An adaptive casteship mechanism for developing multi-agent systems

Xinjun Mao* and Lijun Shan

Department of Computer Science and Technology,
National University of Defense Technology,
Changsha, Hunan Province, 410073, PR China
E-mail: xjmao@nudt.edu.cn E-mail: lijunshancn@yahoo.com
*Corresponding author

Hong Zhu

Department of Computing, Oxford Brookes University,
Oxford OX33 1HX, UK
E-mail: hzhu@brookes.ac.uk

Ji Wang

National Laboratory for Parallel and Distributed Processing,
National University of Defense Technology,
Changsha, Hunan Province, PR China
E-mail: wj@nudt.edu.cn

Abstract: In this paper, we propose an adaptive casteship mechanism for modelling and designing adaptive Multi-Agent Systems (MAS). In our approach, caste is the modular unit and abstraction that specify agents' behaviour. Adaptive behaviours of agents are captured as the change of castes during their lifecycles by executing '*join*', '*quit*', '*activate*' and '*deactivate*' operations on castes. The formal semantics of caste operations are rigorously defined. The properties of agent's adaptive behaviours are formally specified and proved. A graphical notation of caste transition diagrams and a number of rules for check consistency are designed. An example is also presented throughout the paper.

Keywords: multi-agent systems; MAS; adaptive agents; caste; adaptive casteship.

Reference to this paper should be made as follows: Mao, X., Shan, L., Zhu, H. and Wang, J. (2008) 'An adaptive casteship mechanism for developing multi-agent systems', *Int. J. Computer Applications in Technology*, Vol. 31, Nos. 1/2, pp.17–34.

Biographical notes: Xinjun Mao received his MSc and PhD in Computer Science from the National University of Defense Technology, China, in 1995 and 1998, respectively. After that, he worked at the Department of Computer Science and Technology, National University of Defense Technology, as a Lecturer, Associate Professor and Full Professor in 1998, 2000 and 2005, respectively. His current research interests are software engineering, agent theory and technology, distributed computing and engineering technology for self- systems.

Lijun Shan is a PhD candidate at the Department of Computer Science and Technology of the National University of Defense Technology, where she obtained BSc and MSc in Computer Science in 2001 and 2003, respectively. Her research interest is in software development methodology, in particular, the agent-oriented methodology, service-oriented software engineering and model-driven software development. She has developed an agent-oriented modelling language (CAMLE) and implemented its automated modelling environment. She has published 13 papers in refereed international conference proceedings, two papers in international journals and two peer-reviewed book chapters.

Hong Zhu is a Professor of Computer Science at Oxford Brookes University, where he chairs the Applied Formal Methods Research Group. He obtained his BSc, MSc and PhD in Computer Science from Nanjing University, China, in 1982, 1984 and 1987, respectively. He was with Nanjing University from 1987 to 1998. He was a research fellow at Brunel University and the Open University, UK, from 1990 to 1994. He joined Oxford Brookes University in 1998. His research interests are in the area of software engineering, including software development methodology, software testing, agent technology, automated software development tools. He has won a number of honours and prizes in China for his research achievements, which include the

Premier's Award of Distinguished Young Scientists in China awarded by the National Natural Science Foundation of China in 1997 and the Cheung Kong Scholar Professorship by the Ministry of Education of China in 2000.

Ji Wang received his BSc, MSc and PhD in Computer Science from the National University of Defense Technology, in 1987, 1990 and 1995, respectively. He is currently a Professor and Deputy Director of National Laboratory for Parallel and Distributed Processing at National University of Defense Technology. His research interests include software engineering, programming languages and systems, and distributed computing.

