

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

Lijun Shan
Department of Computer Science
National University of Defence Technology
Changsha, China
Email: lijunshancn@yahoo.com

Hong Zhu
Department of Computing
Oxford Brookes University
Oxford OX33 1HX, England
Email: hzhu@brookes.ac.uk

Abstract

The necessity of a methodology for developing multi-agent systems (MAS) has been widely recognized due to the difficulties in analysis, specification, design, implementation and testing the autonomous and collaborative behaviours of such systems. This paper presents CAMLE – a caste-centric agent modeling language and environment. The methodology underlying the language and environment is based on the concept of caste, which is a language facility proposed in the formal specification language SLABS for agent-based systems. A process of system analysis and modelling is proposed for modeling MAS at both the macro-level for global properties of the system and micro-level for properties and behaviours associate to each agent.

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 recognised that agents represent an advance in this direction that can unify data abstraction and operation abstraction. Through a great number of successful examples, they have demonstrated a promising power for software developers to understand, model and develop a wide range of important complex distributed systems more naturally, more effectively and more efficiently [1]. The advantages of agent technology have also been recognised in Internet computing. It is widely perceived to be suitable for service oriented computing such as applications of web services, where information systems are constructed from a number of components distributed over the Internet, execute autonomously, and interacts with each other cooperatively. These components can be regarded as agents in a loose concept of agency. However, methodologies for effectively applying agent technology in such areas are still under development.

The OO methodology is insufficient for developing agent-based systems because it cannot naturally represent the essential characteristics of agents, such as autonomous behaviour, designated environment and sociability. These characteristics have shifted the focus of design from pre-determined cooperation between components at design time to dynamically established cooperation at run-time. For example, a web services application would be able to search for service providers from a web services registry and then dynamically establish a cooperation relationship with the service provider and request services from the provider. Furthermore, evolution, self-organization and emergent behaviours resulting from the above characteristics commonly occur in multi-agent systems (MAS). These properties are not properly addressed in OO methodologies. Therefore, a new methodology that is capable of dealing with these issues and builds on the agent concept as a central concept is necessary for systematic, effective and efficient development of MAS.

The existing works on agent-oriented software engineering can be classified into two main camps: technique specific methods, and general methods. A technique specific method is based on a specific agent language or agent theory and/or aims at developing MAS to be executed on a specific agent platform or environment. A typical example of such methods is DESIRE [2]. Technique specific methods have a number of advantages, which include practically usable, efficient and effective for certain types of applications, etc. However, they suffer from the weakness that they may impose unnecessary restriction due to their underlying techniques. It is hard to see how such methods can become the mainstream of software engineering in the near future. General methods aim at providing generally applicable modelling and/or development methods including guidance to development process

* The work reported in this paper is partly supported by the National High Technology Research Programme of China under the grant 2002AA116070.

and supporting languages without refer to any specific agent theory and implementation techniques. Typical examples of this type of work include Gaia [1], MaSE [3], etc. There are also works that are evolutions from OO paradigm, e.g. AUML [4]. However, these methodologies have no clear conceptual model (or meta-model) of MAS that provides clear definitions of the basic concepts and their interrelationships.

In our previous work on formal specification language SLABS for agent-based systems [5, 6], we developed a conceptual model as the foundation of the formal definition of the semantics of SLABS language. The central concept of the conceptual model is *caste*, which can be roughly considered as the ‘class’ of agents [7, 8], but there are a number of new features that classes do not have. In the investigation of the concept of caste and its role in the development of MAS, we recognised that caste must play a central role in agent-oriented methodology just like that class is the central concept of object-oriented (OO) methodology. This paper further develops the methodology of agent-oriented software engineering by proposing a modelling language called CAMLE and reports our work in progress on a modelling tool, where CAMLE stands for Caste-centric Agent Modelling Language and Environment. It is based on the same conceptual model of MAS developed in SLABS.

The remainder of this paper is organized as follows. In section 2, we review the underlying conceptual model of MAS. Section 3 outlines CAMLE’s modelling process. Section 4 presents a diagrammatic agent-oriented modelling language. Section 5 concludes the paper with discussions on related work and directions for future work.

2. Underlying Conceptual Model of Multi-Agent Systems

The conceptual model of MAS underlying our methodology is the same as that of the language SLABS, which is a model-based formal specification language designed for engineering MAS. The SLABS language was first proposed in [5] and further developed in [6]. In [7, 8], its pragmatic issues and practical usability were investigated.

Our conceptual model of MAS can be characterized by a set of pseudo-equations. A formal definition of the conceptual model can be found in [6].

Pseudo-equation (1) states that agents are defined as real-time active computational entities that encapsulate data, operations and behaviour and they situate in their designated environments.

$$\text{Agent} = \langle \text{Data, Operations, Behaviour} \rangle_{\text{Environment}} \quad (1)$$

Here, data represents an agent’s state. Operations are the actions that an agent can take. Behaviour is represented by a set of rules that determine how the agent behaves include 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’. Therefore, there are two fundamental differences between objects and agents. First, objects do not contain an explicitly programmed behaviour rule. Second, objects are open to all computation entities to call its public methods without any distinctions between them.

In our conceptual model, agents are members of castes, similar to that data have types and objects have classes. The concept of caste was first proposed in SLABS [5]. A caste at time t is a set of agents that have the same structural and behavioural characteristics as stated in pseudo-equation (2).

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

Similar to the notion of class in OO, a caste acts as a structural template of agents and an organizational unit in MAS. A caste can inherit its structure and behaviour rules from a number of other castes. Figure 1 shows the structure of the description of a caste in SLABS.

