CAMLE: A Caste-Centric Agent-Oriented Modeling 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

This paper presents an agent-oriented modeling 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 modeling MAS at both macro-level for the global properties and behaviors of the system and micro-level for properties and behaviors 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.

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 modeled, analyzed 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 behavior 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 [6, 7]. In [8] and [9], a diagrammatic notation for modeling agent behaviors and collaborations was respectively developed. This paper proposes a methodology of agent-oriented software engineering called CAMLE, which stands for Caste-centric Agent-oriented Modeling Language and Environment. Caste is the classifier of agents in our modeling 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). It also allows multiple inheritances between castes. It can be used to model a wide variety of MAS concepts, such as roles, agent societies, behavior 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 behavior rules as the basic abstraction for agent's behavior 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. Behavior rules incorporating agent's perception to its environment represent the autonomy of agent's behavior. With the CAMLE language, a software system can be modeled from three perspectives following the proposed process. 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 outlines the modeling process. Section 4 presents the modeling language. Section 5 briefly reports the modeling tools. Section 6 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 behaviors, and situate in their designated environments.

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

Here, data represent an agent's state. Operations are the actions that the agent can take. Behavior 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 behavior rules. Therefore, there are two fundamental differences between objects and agents in our conceptual model. First, objects do not contain any explicitly programmed behavior 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 behavior properties of agents in the caste, and as the basic organizational units in the design and implementation of MAS. Pseudo-equation (2) states that s caste at time *t* is a set of agents that have the same structural and behavioral characteristics. The structure of caste descriptions in SLABS is shown in Figure 1.

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' behavior capability. Agents can have behavior 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 multi-agent systems naturally and to maximize the flexibility and power of AOP.

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

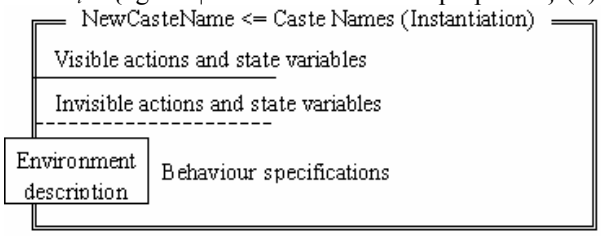


Figure 1. Caste descriptions in SLABS

Equation (3) states that 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 behavior, i.e. "executes the corresponding method when receives a message".

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

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, 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.

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

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 of the conceptual model and the language 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. As indicated above, this concept of visibility is different from the concept of scope.

A ---> B = A.Action & B.Observation (5)

3. Modeling process

CAMLE is intended to support software engineers to develop information systems systematically through smooth and orderly transitions from models of the current system and users' requirements to the design and implementation of a new system evolutionarily and collaboratively. At the highest level of abstraction, our process model is based on the current best practice of software engineering rather than a dramatic revolution. In particular, we take a model driven and evolutionary approach. We consider the evolutionary development of information systems as repeated cycles of modeling the current system and its operation environment, then designing and implementing a new system to be executed in a new environment to meet users' new requirements. In this process, engineers move from the concrete current system to an abstract model, analyze how to modify the current system to satisfy the users' requirements, and then build an abstract model of the new system. The abstract model is refined and the new system is realized. The new system is subject to further modifications as users' requirements change and the organizational environment and technology evolve. Then, a new cycle of modeling, design, refinement and implementation begin. 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 modeling of the current information system, (b) the design of a new system as modification to the existing system, hence the building of a model of the new system, (c) the implementation of the new system according the design model.

However, at a more concrete level of abstraction, our modeling and analysis process shifts the focus from object to agent and from control mechanism to collaboration. It is a repeated iteration of the following activities.