1 Introduction

Agent orientation has recently emerged as a new software development paradigm for developing complex software systems that operates in a dynamic environment such as the internet (Jennings, 2001). Several industrial experiences have testified to the advantages and potentials of using agents in manufacturing processes, web services and web-based computational markets, and distributed network management. Case studies have demonstrated the possibility of exploiting agents and Multi-Agent Systems (MAS) as enabling technologies for a variety of future scenarios, i.e., pervasive computing, grid computing, semantic web, web service, etc. (Zambonelli and Omicini, 2004). A wide range of potential applications of agent technology are also predicated in Luck et al. (2005). From the software engineering point of view, agent orientation provides a higher level of abstractions and a complete set of metaphors for information system modelling that are more suitable than object orientation to tackle the complexity in the development of software systems where autonomous behaviours in dealing with dynamic environment are essential.

In the past few years, with the increasing acceptance and expectation of agent orientation as an emerging software engineering paradigm, there have been a great number of efforts in the research on developing methodologies for complex MAS. Agent-Oriented Software Engineering (AOSE) has become an active research area in agent-based computing (Zambonelli and Omicini, 2004). A number of methodologies, modelling languages, programming languages and CASE tools or environments have been proposed, such as Gaia (Zambonelli et al., 2003), ROADMAP (Juan et al., 2002), AUML (Bauer et al., 2001), Tropos (Giunchiglia et al., 2002), CAMLE (Shan and Zhu, 2005; Zhu and Shan, 2005; Zhu, 2006) and SLABS (Zhu, 2001, 2003a, 2003b), etc.

Nevertheless, the research in AOSE is still in its early stages. Several challenges need to be faced before AOSE can deliver its promises and become a widely accepted and practically usable paradigm for the development of complex systems (Zambonelli and Omicini, 2004) that are typically open, dynamic, unpredictable, hierarchically structured but confined by global constraints (Mao and Yu, 2004). We believe that, to be a successful general-purpose software engineering paradigm, new language facilities and computational mechanisms must be developed to enable high level abstractions to be smoothly and naturally transformed into concrete language facilities and efficiently implemented in a systematic, robust, reliable and repeatable fashion (Shan and Zhu, 2005). Moreover, the methods must be universally applicable.

A key feature of agents that enables them to deal with dynamic environments is their adaptation capability. For example, in a social organisation, an agent may be assigned to different roles at different times during an execution and thus performs different functions and demonstrates different behaviours to satisfy the design objectives (Odell et al., 2002). In the sequel, we will use *adaptive agents* to denote agents that are capable of adapting to different behaviours at run time. Such agents can be found in many applications such as enterprise information systems, electronic commerce, medical care systems, and military systems. Therefore, mechanism and language facilities should be developed to support the specification, analysis, design and implementation of adaptive agents and MAS.

However, existing agent-oriented methodologies have not provided enough supports to develop such systems that naturally take such advantages, especially the modelling, specification, design and implementation of such dynamic behaviours in a systematic way. It is no surprise that it is extremely difficult to develop MAS with dynamic behaviours (Zhu, 2003a, 2003b). This paper addresses this problem by presenting an adaptive casteship mechanism based on the language facility caste proposed in Zhu and Lightfoot (2003), proposing a graphical notation and a number of rules for modelling adaptive behaviours and checking models' properties (e.g., conflict and inconsistency) of adaptive agents in MAS, and discussing how properties of such systems can be reasoned based on the formal definition of the adaptive casteship mechanism and the extended language facility.

The remainder of the paper is organised as follows. Section 2 describes an example of adaptive agents that is used throughout the paper to illustrate our proposed concepts, mechanism and facilities. Section 3 briefly reviews the basic concepts and methods of the caste-centric methodology of agent-oriented software development that is related with the adaptive casteship mechanism. Section 4 proposes the adaptive casteship mechanism to enrich the caste language facility, formally defines the mechanism and the semantics of its operations on castes. Section 5 presents the approach to reasoning about the adaptive behaviours of agents. Section 6 proposes a graphical notation of caste transition diagram and a number of consistency rules for modelling and analysing adaptive agents based on the adaptive casteship mechanism. Section 7 illustrates our approach with an example. Section 8 gives a comparison with related work. Section 9 concludes the paper with a discussion of future works.