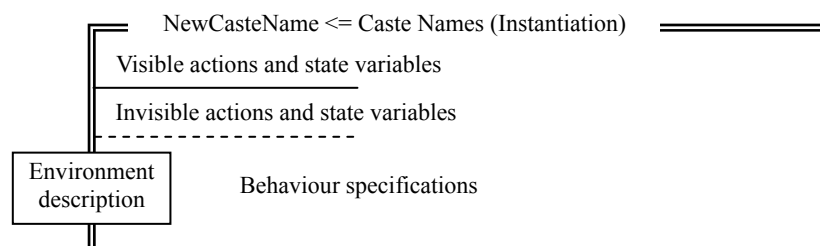


Figure 1 Caste descriptions in SLABS

An agent is an instance of a caste. However, different to the instance relationship between object and class, an agent can dynamically change its casteship. By casteship, we mean an agent's membership to a caste. To change its casteship, an agent join a caste or retreat from a caste at run time, which is a part of its behaviour capability defined by the behaviour rules. Therefore, which agents are in a caste depends on the time, hence, the subscript of t in pseudo-question (2). This feature of caste is essential for agents' autonomous behaviour, and the most fundamental property that distinguish castes from classes. We found that this feature is powerful to overcome the weakness of classes in the design and implementation of software systems [9, 10].

Equation (3) states that a MAS consists of a set of agents. Notice that, our definition of agent implies that object is a special case of agent in the sense that it has an implicit and specific fixed rule of behaviour, i.e. "executes a method when receives a corresponding message". Therefore, everything in a MAS and its environment is an agent.

$$\text{MAS} = \{\text{Agent}_n\}_{n \in \text{Integer}} \quad (3)$$

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. 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 which agents are visible. Hence, we have pseudo-equation (4).

$$\text{Environment}_t(\text{Agent}, \text{MAS}) \subseteq \text{MAS} - \{\text{Agent}\} \quad (4)$$

It is worth noting that designated environment is neither closed, not fixed, nor totally open. Since an agent can change its casteship, the set of agents in the environment of an agent may change dynamically. For example, the environment of an agent changes when the agent joins a caste that have a different environment, or other agents join the caste that is in the agent's environment. Therefore, which agents are in the environment of an agent depends on the time, hence, the subscript of t in pseudo-question (4).

The underlying conceptual model of MAS and the SLABS language do not define any specific communication language and protocol. The mechanism of communication is that an agent's actions and states are divided into two parts, the visible ones and invisible ones. Agents communicate with each other by making visible actions and changing visible state variables, and by observing other agents' visible actions and state variables. This is expressed in the pseudo-equation (5).

$$\text{Communication from agent A to B} = \text{A. Action} + \text{B. Observation} \quad (5)$$

An agent's behaviour is defined by a set of scenario-action rules that describe its responses to environment scenarios, which have the structure expressed in the pseudo-equation (6). These rules constitute the body of the agent.

$$\text{Behaviour-rule} ::= [\text{<rule-name>}] \text{pattern} \mid [\text{prob}] \rightarrow \text{event}, [\text{if Scenario}][\text{where pre-cond}]; \quad (6)$$

Please refer to [6] for details about the syntax and semantics of scenarios and rules.

3. Methodology

An attractive feature of agent-orientation is that it provides a powerful metaphor for describing, understanding and modelling information systems that contain multiple autonomous active information processing agents and information sources and receivers. CAMLE is intended to enable software engineers to use this metaphor effectively to develop such cooperative information systems systematically through smooth and ordered transitions from models of the current system and users' requirements to the designs and implementations of new systems evolutionarily.

As shown in Figure 2, we consider such evolutionary development of information system as repeated cycles of modelling the current system and its operation environment, designing a new system to be executed in a new environment (which is probably different from the current) to meet users' new requirements, and realising the system. In this process, an analyst moves from concrete concept that describes the current system to an abstract model of the current system, then analyse how to modify the current system to achieve the users' requirements, and build an abstract model of the new system. This abstract model is then refined and realized using more concrete concepts to implement the new system. This new system is subject to further modification as users' requirements change and the organizational environment and technology evolve, then a new cycle of modelling, design, refinement and implementation begins. This cycle continues as the system evolves. Therefore, CAMLE's process of agent-oriented software development can be divided into three stages: (a) the analysis and modelling of

the current information system, (b) the design of a new system as modifications to the existing system, hence the building of a model of the new system, (c) the implementation of the new system.

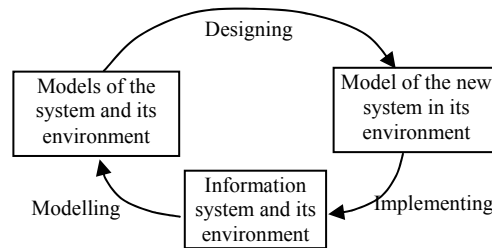


Figure 2 Evolutionary lifecycle

Models of information systems play at least two important roles in this process. One is as the transferable knowledge of the existing system so that effective analysis of the system can be made. The other is a representation of the design of a new system so that properties of the new system can be inferred and predicted. For different roles, software engineers will be interested in different properties of the model. In the former case, software engineers will be interested in the following properties: (1) *correctness* in terms of whether the model accurately represents the real system to certain abstraction level; (2) *comprehensibility* in terms that complex systems can be represented in an comprehensible way. In the latter case, in addition to these properties, software engineers will also be interested in the following properties of the new system: (1) the *correctness* of the design in terms of whether users' requirements are met; (2) the *feasibility* in terms of whether the design can be implemented and how costly the implementation will be; and, (3) the *sustainability* in terms of its easiness of maintenance and easiness of modification for the evolution of the system in the future. Therefore, it is desirable to know how much modification to the existing systems required by the new design. It is also desirable to know the *modifiability* and *reusability* of the designed system in view of possible future development. Of course, there are many other properties that other stakeholders of information system development will be interested in.