1. Identifying agents and their roles in the system according to their functionality and responsibilities and grouping agents into castes;
2. Analyzing inheritance and aggregate relations between the castes;

   The outcome of these activities is a caste model.
3. Identifying the communications between the agents in terms of how agents influence each other, and documents the communications in collaboration diagrams.
4. Identifying the specific visible actions and state variables of each agent, and associating them with the communication links between the agent nodes in collaboration diagrams;

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

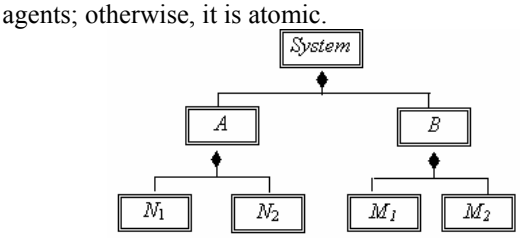
   The following activities are applied to each caste.
5. Identifying the typical scenarios in the operation of the system;
6. Analyzing and describing each agent's responses in each scenario.

   The outcome of these two activities is a behavior model for each caste. This may not be successful when the behavior of agents in the caste is too complicated. In such cases, the complicated caste should be decomposed into component castes and analyzed as a system by applying activities (1) ~ (4). As a result, the caste model and collaboration model are revised.

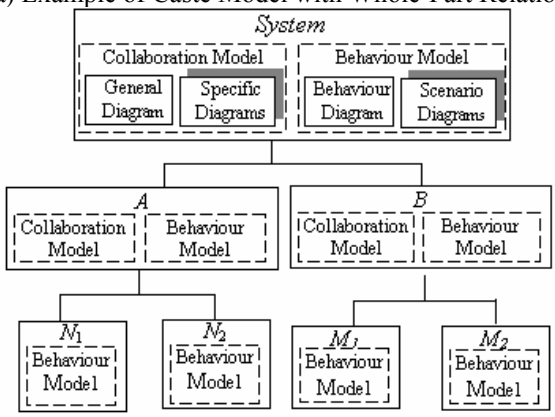
The outcome of the whole process is a caste model that represents the organization of the whole system, a collaboration model that comprises a hierarchy of collaboration diagrams and represents the communication patterns between the agents in the system, and a set of behavior models that for each caste.

4. Modeling language

CAMLE employs the multi-view principle. There are three types of models: caste models, collaboration models and behavior models. Each model consists of one or more diagrams. The caste model specifies the castes of the system and the relationships between them, such as the inheritance and whole-part relations. A caste is a compound caste if its agents are composed from a number of other agents; otherwise, it is atomic.



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



(b) Collaboration Models and Behavior models

Figure 2. Overall Structure of CAMLE Models

For example, as shown in Figure 2(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 behavior model are constructed. Atomic castes only have behavior models because they have no components thus no internal collaboration. The overall structure of a system's collaboration models and behavior models can be viewed as a hierarchy, which is isomorphic to the whole-part relations between castes described in the caste model; see e.g. Figure 2(b).

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 and checkers by the tools in the environment. 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. For the sake of space, details of the constraints are omitted here, and will be reported separately. The following subsections describe each model and discuss their uses in agent-oriented software development; see [8, 9] for more details.

4.1. 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 behavior of an individual agent in a system is determined by 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 [12]. Such an organizational structure of information systems is captured in our caste model.

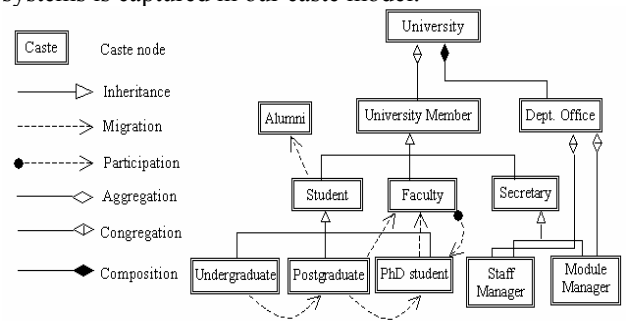


Figure 3. Caste diagram: notations and example

Figure 3 shows the notation and an example of caste diagrams. A caste diagram identifies the castes in a system, indicates the inheritance, aggregation and migration relationships between them. Migration relations specify how agents in the castes can change their casteships.

The inheritance relationship between castes defines sub-groups of the agents that have special responsibilities and hence additional capabilities and behaviors. For example, in Figure 3, 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.

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 Figure 3, 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.