2 An example of adaptive agents

In order to understand adaptive agents and the requirements on adaptive casteship mechanism, in this section, we examine an example of an information system to illustrate the basic characteristics of adaptive agents and discuss the issues in modelling and designing such systems. The example will be used throughout the paper to illustrate the proposed concepts, mechanism and approach. A complete study of the example using the adaptive casteship mechanism and modelling language facilities proposed in this paper will be given in Section 5.

Suppose that an information system is to be developed for a university to support the management of the university's members such as students, staff, secretary and related affairs such as registration, course selection, authorisation, etc. The system will provide personal assistances to the members of the university so that each member can participate in the operations of the university efficiently and effectively. Depending on the type of roles that a member plays in the university, such as a student or a staff member, a member can take certain actions in the operation of the information systems, access a certain subset of the information stored in the systems, and must obey a certain set of rules that confines the member how to fulfil his tasks. Each member in the university is therefore supplied with a 'personal assistant', which is actually a software agent that stores the personal information about the member, the permitted actions that the member can take according to his role, the behaviour rules that describe how members behave in the university, as well as the personal preferences and private information.

A member of the university is inevitably related to other members of the university. Some of the relationships between members are determined by the roles that the members play rather than just a personal issue. Therefore, the personal assistant agent must also support the collaborations between the members. Such a structure of the information systems that consists of a number of personal assistant agents fits well with the current trend in the development of computing technologies and their applications represented by the rapid growth of mobile computing devices such as handhold computers, notebook computers, mobile phones with computing capabilities, ubiquitous computing through the internet platform, etc.

In order to understand the dynamic and adaptive behaviours of agents, now let us consider the following specific scenarios for the members of the university in the example.

The behaviour of the agent depends on the roles that it plays in the organisation

Obviously, the behaviours of undergraduate and postgraduate students in the university are different. For example, when a student, say Alex, acts as an undergraduate student, he can take undergraduate courses but can not take courses only for graduate students. Registering as a Teaching Assistant (TA) is only permitted for postgraduate students and prohibited for undergraduate students.

A member of the university may change roles while remaining as a member of the organisation

As in almost all organisations, a member in the organisation often changes role in order to adapt to the organisation context and exhibit flexible behaviours. For example, Alex is going to graduate from the university. After graduation, he gets an offer from the university to study for a Master degree. The graduation does not mean that Alex will change his identity or leave the university. Actually, he is still a member of the university. It just manifests that he will play a new role in the organisation of university. By changing role, he will no longer be an undergraduate but a postgraduate. Therefore, by changing role, Alex will change his behaviours in the university organisation context. Alex's personal assistant agent must be able to help Alex to change his roles in the organisation of university during its lifecycle.

A member of the university may play multiple roles at the same time

As in almost all organisations, a member in the organisation often plays multiple roles at some moment in order to exhibit diversity of behaviours. For example, when Alex, becomes a postgraduate, he may also take a job as TA or Research Assistant (RA), which is a part of his offer of the postgraduate studentship. Therefore, in his first year, Alex is not only a postgraduate but also a TA. Alex's personal assistant agent must be able to help Alex to play two or more different roles.

The changes of roles occur not only as moving from one role to another, but also as in the form of temporarily leaving a role and then returning to the role

As in almost all organisations, a member in the organisation often temporarily leaves a role and then returns to the role, which means that the behaviour specification of the role will be suspended and unavailable when the agent leaves from the role, and will be resumed when the agent returns to the role. For example, after one year's study, Alex decides to take one year's industrial placement job in order to gain some work experience. He thus leaves the university for one year and then comes back to school and carries on his study. This does not mean that Alex will retreat from the role of postgraduate student forever. Instead, during the work placement year, he suspends his behaviour as postgraduate temporarily. He is officially still a registered postgraduate during this year, but he will not go to the university to take courses or conduct researches. Alex's personal assistant agent must be able to help Alex to suspend by keeping all personal information such as the study program, scores of passed or failed modules, credits earned, etc., and suspending the permitted activities as well as some restrictions on his performances while he is on leave, and resume his study by reactivating such information and activities after his returning to the university.

