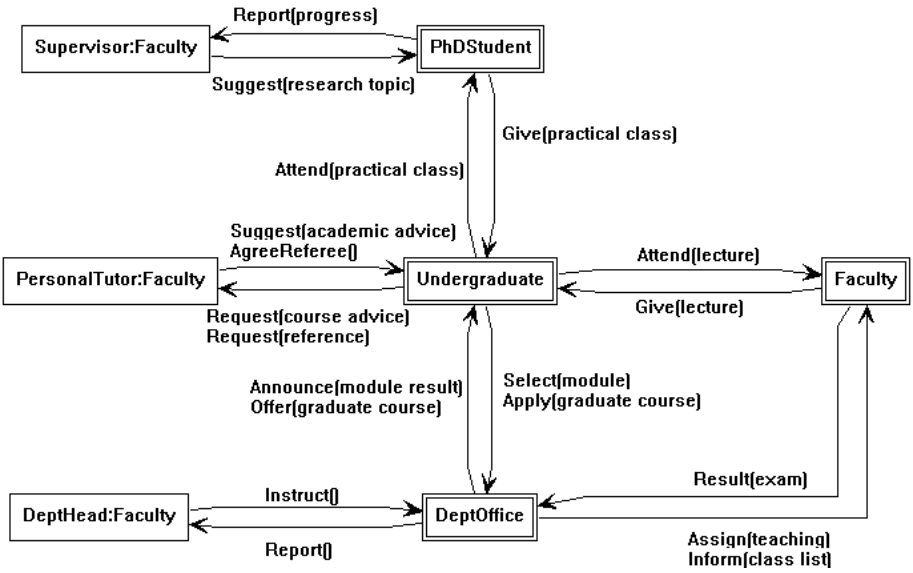


Fig. 7. An example of general collaboration diagram

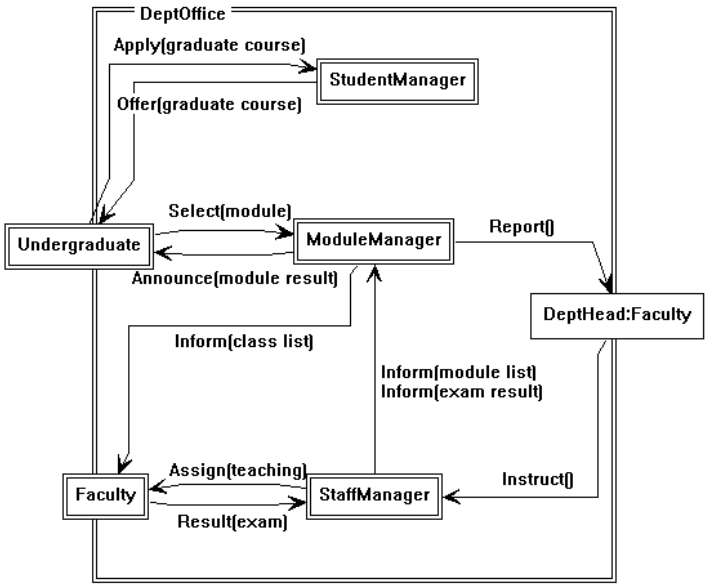


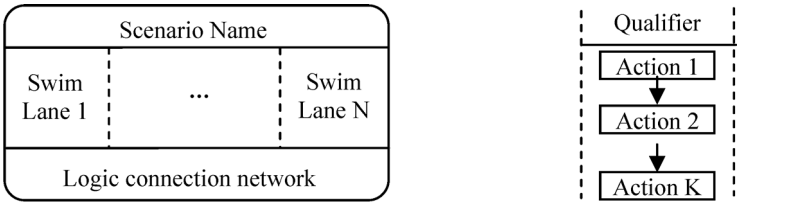
Fig. 8. An example of general collaboration diagram that refines a caste

3.3 Behaviour Model

While caste and collaboration models describe MAS at the macro-level from the perspective of an external observer, behaviour model adopts the internal or first-person

view of each agent. It describes an agent's dynamic behaviour in terms of how it acts in certain scenarios of the environment. A behaviour model consists of two kinds of diagrams: scenario diagrams and behaviour diagrams.