The design of the modelling language of CAMLE aims to represent information systems naturally using the conceptual model of MAS presented in the previous section and to facilitate the reasoning about such systems. Of course, a complete agent-oriented methodology must also support the implementation of a design. Although the focus of CAMLE is on modelling, in the design of the modelling language, we also considered seriously the support to the implementation of MAS from design models. Therefore, we required that our models represented in CAMLE should be able to be transformed into formal specifications of MAS in the language SLABS with sufficient details. Such a specification should be able to be implemented in a high-level programming language, ideally, in an appropriate agent-oriented programming language that is also based on the concept model discussed in the previous section.

In this paper, we focus on the analysis and modelling of existing systems. We believe that the design stage can be supported by the same modelling language, although it must involve a different set of activities and using a different process.

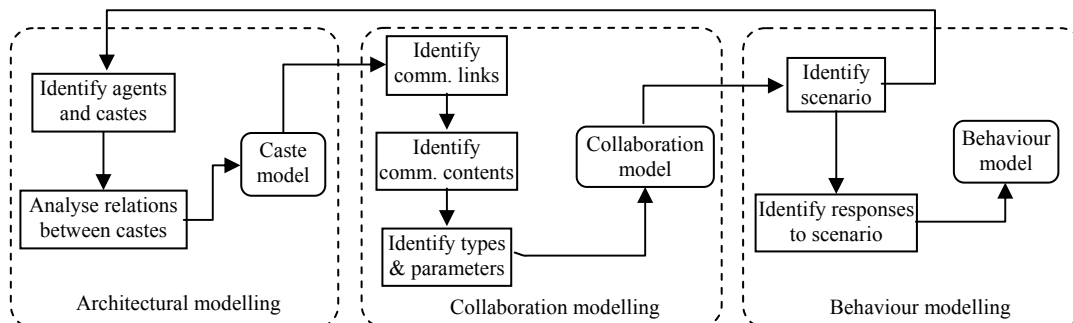


Figure 3 CAMLE's process of analysis and modelling

As shown in Figure 3, the process of agent-oriented analysis and modelling is a repeated iteration of the following activities.

1. Identify agents and their roles in the system according to their functionality and responsibilities and group agents into castes according to their functionality and responsibility;

2. Analyse inheritance and aggregation relationships between the castes;

The outcome of these activities is a caste diagram.

3. Identify the possible communication links between the agents in terms of how agents influence each other.
4. Identify the specific visible actions and state variables of each agent, and associate them with the links between the agent/caste nodes in collaboration diagrams;
5. Identify the parameters of the actions and the data type of the state and annotate the parameters to the collaboration diagram on the links between the nodes.

The outcome of these activities is a collaboration model, which captures the main inter-agent interaction.

The following activities are applied to each caste.

6. Identify the typical scenarios in the operation of the agents in each caste;
7. Analyse and describe agent's responses in each scenario.

The outcome of these two activities is a behaviour model for each caste. This may not be successful when the behaviour of the caste is too complicated. In such cases, we analyse the complicated caste as a system by applying activities (1) ~ (5) to decompose the caste into component castes. As a result, the caste diagram will be revised.

The outcome of the whole process is a caste diagram that represents the structure and organization of the whole system, a hierarchy of collaboration diagrams represents the collaboration patterns between the agents in the system, and a set of behaviour diagrams that for each caste a behaviour diagram describes the dynamic behaviour of its agents in terms of their responses to environment scenarios. As show in [8, 11], such a set of diagrams can be translated into formal specifications in SLABS fairly straightforwardly. It is worth noting that the process should not be considered as a linear sequence of activities. In each activity, we may find that it is necessary to modify other models. Hence, the consistence between the models must be checked constantly.

4. Modelling language

In CAMLE, a MAS is modelled by a set of models from different views. Each model may consist of a set of diagrams. The following subsections define the syntax and semantics of each model and discuss their uses in agent-oriented software development.

4.1. Caste Model

We view an information system as an organization that consists of a collection of agents that stand in certain relationships one to 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 scenarios. The behaviour of an individual agent in a system is determined by the 'role' it is playing. An individual can have its 'personality' (i.e. special behaviour patterns), and 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 [12]. Such an organizational structure of information systems is captured in our caste model.

Figure 4 shows the caste diagram notation and an example. A caste diagram identifies the castes of the system, indicates the inheritance and aggregation relationships between them and how agents could move from one role to another. A *caste* represents a set of agents that play a certain role in the system and have some common structural and behavioural characteristics. The *inheritance* relationship between castes defines sub-groups of the agents that have special responsibilities and hence additional capabilities and behaviours. In the example of the caste diagram in Figure 5, the members of a university are classified into three castes: students, faculties and secretaries. Students are further classified into three sub-castes: undergraduates, postgraduates and PhD students.

A *migration* relation from caste A to caste B means that an agent of caste A can retreat from caste A and then join caste B. A *participation* relation from caste A to caste B means that an agent of caste A can join caste B while remains its casteship of A. For example, as shown in Figure 4, an undergraduate student can become a postgraduate student when graduates. A postgraduate student can become a PhD student after graduation or become a faculty member. All students, including undergraduates, postgraduates and PhD students, can become a member of the alumni of the university when graduate and leave the

university. A faculty member can become a part time PhD student while remains employed as a faculty member. From this model, we can infer that an individual can be both a student and a faculty at the same time if and only if he/she is a PhD student.

An agent can contain of a number of components that are also agents. Therefore, there may have a part-whole relationship between these agents. CAMLE represents such part-whole relations through aggregation links between castes. We identified three types of part-whole relationships between agents according to the ways that bind a component agent to the compound agent and the ways a compound agent controls its components. The strongest binding between a compound agent and its component is *composition*. In such a relation, the compound agent is responsible for creation and destruction of the component. If the compound agent no longer exists, the component will not exist. This is very much similar to the composite aggregation relation in OO paradigm. The weakest binding is aggregation, in which the compound and the component are independent, so that the component agent will not be affected for both its existence and casteships when the compound agent is destroyed. This is similar to the aggregation relation in OO paradigm. The new part-whole relation we identified is called congregation. It is a binding between an agent and it components in such a way that if the compound agent is destroyed, the component agents will still exist, but they will lost the casteship as a component of the compound agent. For example, as shown in Figure 4, a university consists of a number of departments. The departments are components of the university. If the university is destroyed, departments will no long exist. There part-whole relationship between *University* and *Department* is therefore a *composite* relation. A university also 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 of the *University Member* caste. Therefore, this part-whole relationship is a congregation relation.