The above scenarios substantially exhibit the dynamic and adaptive behaviours of agents in many applications. In order to support the development of such applications, the method should be proposed to model, analyse and design the above dynamic and adaptive behaviours in a natural way.

3 Caste-centric methodology

This section briefly reviews the caste-centric methodology of agent-oriented software development and its basic concepts that are related to the adaptive casteship mechanism presented in the paper. More details can be found in Zhu (2003a), Shan and Zhu (2006) and Shan et al. (2006).

3.1 Conceptual model

In our conceptual model, the basic unit that forms a system is agent. An agent is defined as computational entity that encapsulates data, operations and behaviours, and situates in its designated environments.

$Agent = <Data, Operations, Behaviour>_{Environment}.$

Here, *Data* represents an agent's state. *Operations* are the actions that the agent can take. *Behaviour* is described by a set of rules that determine how 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 agent itself. Consequently, a MAS consists of agents and nothing but agents, as stated in the following.

$MAS = \{Agent_n\}, n \in Integer.$

The classifier of agents is called caste. The notion of caste was originally presented in SLABS intending to deal with the limitation of object technology (Zhu, 2003a, 2003b). Here, we regard caste as the basic abstraction to specify agents' behaviours and the elementary modular unit to design and implement MAS. As a modular language facility, caste serves as a template that describes the structure and behaviour of agents.

A state space defined by a caste is represented as a set of state variables. Each action consists of an action identifier and may contain a number of parameters. The state space and the set of actions are usually divided into two kinds: visible ones and invisible (or internal) ones. When an agent takes a visible action, it generates an event that can be observed by other agents in the system. An agent taking an internal action generates an event that can only be perceived by its components. Similarly, the value of a visible data can be observed by other agents, while the value of an internal data can only be observed by its components. For the sake of simplicity, in this paper, we are not concerned with the visibility issues.

While a set of castes represent various types of participants, the structure of the problem domain is captured with certain relationships between castes, including part-whole relations, inheritance (is-a relation) and migration (membership-shift relations). In particular, there are two types of migration relations, i.e., migrate and participate. A *participate* relation from caste *A* to caste *B* means that agents of caste *A* can join caste *B* without quitting from caste *A*. A *migrate* relation from caste *A* to caste *B* means that agents of caste *A* can join caste *B* and quit from caste *A*.

Although the relationship between agents and castes is similar to the relationship between object and class and the relationship between data and type, none of the terms like *instance*, *member*, *classifier* and *type* that are commonly used to represent the relationship between object and class accurately represents the relationship between agent and caste. The main difference is that an agent can take actions to *join* a caste or *quit* from a caste at run time, and consequently, it obtains or loses the structural and behavioural features defined by the caste. Agents' capability of dynamic joining and quitting from castes is called dynamic casteship (Zhu, 2003a).

Based on the conceptual model of MAS presented above, a caste-centric agent-oriented methodology has been developed for the development of MAS. It consists of a process model called growth model and a set of agent-oriented languages and software tools that support various development activities in the process. In requirement analysis phase, a modelling language and environment CAMLE (Shan and Zhu, 2005) supports the analysis and design of MAS. The semi-formal models in CAMLE can be automatically transformed into formal specifications in SLABS (Zhu, 2001, 2003a, 2003b), which is a formal specification language designed for formal engineering of MAS. In implementation phase, MAS are implemented directly in an agent-oriented programming language called SLABSp (Wang et al., 2005).

3.2 Modelling MAS with CAMLE

In our methodology, modelling aims at representing the users' requirements with a set of agents at various granularities and organising the agents into an information system. The key activities in the modelling and analysis phase include the following.

- First, identify the agents and castes of agents in the system as well as the relationships between them. The artifact produced in this phase is a *caste model* for the system from the perspective of system architecture.