3.3.1 Scenario Diagrams. From an agent's point of view, the situation of its environment is characterized by what is observable by the agent. In other words, a scenario is defined by the sequences of visible actions taken by the agents in its environment. Scenario diagrams identify and describe the typical situations that the agent must respond to. Fig. 9 shows the layout of scenario diagrams. Fig. 10 gives the notations for specifying visible events and their temporal ordering in scenario diagrams, as well as logic connective for the combination of situations.



(a) The layout of scenario diagrams (b) The layout of swim lanes

Fig. 9. Format of Scenario Diagram

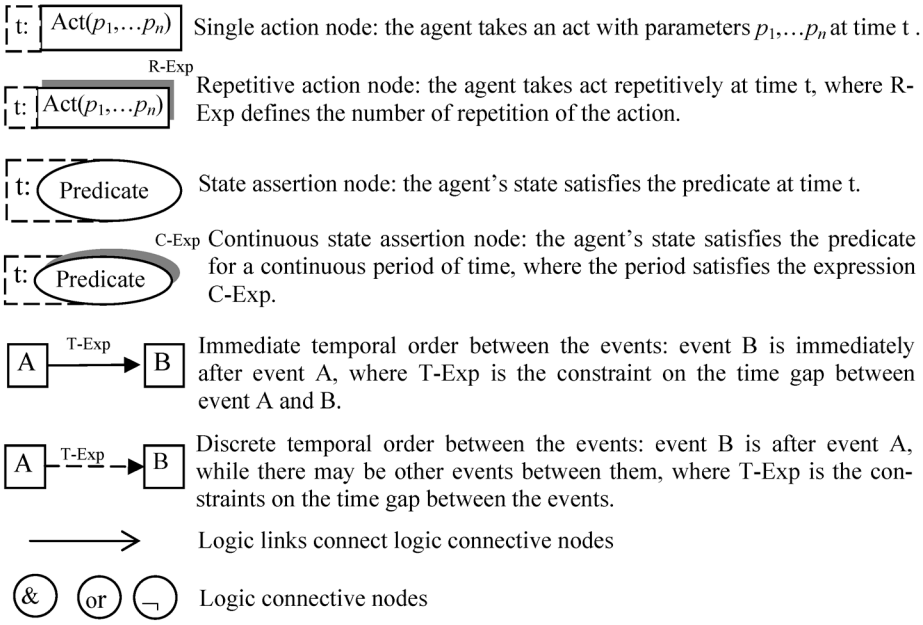


Fig. 10. Notations of Scenario Diagram

For example, Fig. 11 describes a scenario where Greenspan announces that the interest rate will increase by 0.25 points and all stock market analysts recommend sell Microsoft's share.

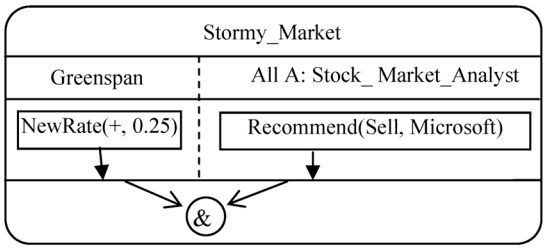
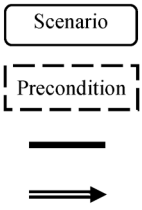


Fig. 11. Example of scenario diagram

3.3.2 Behaviour Diagrams. Behaviour diagrams describe agents' designed behaviour in certain scenarios. For each caste, a behaviour diagram defines a set of behaviour rules. Each rule describes how the agent of the caste should respond to a particular situation in the environment (i.e. in a scenario). The notation of behaviour diagrams includes the notation of scenario diagrams plus those in Fig. 12.



Scenario node: a scenario identifier, or a detailed scenario description.

Precondition node: give the precondition of an event.

Transition bar: conflux of scenario, precondition and previous events as premise of behaviour rule.

Action arrow: link from behaviour rule's previous events to transition bar.