An aggregate relation specifies a whole-part relationship between agents. An agent may contain of a number of components that are also agents. The former is called compound agent of the latter. In such case, there exists a whole-part relationship between the component and the compound agent, which is represented through an aggregate relation between castes. We identify three types of part-whole relationships between agents according to the ways a component agent is bound to the compound agent and the ways a 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, 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 of the compound agent. The composition and aggregation relation is similar to the composition and aggregation in UML, respectively. However, congregation is novel concept in modeling languages introduced in by CAMLE. There is no similar counterpart in object oriented modeling languages, such as UML. It has not been

guages, such as UML. It has not been recognized in the research on object-oriented modeling of whole-part relations [13]. We believe that it is important for agent-oriented modeling because of agents' basic features viz. dynamic casteship. For example, as shown in Figure 3, 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 Member and University 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.

4.2. 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 fulfill 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. Figure 4 gives the notations.

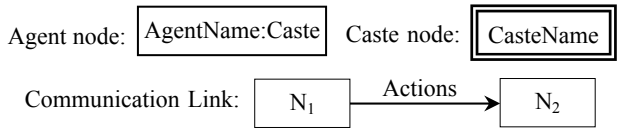


Figure 4. Notation of Collaboration Diagram

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 behavior 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 modeling focus from controls represented as method calls in OO paradigm to collaborations represented as signaling and observation of visible actions. It fits well with the autonomous nature of agents.

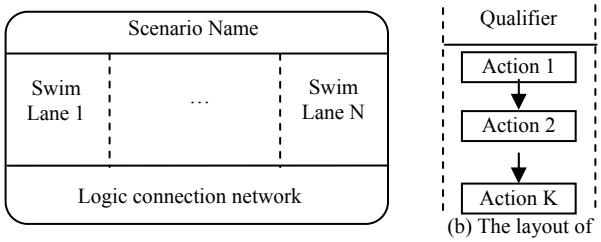
4.2.1. Scenarios of collaboration. One of the complications in the development of collaboration models is to deal with agents' various behaviors in different scenarios. By scenario, we mean a typical situation of the operation of the system. In different scenarios, agents may pass around different sequences 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. 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.

4.2.2. Refinement of collaboration models. The modeling language supports modeling 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.

4.3. Behavior model

While caste and collaboration models describe MAS at the macro-level from the perspective of an external observer, behavior model adopts the internal or first-person view of each agent. It describes an agent's dynamic behavior in terms of how it acts in certain scenarios of the environment at the micro-level. A behavior model consists of two kinds of diagrams: scenario diagrams and behavior diagrams.

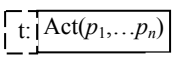
4.3.1. Scenario diagrams. We believe that each agent's perception of its environment should be explicitly specified when modeling its behavior. From an agent's point of view, the situation of its environment is characterized by what is observable by the agent. In other words, a scenario is defined by 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. Figure 5 below shows the layout of scenario diagrams.



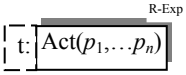
(a) The layout of scenario diagrams (b) The layout of swim lanes

Figure 5. Format of Scenario Diagram

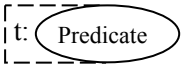
Figure 6 depicts the notations to specify visible events by nodes and temporal ordering by arrows in scenario diagrams, as well as logic connective nodes and links for the combination of situations.



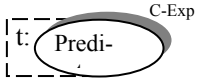
Single action node: the agent takes an act with parameters p_1,…p_n at time t .



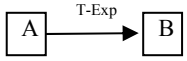
Repetitive action node: the agent takes act repetitively at time t, where R-Exp defines the number of repetition of the action.



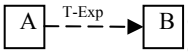
State assertion node: the agent's state satisfies the predicate at time t.



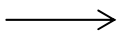
Continuous state assertion node: the agent's state satisfies the predicate for a continuous period of time, where the period satisfies the expression C-Exp.



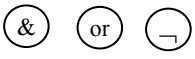
Temporal order between the events: event B is immediately after event A, where T-Exp is the constraint on the time gap between event A and B.



Temporal order between the events: event B is after event A, while there may be other events between them, where T-Exp is the constraints on the time gap between the events.