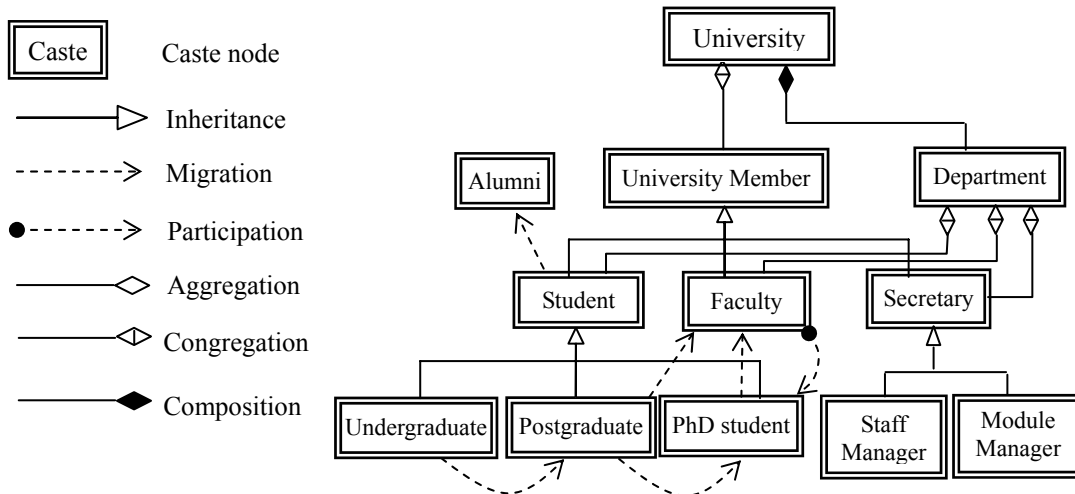


Figure 4 Caste diagram: notations and example

4.2. Collaboration Model

Agents in a MAS collaborate with each other through communications. The interplay between agents is essential to fulfill the system’s functionality. Such interactions between agents are captured and represented in a collaboration model. In CAMLE, a collaboration model consists of a set of collaboration diagrams organized hierarchically. Each diagram documents the interaction relationships between the agents of the system, or those constitute a compound agent. Each diagram contains a number of nodes that represent castes or specific agents and arrows represent communication links between castes of agents. Figure 5 gives the notations and an example 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 OO methodologies such as UML, there are significant differences in the semantics. In OO paradigm, when a message is passed from object A to object B, object B must execute the corresponding method. Therefore, what is annotated on the arrow from A to B in OO paradigm is a method of B. However, in our model, the action annotated on an arrow from A to B is a visible action of A. Moreover, in our model, agent B is not

necessarily to respond to agent A's action. This abstract model of inter-agent interactions is independent to any particular communication language and protocol, thus the attention is focused on the essential purpose of the interaction. It fits well with the autonomous nature of agents. It also reflects the shift of modelling focus from controls represented as method calls in OO paradigm to collaborations represented as signalling and observation of visible actions.

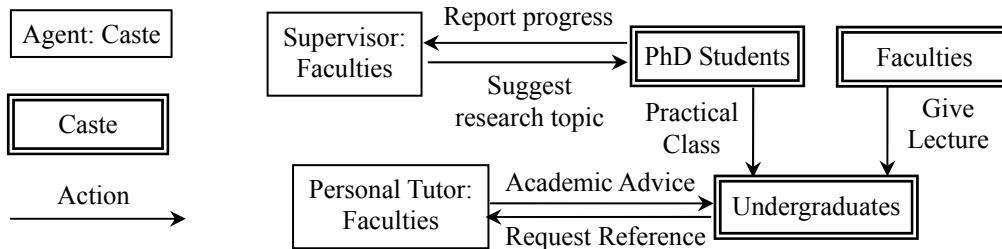
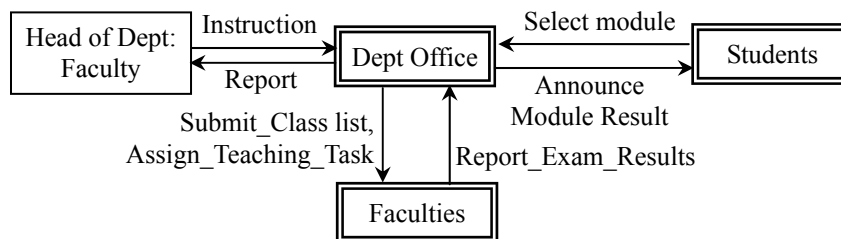


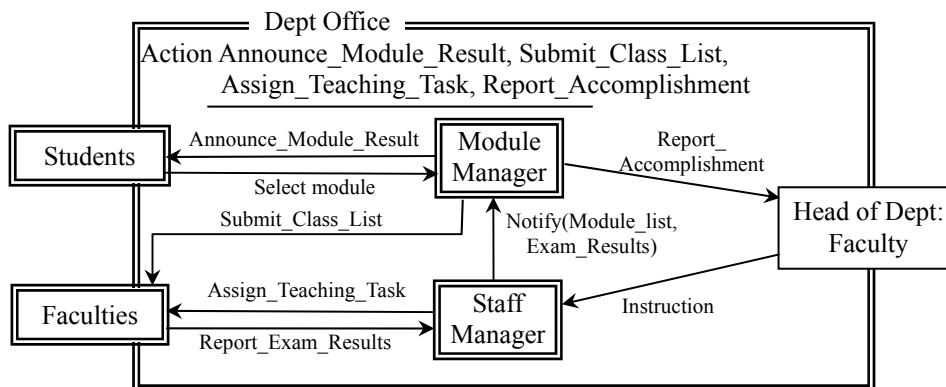
Figure 5 Collaboration diagrams: Notation and Example

For example, as shown in Figure 5, an undergraduate student will listen to faculty members' lectures and attend practical classes given by the PhD students as teaching assistants. An undergraduate student will also listen to his/her personal tutor for academic advices on the selections of modules. Personal tutors are faculty members and they will respond to his/her tutee's (an undergraduate student) request of reference for entering postgraduate course. A PhD student will also interact with his/her supervisor, who is also a member of the faculty.