Fig. 12. Notation for behaviour diagrams

A behaviour diagram contains event nodes linked together by the temporal ordering arrows as in scenario diagrams to specify the agent's previous behaviour pattern. A transition bar with a conflux of scenario, precondition and previous pattern and followed by an event node indicates that when the agent's behaviour matches the previous pattern and the system is in the scenario and the precondition is true, the event specified by the event node under the transition bar will be taken by the agent. In a behaviour diagram, a reference to a scenario indicated by a scenario node can be replaced by a scenario diagram if it improves the readability. The behaviour diagram in Fig. 13 partly defines the behaviour of an undergraduate student. It states that if the student is in the final year and the average grade is 'A', the student may request a reference from the personal tutor for the application of a graduate course. If the personal tutor agrees to be a referee, the student may apply for a graduate course. If the department office offers a position in a graduate course, the student will join the Graduates caste and retreat from the Undergraduates caste.

In CAMLE language, each agent/caste has a designated environment, which is defined by its environment description. Therefore, in the development of a behaviour model for a given agent/caste, the modeller needs not to know all agents in the system, but only those in the environment. The modeller also does not need to know the full details of the behaviour of the agents in its environment, but just their capabilities in terms of the actions that they can take. As shown in the above example, the modeller needs not to know how a faculty member makes decisions on whether or not to write a reference for his/her tutee, and how the department decides who will be ac-

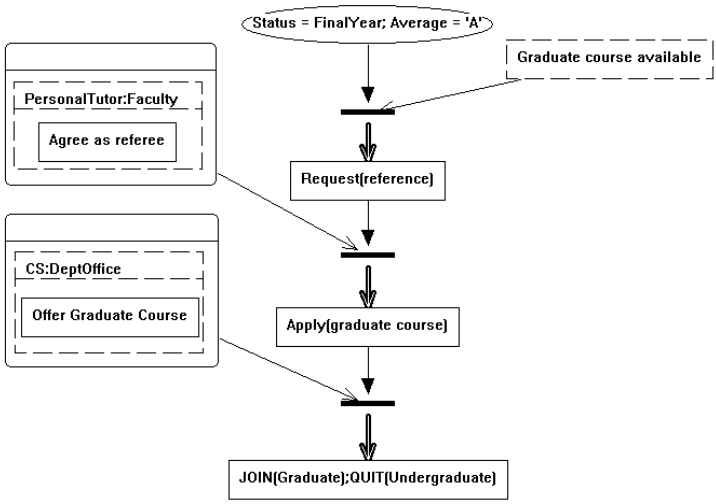


Fig. 13. Behaviour Diagram for Undergraduate Student

cepted as a postgraduate student. What the modeller only needs to know is that the personal tutor is capable of writing a reference letter and the department is capable of make a decision on the intake of postgraduate students[1]. We believe that this is perhaps the minimal amount of information that a modeller need to know about the system that his component will collaborate with.

3.4 Consistency of Models

Like all multiple view modelling languages, a model in CAMLE may have inconsistency between various types of models. Errors such as ill-formed diagrams may be introduced. Furthermore, because there are overlaps of information in different models, inconsistency between models can occur. In the case studies of CAMLE language and the modelling environment, we found the following types of errors and inconsistencies are among the most common problems.

(1) The same agent or caste may be refereed to with slightly different names.
(2) The set of actions occurred in the specific collaboration diagrams that specify various collaboration scenarios often does not match the set of actions declared in the corresponding general collaboration diagram.
(3) The names and parameter types of the actions defined for one caste/agent often do not match the references to these actions in scenario diagrams and behaviour diagrams.
(4) The behaviour of an agent or caste as described in a collaboration model is often inconsistent with what is defined in the behaviour model. For example, a collaboration model requires an agent to have certain sequence of activities, but the behaviour model does not define a corresponding behaviour rule or rules. It is also common that a behaviour rule requires the observation of an agent's visible action but the definition of the agent does not contain the action as a visible one.

[1] Of course, in a more complicated system, if the modeller does not have such knowledge, he/she needs to know how his/her agent can discover such knowledge at runtime.