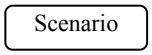
Logic links connect logic connective nodes



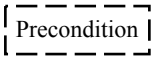
Logic connective nodes

Figure 6. Notations of Scenario Diagram

4.3.2. Behavior diagrams. Behavior diagrams describe agents' designed behavior in certain scenarios. For each caste, a behavior diagram defines a set of behavior rules. The notation of behavior diagrams includes the notation of scenario diagrams plus those in Figure 7.



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 behavior rule.

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

Figure 7. Notation for behavior diagrams

A behavior diagram contains event nodes linked together by the temporal ordering arrows as in scenario diagrams to specify the agent's previous behavior 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 behavior 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 behavior diagram, a reference to a scenario indicated by a scenario node can be replaced by a scenario diagram if it improves the readability. The behavior diagram in Figure 8 partly defines the behavior 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.

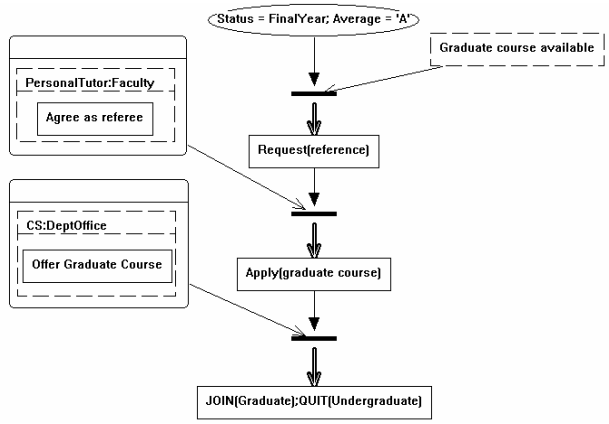


Figure 8. Behavior Diagram for Undergraduate Student

5. Support environment

A software environment to support the process of system analysis and modeling in CAMLE has been designed and implemented. CAMLE aims at representing informa-

tion 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. Such a consistent model and specification should be able to be implemented in a high-level programming language, ideally, in an agent-oriented programming language based on the same conceptual model.

Figure 9 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 the well-formedness of the user entered models, hence prevents ill-formed artifacts from being input. The partial diagram generator can generate partial models (incomplete diagrams) from existing diagrams to help users in model construction. The rules to generate partial models are based on the consistency constraints so that the generated partial diagrams are consistent with existing ones according to the consistency conditions.

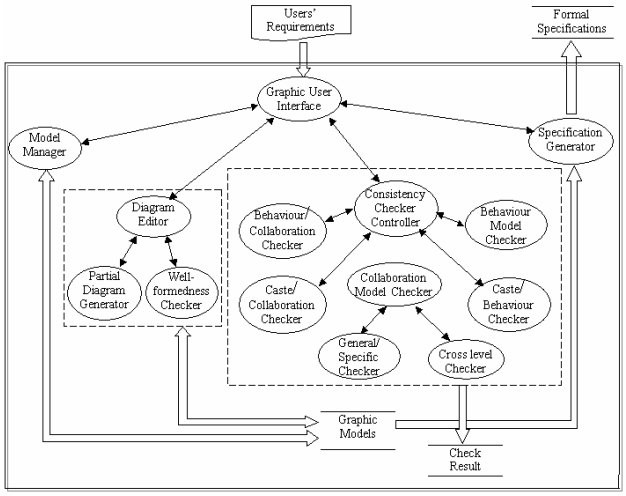


Figure 9. The Architecture of CAMLE Environment

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. The transformation from graphic models in CAMLE into formal specifications in SLABS enables engineers to analyze, verify and validate system models before the system is implemented. The specification generation tool in the CAMLE environment automatically derives a formal specification in SLABS when a model is constructed and its consistency checked. Details of the transformation rules and algorithms are omitted here for the sake of space. Figure 10 shows a screen snapshot of the tool-generated specification of a caste.