In the analysis and modelling of complex systems, a component of the system may be still too complicated to analyse and design. Therefore, it is necessary to further decompose the component to lower level elements. CAMLE allows the decomposition of such a component, which is an agent or caste, in the same way as the analysis and modelling of the whole system. The only difference is that the environment of the component is clearer than the whole system, and that information available from the analysis at the higher level of abstraction is automatically carried over to the analysis and modelling of the component. A separate collaboration diagram is drawn for a caste/agent node in the higher level collaboration diagram. The lower level collaboration diagram has a boundary and have boxes drawn on the boulder representing the caste or agents in its environment.



(a) Diagram at the higher level



(b) Diagram that decomposes the Dept Office caste

Figure 6 Collaboration Diagram: Decomposition of a caste

For example, Figure 6 shows an example of the decomposition of the caste *Dept Office* in a lower level collaboration diagram. The castes *Students* and *Faculties* and the agent *Head of Dept* at the higher level that interact with the caste *Dept Office* are carried to the lower level diagram for the caste *Dept Office* as the nodes in the environment. The lower level diagram for caste *Dept Office* shows that the caste consists of two groups of agents: the *Module Managers* and *Staff Managers*. This gives the internal structure of the caste. It also shows the interactions between these component castes. Therefore, internal actions can also be introduced to the decomposed caste. Such decompositions are applied to each caste until its behaviour rules can be expressed directly. The hierarchical structure of collaboration diagrams can also be used for bottom-up design and composition of existing components to form a system.

4.3. Behaviour Model

The behaviour model describes agents' dynamic behaviour in terms of how an agent acts in certain scenarios of the environment. It aims at defining the behaviour rules for the agents of the system. A behaviour model consists of two kinds of diagrams: the scenario diagrams and behaviour diagrams.

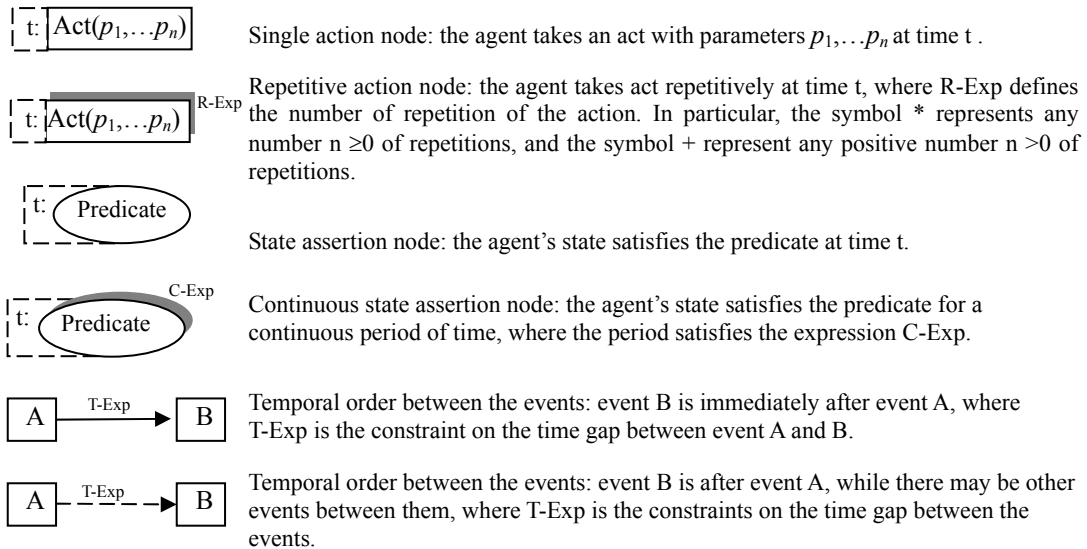
(A) Scenario Diagrams

Scenario diagrams identify and describe the typical situations in the operation of a system in form of scenarios. They are referred to in behaviour diagrams. Figure 7 depicts the format of scenario diagrams and shows the notations. Each scenario diagram defines a scenario of the situation of the system and associates a name to the situation. The quantifiers in a scenario diagram can be in any of the formats given in Table 1.

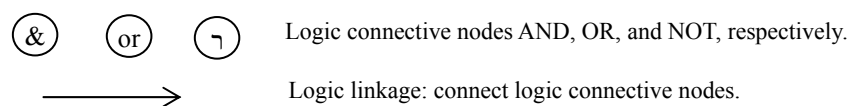
Table 1 Quantifiers in Scenario Diagrams

Quantifier format	Meaning
\forall Variable: Caste_name	For all agents in the caste
\exists Variable: Caste_name	There is at least one agent in the caste
$\exists(n)$ Variable: Caste_name	There is n agents in the caste, where n is a positive natural number.
Agent_name: Caste_name	Agent of the caste