In order to ensure the consistency between various models and models at different levels of abstraction, three types of the consistency constraints have been identified and formally defined in the CAMLE language. Automated tools are implemented in the modelling environment to check if these consistency constraints are satisfied. These consistency constraints including (A) well-formedness conditions imposed on each diagram, (B) intra-model consistency constraints that are imposed on diagrams of the same model at the same abstraction levels, and (C) inter-model consistency constraints that are imposed either on the same type of models at different abstraction levels, or on different types of models at the same level of abstraction. The definitions of the constraints are omitted here for the sake of space; see [15] for details.

4 Support Environment

A software environment to support the process of system analysis and modelling in CAMLE has been designed and implemented[2]. CAMLE aims at representing information systems naturally using the conceptual model of MAS presented in the previous section and facilitating the reasoning about such systems. It serves two interrelated purposes, i.e. to develop abstract descriptive models of current systems and to develop prescriptive designs of systems to be implemented. Therefore, in addition to model construction, two key features of the language and environment are regarded as of particular importance: (a) the consistency check between various models from different views and at different levels of abstraction, and (b) the transformation of diagrammatic models into formal specifications. Details of these functionalities are beyond the scope of this paper and are reported separately [15].

Fig 14 shows the architecture of the current CAMLE environment and its main functionality. The diagram editor supports the manual editing of models through graphic user interface. The well-formedness checker ensures that user entered models are well-formed, hence prevents syntactically incorrect diagrams from being processed. The partial diagram generator can generate partial models (incomplete diagrams) from existing diagrams to help users in model construction. It is based on the consistency constraints so that the generated partial diagrams are consistent with existing ones according to the consistency conditions. Consistency checking tools that help to ensure the well-formedness, consistency and completeness of system models are based on consistency constraints defined by the CAMLE language. Fig 15 is a screen snapshot of the output of the consistency checking tool. The diagnostic information helps users to locate and correct errors in the checked model.

The transformation from graphic models in CAMLE into formal specifications in SLABS enables engineers to analyse, verify and validate system models before the system is implemented. The specification generation tool in the CAMLE environment can automatically derive a formal specification in SLABS after a model is constructed and its consistency checked. Fig 16 shows a screen snapshot of the tool-generated specification of the caste Undergraduate.

Besides the University example used in this paper, a number of case studies of the modelling language and its modelling environment have been conducted. These case studies include the following.

---

[2] The tool is available for free for academic and research purposes. Please contact the authors.