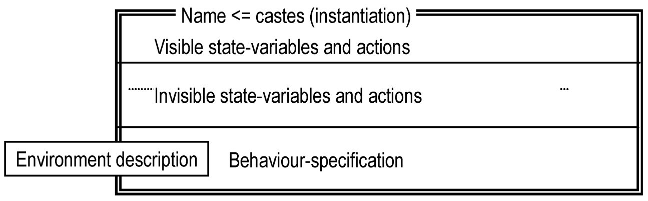
- Second, identify the agents' interaction patterns in various scenarios, and produce a set of *collaboration models* for the system from the perspective of dynamic behaviour.
- Third, elaborate and specify how its agents perform actions and/or change states in typical scenarios for each caste so that a set of behaviour rules can be assigned to the caste. The artifact produced in this phase is a set of *behaviour models*, each associated to a caste in the system.

The result of the modelling is a system model comprising a set of diagrams that represent the system from various views and at different levels of abstraction.

3.3 *Specifying MAS with SLABS*

One of the most appealing features of agent technology is its natural way to modularise complex systems in terms of multiple interacting autonomous components. This feature is supported by the language facility caste in SLABS for specifying MAS. It bridges the gap between graphic modelling and implementation in the development process of agent-oriented information system. At the specification phase, a system model in CAMLE as the output of the modelling phase can be transformed into formal specification of the system. The specification not only lays the ground for future system implementation, but also facilitates further formal analysis of the system design. The formal definition of the SLABS language and its meta-model can be found in Zhu (2003a). Figure 1 shows the format of caste specification in SLABS.

Figure 1 Caste specification in SLABS


Name <= castes (instantiation)
Visible state-variables and actions
Invisible state-variables and actions
Environment description
Behaviour-specification

In the specification language SLABS, a behaviour rule is specified in the form of "*pattern* → *event*, [*scenario*] [where *pre-cond*]", where *scenario* and *pre-condition* are optional. The *pattern* in a rule describes the pattern of the agents' previous behaviours. The *scenario* part describes the situation in the environment in terms of the behaviours of other agents in its environment. The *where-clause* is the pre-condition of the action to be taken by the agents. The *event* describes the action to be taken or state to be in when the scenario happens and the pre-condition is satisfied. In modelling language CAMLE, behaviour rules are described by behaviour diagrams that define the scenarios, patterns and events in various types of nodes linked by arrows (Shan and Zhu, 2005). In programming language SLABSp, behaviour rules are expressed in a structural control statement that consists of a list of scenario expression and action statement pairs so that the statement is executed only if the scenario expression is true (Wang et al., 2005).

3.4 *Implementing MAS with SLABSp*

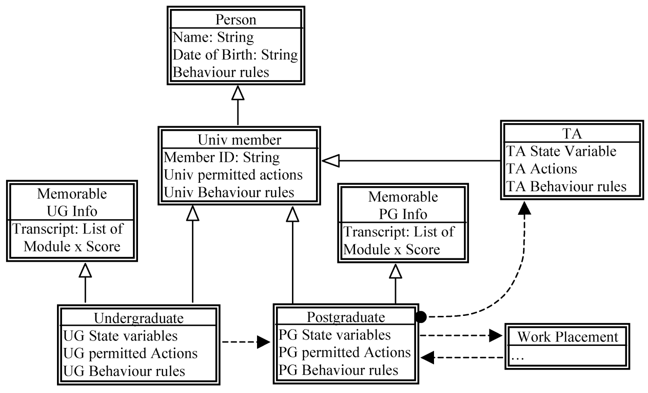
A distinctive feature of our agent-oriented methodology of information systems is that we aim at the direct implementation of information systems with a novel agent-oriented programming language based on caste language facility. Such a programming language can significantly narrow the gap between specification and implementation. We have designed and implemented an agent-oriented programming language SLABSp (Wang et al., 2005), which is based on Java and extends Java with three key concepts and language facilities, i.e., caste, scenario and environment description. The design and implementation of SLABSp demonstrates that caste and scenario are feasible as programming language facilities.

3.5 *Developing adaptive agents with dynamic casteship*

It is worth noting that the notion of caste as the encapsulation of data, operation, behaviour rules and environment description in terms of the agents in collaboration meets the requirements induced from the structures and functionalities of the agents and MAS described in the example in Section 2.