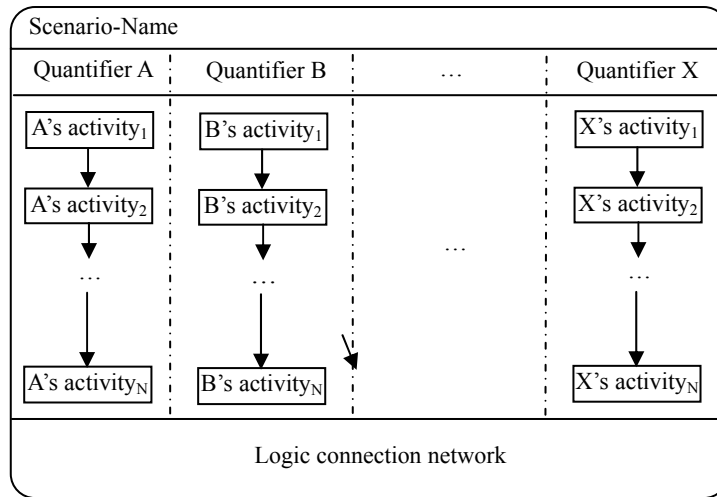
Event nodes and temporal ordering arrows:



Logic connective nodes and links:



(a) Notations



(b) Format

Figure 7 Scenario Diagram: notation and general format

(B) Behaviour Diagrams.

Each behaviour diagram describes an agent’s designed behaviour in certain scenarios. It defines a set of behaviour rules for a caste. Its notation includes the notation of scenario diagrams plus those in Figure 8.

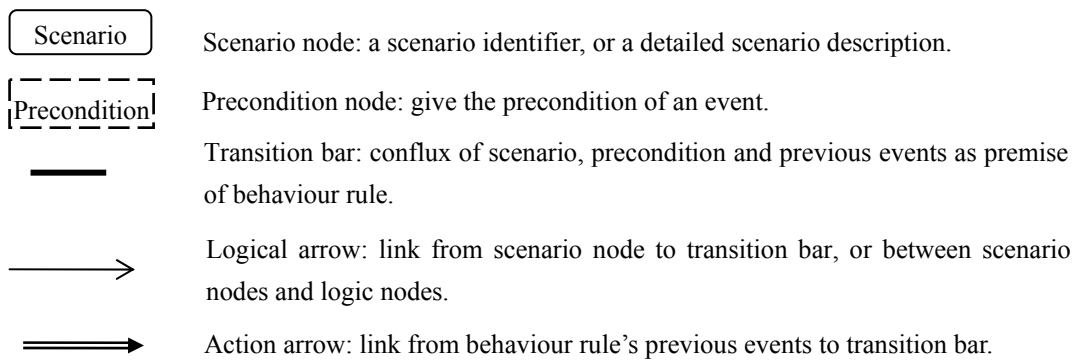


Figure 8 Additional notation for behaviour diagrams

As shown in Figure 9, 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. Figure 9 shows that after performing actions Activity 1, Activity 2 and Activity 3, the agent will take action A if Scenarios 1 or Scenario 2 is true and precondition A is satisfied. It will take action B if Scenario 3 and Scenario 4 are true and precondition B is satisfied.

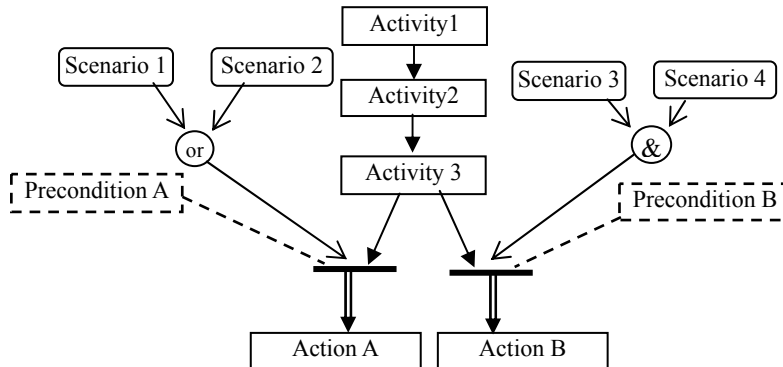


Figure 9 Behaviour Diagram: An example

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. In that case, the name for the scenario can be omitted. The behaviour diagram in Figure 10 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 will request a reference from the personal tutor for the application of a postgraduate course. If the personal tutor agrees to be a referee, the student will apply for a postgraduate course. If the department office offers a position in a postgraduate course, the student will join the Postgraduates caste and retreat from the Undergraduates caste.

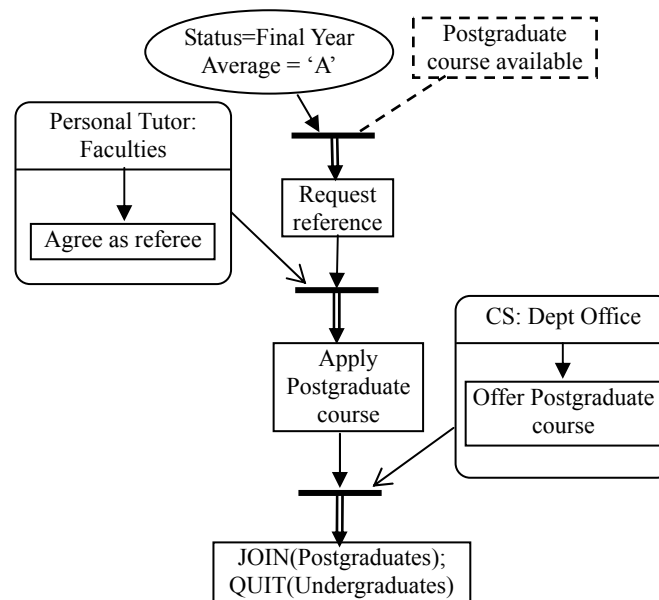


Figure 10 Behaviour Diagram: Example of undergraduate student's behaviour rules

5. Support Environment

A software environment to support the process of system analysis and modelling in CAMLE has been designed. The functionalities of the environment are:

- (1) Model construction: it consists of a set of graphical editors to support the development of models, and tools for version control and configuration management;
- (2) Model consistency check: it provides the functions of checking if models satisfy the grammar constraints, and checking the consistency and completeness between models;
- (3) Model-based analysis: it consists of a set of tools to support the proof and inference of properties of the models, and calculation of metrics defined on models;
- (4) Model-based design: it consists of a set of tools to analysis the differences between a model of the current system and a model of designed new system, to analyse if a model is consistent with a requirements, and tools to transform models in CAMLE to formal specifications in SLABS.