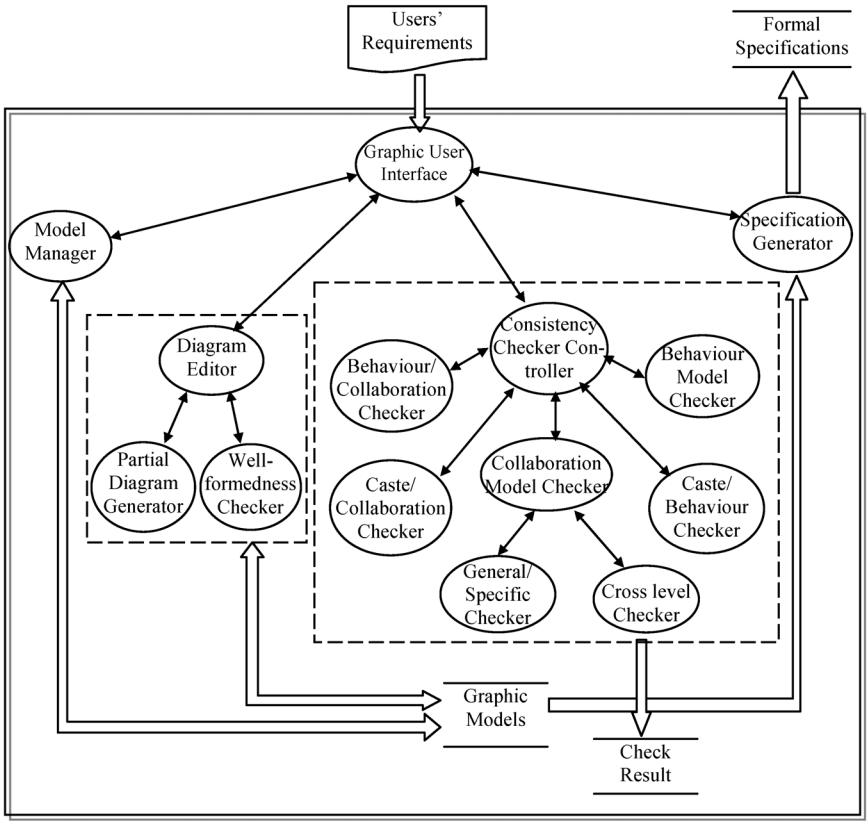


Fig. 14. The Architecture of CAMLE Environment

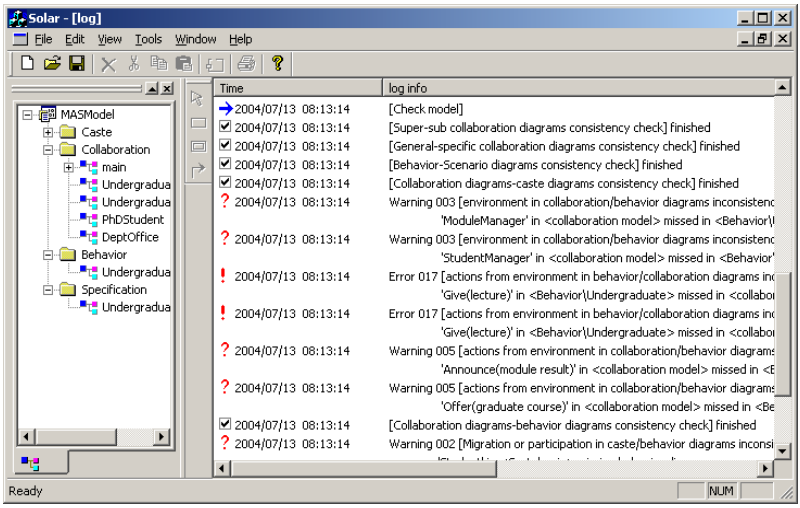


Fig. 15. Screen snapshot of the consistency checking tool's output

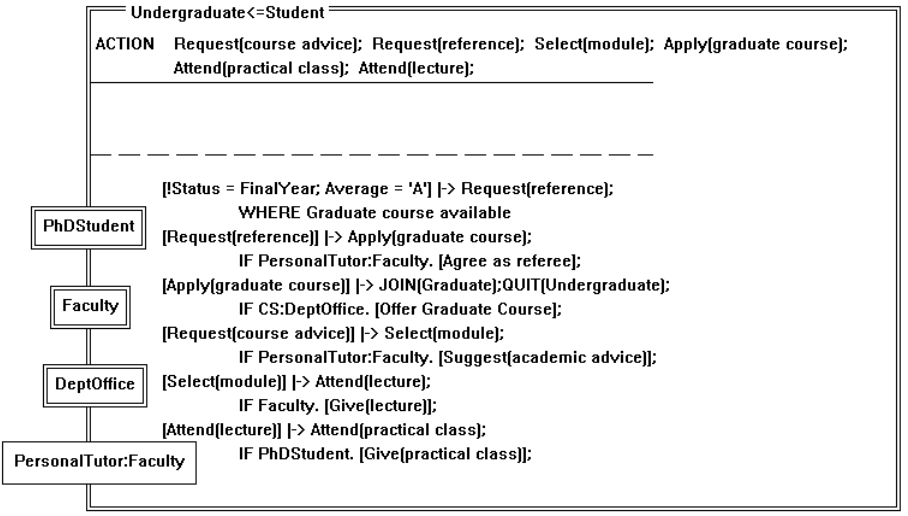


Fig. 16. Screen Snapshot of Generated Specification of the Caste Undergraduate

- *United Nations' Security Council:* The organisational structure and the work procedure to pass resolutions were modelled and a formal specification of the system in SLABS was generated. Details of the case study as well as modelling in other agent-oriented modelling notations can be found on AUML's website[3].
- *Amalthaea:* Amalthaea is an evolutionary multi-agent system developed at MIT's Media Lab to help the users to retrieve information from the Internet [16]. The system was modelled and a formal specification was generated.
- *Web Services:* The case study modelled the architecture of web services and an application of web services on online auctions. A formal specification in SLABS of the architecture and application was generated successfully. More details of the specification of web services in SLABS can be found in [17].