For the first scenario described in the example where the behaviour of agent depends on the roles that it plays in the organisation, the structure and behaviour specification of a role can be specified and encapsulated as a caste. This means that caste can act as a fundamental abstraction and modular unit to describe what state and actions that an agent has, what environment that the agent is situated in and what behaviour rules that the agent should respect when the agent plays some role. For the second scenario, Alex will no longer be an undergraduate since then and start to be a postgraduate. This means he must give up certain access to the information, certain actions that he was granted as an undergraduate student as well as certain relationships with other members of the organisation such as his personal tutor, etc. This will also give him some new capability of actions, such as the access to the graduate's labs, new personal information, a master degree student ID number, and new relationships to other members, such as a professor as his supervisor, etc. The change of Alex's role in the university must also be reflected in his personal assistant software agent in terms of the changes of the restrictions on his access to the information system, the set of permitted actions, and the set of rules that confine his activities, etc. What is more important is that the personal assistant agent must carry over all the personal information such as his academic records, personal details and personal preferences, etc. rather than start from scratch. The dynamic change of roles that agents play can be modelled by the language facilities proposed in Shan and Zhu (2005). Suppose that each role in the university information system is modelled by a caste, which encapsulates the data related to the role, the actions the agent of the role can take, the set of behaviour rules that the agent playing the role must obey and the agents in collaboration with. Moving from one role to another can be modelled by changing the agent's casteships to a caste to another caste. The CAMLE-like diagram in Figure 2 illustrates how dynamic casteship can be used to model role changes from undergraduate to postgraduate, where the dash line arrow is the migration from one caste to another. For the third scenario in the example in Section 2, playing multiple roles at the same time can be modelled as agent with some casteship(s) participating another caste. This means that agent can bind to two or more castes by participating.

Figure 2 Illustration of the modelling of role changes in dynamic casteship



However, the caste model presented in Figure 2 has some drawbacks. Note that, the caste that represents undergraduate students is split into two castes in order to enable a part of the information that Alex gains in the study to be remembered in his whole lifetime. Similarly, the caste of postgraduate students is also split into two. In fact, many other castes should also have to be split into two, such as the TA and Work Placement caste. It is quite universal that the participation in a role will leave memorable information and knowledge. Splitting a caste into two parts to model what are memorable and what is forgettable has two particular disadvantages in the modelling and designing adaptive agents. First, it results in an unnecessarily higher complexity of the model. The level of abstraction is not high enough to enable software developers to concentrate on more important issues. Second, it is structurally awkward and unnatural to view an agent as a compound of two parts, one as the forgettable and that other as remembered.

Another problem that must be addressed in the modelling and designing adaptive agents is that while new roles can be adapted by an agent during execution, two roles may have conflictions of interests and conflictions in the behaviour rules, etc. For example, in the university information system, it is prohibited for any student to play undergraduate and graduate roles at the same time. The inconsistency may occur when agent binds to multiple roles. The modelling and design method should support the identification and resolution of such conflictions.

Addressing these problems, this paper proposes an adaptive casteship mechanism as an extension to the dynamic casteship mechanism. We will also investigate how to analyse the consistency of adaptive MAS by defining a set of consistency notions, devising a graphic notation to represent transitions between configuration states of adaptive agents and imposing consistency constraints on the configuration states.

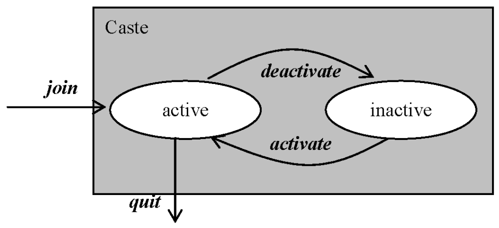
4 Adaptive casteship mechanism

This section formally defines the adaptive casteship mechanism.

4.1 Informal introduction to the mechanism

In the proposed adaptive casteship mechanism, an agent can not only change its casteship to a caste dynamically, but also distinguish the states of the casteship as either *active* or *inactive*. An agent can take actions '*activate*' and '*deactivate*' to change the state of casteship at run time. Figure 3 shows the transition of the agent's casteships and their states. The following explains the meanings of the states of casteship.