We have now implemented functionality (1) and (2) listed above. Screen snapshots of the tool are shown in Figure 11. We are currently developing tools for transformation of diagrammatic models into formal specifications in SLABS.

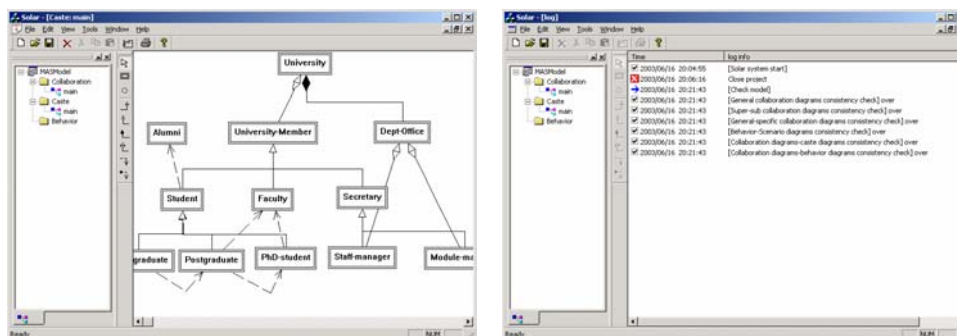


Figure 11 screen snapshots of the tool: (left) model construction; (right) consistency check

6. Conclusion

In this paper, we further developed the modelling language CAMLE proposed in [8] and [11]. It is based on the conceptual model of MAS developed in our formal specification language SLABS [6, 9]. Models represented in CAMLE can be easily translated into formal specifications in SLABS. Notations of the modelling language CAMLE defined previously in [8, 11] are revised. For example, ‘transition bar’ is introduced into behaviour diagrams. The definition of the modelling language supersedes our previous work [8, 11]. We also reported the progress in the development of the support system. A modelling environment with model construction tools and consistency checkers are developed.

The key concept of the methodology is caste, which embodies the concepts of roles and agent society. Caste is a natural extension of the concept of class. However, it has a number of new features that class does not possess, such as migration between castes, congregation part-whole relation, etc. Agents are instances of castes. They cooperate to fulfil the system’s functionality by taking the behaviours encapsulated into castes. An agent can also change its behaviour by changing its membership of certain castes to take on or to give up various roles. Agents interact with each other by observing its environment and taking visible actions or making state changes to influent other agents. The model of MAS supported by CAMLE reflects the shift of software construction focus from control to cooperation in service oriented computing.

As an extension to OO, our methodology share certain common concepts, methods and principles with OO methodologies. We believe that the following principles are the most important ones among many of such principles. First, the development of information systems starts with an analysis and modelling of the current system. Second, in the analysis and modelling of complicated systems, separation of concerns and multiple views are essential vehicles. However, agent-orientation in general and our method in particular differs from OO in the conceptual model and metaphor. OO provides a good model and metaphor to model real world objects that can naturally lead to computational representations. However, OO models consider that everything in the world is an object and objects can be hierarchically classified as in taxonomy. This model is too restrictive and unsuitable for modelling applications that involve a large number of active autonomous information processing components. Such applications are becoming increasingly important as more and more software systems are web-based and service-oriented computing technology is becoming mature. In particular, on the micro level, OO methodologies model all computational entities with objects. Their structures consist of attributes and methods. It is not sufficient to represent entities with agent characteristics, such as the autonomous behaviour. The need of an active body for objects has been widely recognized. However, modelling such ‘active objects’ alone is not adequate for developing service oriented computing because of its control centric view of software systems. On the macro level, interactions between objects can only be in the form of method-calls. It is not natural to represent the interactions of agent characteristics, for example, collaborations, negotiations, and selective observations of the environment, etc. The static relations of instance and inheritance are not enough to represent the organizational relationships in agent society. As shown in the university example, elements of a system often change their behaviours by changing their caste membership. Static classification of objects cannot naturally represent such dynamic change of behaviour. We believe that agent orientation provides a better metaphor for modelling dynamic information systems. This metaphor contains OO conceptual model and metaphor as a subset. However, it is still unclear whether or not a simple extension of notations developed for object orientation can shift the emphasis on object to agents, and whether such extension could fully realize the power of agent concept. The design of the CAMLE modelling language take a more radical approach intended to overcome the weakness of OO methodology and to maximize the power of agent concept. This is achieved through the concept of caste and treating caste as the central concept of agent oriented modelling.

A number of agent-oriented modeling techniques and methodologies have been proposed in the literature [13], including extensions of OO and knowledge engineering methodologies. A related work is AUML, which is a set of UML idioms and extensions for modelling MAS [4]. The core parts of AUML are protocol diagrams and agent class diagrams, which are extensions of UML’s sequence diagrams and class diagrams, respectively. Some ideas of AUML are similar to our conceptual model. However, AUML has no clearly defined meta-model so far. Basic concepts such as agents, agent classes and roles are left undefined. It is still unclear about what are the main features of so called agent classes in AUML. Moreover, in AUML, agent’s environment is not explicitly declared to confine the agent’s cooperators in the system. Concerning the concept of role, in AUML, ‘role’ is employed in protocol diagram to prescribe interfaces and responsibilities of the players in an interaction, while it is not related to the structure and behaviours of the agents per se. It is worth noting that the recently established Modelling

Technical Committee of FIPA is further developing AUML as an extension of UML[†].