In the case studies, we found that the CAMLE language was highly expressive to model information systems' organisational structures, dynamic information processing procedures, individual decision making processes, and so on. Models in CAMLE were easy to understand because they naturally represent the real world systems. The automated environment greatly helped to manage the consistency between various diagrams from difference views and at different abstraction levels. It significantly reduced the difficulty in the development of formal specifications of multi-agent systems especially for complicated systems such as Amalthaea.

5 Conclusion

This paper presented the agent-oriented modelling language and environment CAMLE. It is based on the conceptual model of MAS of the formal specification language SLABS. Models represented in the CAMLE can be automatically checked

[3] URL: http://www.auml.org/

for consistency and transformed into formal specifications in SLABS by the tools in the modelling environment. The modelling language has a number of novel features, which include the congregate whole-part relation and migration relation between castes, the designated environment descriptions, scenario diagrams, scenario driven behaviour rules, and most importantly, the concept of caste.

There have been a number of efforts in the direction of AO methodology, many of which focus on the process of MAS engineering as well as the representation of MAS. With regard to conceptual model of the methodologies, there is a fundamental difference between CAMLE and the methodologies that are based on mental state related notions such as belief, desire, intention, goal and plan. Although these notions are widely used, their meanings vary from people to people in different methodologies. CAMLE replaces these notions with an abstract model of agents as encapsulation of data, operation and behaviour. CAMLE also has a fundamental distinction from the methodologies that are based on social organization related notions such as roles, agent society and organization structure. CAMLE replaces such intuitive concepts with a well-defined language facility caste, which is easy to understand and use from software engineering perspective. Caste can be used to represent a number of the concepts in agent-oriented modelling, such as roles, agent societies, normative behaviour, common knowledge and protocols, etc. [6]. The caste-centric feature enables us to achieve simplicity in the design of an expressive modelling language and efficiency in the implementation of the powerful environment.

Among the related work, Gaia is perhaps one of the most mature agent-oriented software development methodologies at the moment. It does not commit to specific notations for modelling concepts such as roles, environment and interaction [1]. UML notation are widely used, e.g. in Tropos, PASSI [18] and AUML [19]. However, there is no clearly defined conceptual model or meta-model underlying the uses of UML notations for agent-oriented modelling although there are fundamental differences between agents and objects as discussed in section 2 and 3.

A related work on agent-oriented modelling language is ANote [20], which also provides a set of diagrams to model different views of MAS. Systems' structural aspects, dynamic aspects and physical aspects are specified with the notations representing the concepts of goal, agent, ontology, scenario, planning, interaction and organization etc. Although the concepts and notations are different from the ones used in CAMLE, we share the same opinion that using (modified) OO paradigm to model agent-based system is not desirable.

There are several issues remaining for future work. We are investigating software tools that support model-based implementation of MAS in CAMLE. The design and implementation of an agent-oriented programming language with caste as the basic program unit is on the top of our agenda.

Acknowledgement

The work reported in this paper is supported by China High-Technology R&D Programme under the grant 2002AA116070.