Figure 3 The adaptive casteship mechanism



Active casteship

An agent has an active casteship to a caste means that the agent obtains all the structural and behavioural features defined by the caste, and these structural and behavioural features will influence the agent's behaviour. In other words, the agent can take actions defined in the action part of the caste according to the behaviour rules defined by the caste, which can be a change to the agent's internal state that belongs to the part of the state space defined by the caste. When agent takes action to *join* a caste, the casteship will initially be active.

Inactive casteship

An agent has an inactive casteship to a caste means that the agent obtains all the structural and behavioural features defined by the caste, but can not change the state variables and take actions defined in the caste while it still holds its values of the state variables defined by the caste. The behaviour rules of the inactive caste will not affect the agent's behaviour. It does not observe the agents in its environment defined by an inactive caste either. However, the state variables defined by an inactively bounded caste are accessible. Moreover, when the agent becomes actively casteship to the caste again, the values of the state variables are resumed to that when the agent's casteship to the caste last changed to inactive.

The adaptive casteship mechanism provides a natural way to model agents' adaptive behaviours in complex systems. It is more suitable than dynamic casteship mechanism for modelling, analysis and realisation of the adaptive agents such as one defined in Section 2.

It is worth noting that an agent's action of activating, inactivating, joining or quitting a caste is also determined by the behaviour rules defined by the caste that the agent currently has active casteship to. Therefore, the behaviour rules specified in each caste explicitly declare what castes the agent can *join* and *quit* as well as *activate* and *deactivate* and in what situation to do so. In the university example, Alex graduates and becomes a postgraduate student must be the result of executing a behaviour rule that enables an undergraduate to quit from the caste *Undergraduate* and to join the caste of *Postgraduate*. His suspension/resuming of study must be the result of executing behaviour rules of the *Postgraduate* caste that enables him to deactivate/activate the casteship to Postgraduate.

4.2 Formal model of Multi-Agent Systems (MAS)

In Zhu (2003a), the caste-centric formal model of MAS has been formally defined. The following extends the formal model to enable the adaptive casteship mechanism to be specified. For the sake of simplicity, we have omitted some aspects of the original model, such as the caste inheritance and scenario definition, so that we can focus on the adaptive casteship mechanism.

In our meta-model of MAS, agent is defined as an autonomous and persistent computational entity that is situated in certain environment and can take actions in order to fulfill its design objectives. Each agent consists of four parts: data, actions, behaviours and environments. That is, the structure of an agent is a 4-tuple, i.e., $Agent = <D_A, A_A, B_A, E_A>$, where D_A represents an agent's state, A_A is the actions that agent can take, B_A is the behaviour rules that determine how agent behaves in the context of its designated environment E_A. It is worth noting that these parts may change during the execution of the agent. Such changes are realised by the adaptive casteship of an agent to castes.

A caste defines a set of structural features and behaviour patterns for agents. A caste also contains four parts. Thus, it is a 4-tuple, i.e., $Caste = <D, A, B, E>$, where D defines a state space that the agents can have, A defines a set of actions that agents can take for the operation on the state space, B defines a set of behaviour rules specifying how agents behave in terms of when to take an action and how to update its state in the context of their designated environment, and E defines an environment that consists of a set of agents that have influence on agent's behaviour. When an agent holds a casteship to a caste, it obtains the state space and the capability to take an action defined by the caste, becomes in collaboration and influenced by the agents specified in the caste's environment definition and must obey the behaviour rules specified in the caste. For example, in the university example, we can identify and define two castes to represent the roles of undergraduate and postgraduate students, respectively. Each caste has its associated data, actions, behaviour rules and environments.

Definition 1: A model M of MAS consists of two parts $<MAS, CASTE>$, where $MAS = \{a_1, a_2, \ldots, a_n\}$ is a finite set of agents and $CASTE = \{c_1, c_2, \ldots, c_m\}$ is a finite set of castes.