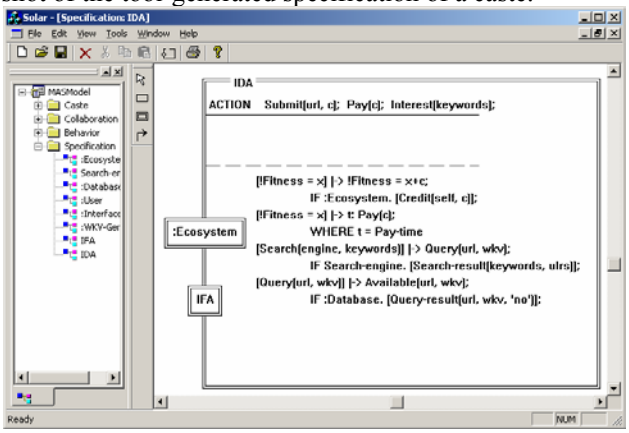


Figure 10. Screen Snapshots of Specification Generator

6. Conclusion

This paper presents the agent-oriented modeling language and environment CAMLE. It is based on the conceptual model of MAS developed and formally defined in the formal specification language SLABS. Models represented in the CAMLE language can be automatically checked for consistency and transformed into formal specifications in SLABS by the tools in the modeling environment. The modeling 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 behavior 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 distinction between CAMLE and other 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 behavior. CAMLE also has a fundamental distinction from 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. It can be used to represent a number of the concepts in agent-oriented modeling, such as roles, agent societies, normative behavior, common knowledge and protocols, etc. [6]. The caste-centric feature enables us to achieve simplicity in the design of an expressive modeling lan-

guage and efficiency in the implementation of the powerful environment.

Concerning the modeling process, different from most other methodologies such as Tropos [3], CAMLE emphasizes evolutionary and cooperative development of MAS and to reflect the shift of software construction focus from control to cooperation in service oriented computing. Gaia is perhaps one of the most mature agent-oriented software development methodologies at the moment. It does not commit to specific notations for modeling concepts such as roles, environment and interaction [1]. UML notation are widely used, e.g. in Tropos, PASSI [14] and AUML [15]. However, there are fundamental differences between agents and objects as discussed in section 2 and 3. There is no clearly defined conceptual model or meta-model underlying the uses of UML notations for agent-oriented modeling.

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 5th International Bi-Conference Workshop on Agent-Oriented Information Systems*, Melbourne, Australia, July 2003.

[2] Zambonelli, F., Jennings, NR and Wooldridge, M., "Developing multiagent systems: the Gaia Methodology", *ACM Trans on Software Engineering and Methodology* 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.*, Editors. 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, Modeling, and Application, 4th Pacific Rim International Workshop on Multi-Agents, PRIMA 2001, Taipei, Taiwan, July 28-29, 2001, Yuan, Soe-Tsyr; Yokoo, Makoto (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., Editors. LNCS, Vol. 2495, Springer, 2002, pp. 249~261.

[8] Shan, L. and Zhu, H., "Analysing and Specifying Scenarios and Agent Behaviours", *Proc. of the 2003 IEEE/WIC International Conference on Intelligent Agent Technology*. 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: $\text{Fickle}_{\text{II}}$", *ACM Trans. on Programming Language and Systems* 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*, Aug. 2003, Austria, Boszormenyi, L., & Schojer, P. (eds), LNCS 2789, Springer, 2003, pp.59-62.

[12] Odell, J., Parunak, H.V.D., and Fleischer, M., "The Role of Roles", *Journal of Object Technology*. 2(1), 2002, pp. 39-51.

[13] Barbier, F., Henderson-Sellers, B., Le Parc A., Bruel J-M., "Formalization of the Whole-Part Relationship in the Unified Modeling Language", *IEEE Trans. Software Eng*. 29(5), 2003, pp. 459-470.

[14] Burrafato, P. and Cossentino, M., **"**Designing a Multi-Agent Solution for a Bookstore With the PASSI Methodology", *Proc. of 4th International Bi-Conference Workshop on Agent-Oriented Information Systems* (AOIS-2002), Toronto, Canada, at CAiSE'02, 27-28 May 2002,

[15] 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. Also see URL: www.auml.org.