References

1. Dam, K. H., Winikoff, M., "Comparing Agent-Oriented Methodologies", *Proc. of AOIS'03,* Melbourne, Australia, July 2003.
2. Zambonelli, F., Jennings, NR and Wooldridge, M., "Developing multiagent systems: the Gaia Methodology", *ACM TOSEM* 12(3), 2003, pp. 317-370.
3. Bresciani, P., Giorgini, P., Giunchiglia, F., Mylopoulos J. and Perini, A., "TROPOS: An Agent-Oriented Software Development Methodology", *Journal of Autonomous Agents and Multi-Agent Systems* 8(3), Kluwer Academic Publishers, May 2004, pp. 203 - 236.
4. Zhu, H., "Formal Specification of Agent Behaviour through Environment Scenarios", *Formal Aspects of Agent-Based Systems*, Rash, J.L., *et al.*, (Eds.), Springer, LNCS Vol. 1871, 2001, pp. 263-277.
5. Zhu, H., "SLABS: A Formal Specification Language for Agent-Based Systems", *Int. J. of Software Engineering and Knowledge Engineering* 11(5), 2001, pp. 529-558.
6. Zhu, H., "The role of caste in formal specification of MAS", *Intelligent Agents: Specification, Modelling, and Application, Proc. of PRIMA'01*, Yuan, S-T; Yokoo, M. (Eds.), LNCS, Vol. 2132, Springer, 2001, pp.1-15.
7. Zhu, H., "Formal Specification of Evolutionary Software Agents", *Formal Methods and Software Engineering*, *Proc. of ICFEM'2002*, George, C. and Miao, H., (Eds.), LNCS, Vol. 2495, Springer, 2002, pp.249~261.
8. Shan, L. and Zhu, H., "Analysing and Specifying Scenarios and Agent Behaviours", *Proc. of IAT'03,* Halifax, Canada, Oct. 2003.
9. Shan, L. and Zhu, H., "Modelling Cooperative Multi-Agent Systems", *Proc. of the 2nd Int. Workshop on Grid and Cooperative Computing*. Shanghai, China. Dec. 2003.
10. Drossopoulou, S., Damiani, F., Dezani-Ciancaglini, M. and Giannini, P., "More dynamic object reclassification: Fickle$_{II}$", *ACM TOPLAS,* 24(2), 2002, pp. 153-191.
11. Zhu, H., and Lightfoot, D., "Caste: A step beyond object orientation", *Modular Programming Languages, Proc. of JMLC'2003*, Boszormenyi, L., & Schojer, P. (eds), LNCS Vol. 2789, Springer, 2003, pp.59-62.
12. Shen, R., Wang, J. and Zhu, H., "Scenario Mechanism in Agent-Oriented Programming", *Proc. of APSEC'04*, Oct 30-Dec 3, 2004, Busan, Korea, *in press*.
13. Odell, J., Parunak, H. V. D. and Fleischer, M., "The Role of Roles", *Journal of Object Technology* 2(1), 2002, pp.39-51.
14. Barbier, F., Henderson-Sellers, B., Le Parc A. and Bruel J-M., "Formalization of the Whole-Part Relationship in the Unified Modelling Language", *IEEE TSE* 29(5), 2003, pp.459-470.
15. Shan, L. and Zhu, H., "Consistency Check in Modeling Multi-Agent Systems", *Proc. of COMPSAC'04*, Hong Kong, IEEE CS, Sept., 2004.
16. Moukas, A., "Amalthaea: Information Discovery and Filtering Using a Multi-Agent Evolving Ecosystem", *Journal of Applied Artificial Intelligence*, 11(5), 1997, pp.437–457.
17. Zhu, H., Bin Zhou, B., Mao, X., Shan, L., and Duce, D., "Agent-Oriented Formal Specification of Web Services", *Proc. of the AAC-GEVO'04 at GCC'04*, LNCS Vol. 3252, Springer, Oct. 2004.
18. Burrafato, P. and Cossentino, M., **"**Designing a Multi-Agent Solution for a Bookstore With the PASSI Methodology", *Proc.* of AOIS'02 at CAiSE'02, May 2002.
19. Bauer, B., Muller, J.P. and Odell, J., "Agent UML: A Formalism for Specifying Multiagent Software Systems", *Agent-Oriented Software Engineering*, Ciancarini, P. and Wooldridge, M. (eds.), LNCS, Vol. 1957, Springer, 2001, pp.91-103.
20. Shan, L. and Zhu, H., "CAMLE: A Caste-Centric Agent Modelling Language and Environment", *Proc. of SELMAS'*04 *at ICSE 2004*, Edinburgh, Scotland, UK, May 2004.