Another related work is Gaia [1], which is a methodology for analysis and design of agent-based system. Role is the central concept in Gaia's meta-model. Similar to CAMLE, Gaia deals with both the macro level (society of agents) and the micro level (individual agents) aspects of modelling and designing. The Gaia methodology emphasizes the development of organizational structure of MAS based on the understanding of the system's requirements. The organisational structure characterizes the relationships between a collection of roles that are then turned into agent types. Because role is not a language facility in Gaia, it can not be directly mapped to a software unit in design and implementation. In Gaia, due to roles are fixed to certain agents, the abilities of agents and the services they can provide are determined at analysis and early-design stage and cannot be changed at runtime.

DESIRE is a modeling framework that enables both the specification and implementation of a system's conceptual design by explicitly modeling the knowledge, interaction, and coordination of complex tasks and reasoning capabilities in agent systems [2]. It stands for a framework for Design and Specification of Interacting Reasoning components. A key motivation of the work was to provide a rigorous design methodology adapted from software and knowledge engineering to help system verification and validation. The first and most obvious difference between our approach and DESIRE is the goal of the analysis and design process. DESIRE aims at unifying the analysis and abstract design of a MAS with concrete design and implementation in a particular agent technology or type of architecture. It makes a strong architectural commitment to a specific framework. Given the proliferation of available agent technologies, there are clearly advantages to a more general approach, as proposed here. Despite the common view of the separation between intra-agent and inter-agent functionality towards MAS, a difference exists in the modelling focus between DESIRE and CAMLE. Based on knowledge engineering, DESIRE explicates the knowledge involved in reasoning and task control with and between agents. DESIRE views an agent as composed of several task-based components, and components may be autonomous or passive.

There are several issues remaining for future work. First, we have defined a set of consistency constraints between different models and different diagrams of the same model. These constraints are automatically checked by the CAMLE environment. Work in this direction will be reported elsewhere. Second, we are designing and implementing the transformation of diagrammatic models in CAMLE into formal specifications in SLABS. Third, we are investigating software tools that support model-based design of MAS in CAMLE. The design and implementation of an agent-oriented programming language in which caste is a basic program unit is also in our agenda.

Acknowledgement

The work reported in this paper is partly supported by the National High Technology Research Programme of China under the grant 2002AA116070. The authors are most grateful to their colleagues of the AFM (Applied Formal Methods) Research Group and CAP (Computer, Agent and People) Research Group of the Department of Computing at Oxford Brookes University, especially Prof. D. Duce, Mr. D. Lightfoot, etc. and colleagues at National University of Defence Technology, especially Dr. X. Mao, Mr. Q. Yan, Dr. B. Zhou, etc. for the discussions on related topics and comments on earlier versions of the paper.

References

- [1] Wooldridge, M., Jennings, N.R., and Kinny, D., The Gaia methodology for agent-oriented analysis and design. *Autonomous Agents and Multi-Agent Systems*, 2000. 3(3).
- [2] Brazier, F.M.T., et al., DESIRE: Modeling multi-agent systems in a compositional formal framework. *Int. J. of Cooperative Information System*, 1997. 1(6): pp. 67-94.
- [3] Wood, M.F. and DeLoach, S.A., An overview of the multiagent systems engineering methodology, in *Agent-Oriented Software Engineering*, Ciancarini, P. and Wooldridge, M., Editors. LNCS, Vol. 1957, 2001: Springer, pp. 207-221.
- [4] Bauer, B., Muller, J.P., and Odell, J., Agent UML: a formalism for specifying multiagent software systems, in *Agent-Oriented Software Engineering*, Ciancarini, P. and Wooldridge, M., Editors. LNCS, Vol. 1957, Vol., 2001: Springer, pp. 91~103.
- [5] Zhu, H., Formal Specification of Agent Behaviour through Environment Scenarios, in *Formal Aspects of Agent-Based Systems*, Rash, J.L., et al., Editors. LNCS, Vol. 1871, 2001: Springer, pp.

[†] See URL <http://www.auml.org/> for details.

- 263-277.
- [6] Zhu, H., SLABS: A Formal Specification Language for Agent-Based Systems. *Int. J. of Software Engineering and Knowledge Engineering*, 2001. 11(5): pp. 529-558.
 - [7] Zhu, H., The role of caste in formal specification of MAS, in *Proc. of PRIMA'2001*. LNCS, Vol. 2132, 2001: Springer: Taipei, Taiwan, pp. 1~15.
 - [8] Zhu, H., Formal Specification of Evolutionary Software Agents, in *Formal Methods and Software Engineering, Proc. of ICFEM'2002*, George, C. and Miao, H., Editors. LNCS, Vol. 2495, 2002: Springer: Shanghai, China, pp. 249~261.
 - [9] Zhu, H., A Formal Specification Language for Agent-Oriented Software Engineering, Department of Computing, Oxford Brookes University, 2002. (Extended abstract to appear in the *Proc. of AAMAS'2003*)
 - [10] Zhu, H., Representation of roles in caste, Department of Computing, Oxford Brookes University, 2003.
 - [11] Shan, L. and Zhu, H., Analysing and Specifying Scenarios and Agent Behaviours, Department of Computing, Oxford Brookes University, 2003. (To appear in the *Proc. of IEEE/WIC IAT2003*)
 - [12] Odell, J., Parunak, H.V.D., and Fleischer, M., The Role of Roles. *Journal of Object Technology*, 2002. 2(1): pp. 39-51.
 - [13] Iglesias, C.A., Garijo, M., and Gonzalez, J.C., A Survey of Agent-Oriented Methodologies, in *Intelligent Agents V*, Muller, J.P., Singh, M.P., and Rao, A., Editors. LNAI, Vol. 1555, 1999: Springer: Berlin, pp. 317-330.