Agents behave continuously and autonomously. A time moment is an element in a time index set T, which is defined as the set of natural numbers. Let MAS_t be the set of agents in the system at time moment t. The casteship of agent a at time moment t to caste c is denoted by $a \in_t c$. We write $CASTE(a, t)$ to denote the set of castes that agent a belongs to at moment t, i.e., $CASTE(a, t) = \{c \mid a \in_t c\}$. In our adaptive casteship mechanism, the castes that agent binds may be either in active state or inactive state. Therefore, we write $CASTE^A(a, t)$ to denote the set of active castes that agent a belongs to at moment t, and $CASTE^I(a, t)$ to denote the set of inactive castes that agent a belongs to at moment t. Thus, $CASTE(a, t) = CASTE^A(a, t) \cup CASTE^I(a, t)$. We assume that $CASTE^A(a, t) \cap CASTE^I(a, t) = \varnothing$, for all agents a in MAS and time moments t. This means that for at any moment t, a caste that an agent binds is either in active state or in inactive state.

An agent's state space at some moment depends on the castes that the agent binds to at that time moment and their status. Let $D^A_{a,t} = \cup_{\{c \in \mathrm{CASTEA}(a,t)\}} Dc$ denote the state spaces of agent a at moment t based on active castes that it binds to at moment t, $D^I_{a,t} = \cup_{\{c \in \mathrm{CASTEI}(a,t)\}} Dc$ denote the state spaces of agent a at moment t based on inactive castes that it binds to at moment t. Similar to the above, we define $A^A_{a,t} = \cup_{\{c \in \mathrm{CASTEA}(a,t)\}} A_c$ as the action set of agent a at moment t based on active castes that it binds to at moment t, $A^I_{a,t} = \cup_{\{c \in \mathrm{CASTEI}(a,t)\}} A_c$ the action set of agent a at moment t based on inactive castes that it binds to at moment t. In particular, we assume that for any caste c, A_c includes '*join*', '*quit*', '*activate*' and '*deactivate*' operations on castes. Similarly, let $B^A_{a,t} = \cup_{\{c \in \mathrm{CASTEA}(a,t)\}} B_c$ the behaviour rule set of agent a at moment t based on active castes that it binds to at moment t, $B^I_{a,t} = \cup_{\{c \in \mathrm{CASTEI}(a,t)\}} B_c$ the behaviour rule set of agent a at moment t based on inactive castes that it binds to at moment t; $E^A_{a,t} = \cup_{\{c \in \mathrm{CASTEA}(a,t)\}} E_c$ the environment of agent a at moment t based on active castes that it binds to at moment t, $E^I_{a,t} = \cup_{\{c \in \mathrm{CASTEI}(a,t)\}} E_c$ the environment of agent a at moment t based on inactive castes that it binds to at moment t.

Definition 2: The configuration of agent a at moment t is based on the castes that it binds to and is defined as $s_{a,t} = s^A_{a,t} \times s^I_{a,t},$ where $s^A_{a,t} = D^A_{a,t} \times A^A_{a,t} \times B^A_{a,t} \times E^A_{a,t}$

denotes the configuration of agent a at moment t based on active castes that it binds to, and $s^I_{a,t} = D^I_{a,t} \times A^I_{a,t} \times B^I_{a,t} \times E^I_{a,t}$ denotes the configuration of agent a at moment t based on inactive castes that it binds to.

Let $S_a = \{s_{a,t} \mid t \geq t_0$ for any moment t in T, where t_0 is the moment at which agent a is to be created$\}$ the set of all possible configurations of agent a. The configuration of MAS at moment t is $S_{MAS,t} = \prod_{(a \in MAS)} s_{a,t}$. We write $S_{MAS} = \cup_{(t \in T)} S_{MAS,t}$ to denote the set of all possible configurations of MAS.

Definition 3: A run r of a MAS is a mapping from time T to the set S_{MAS}. The behaviour of a MAS is defined by the set R of all possible runs. For any given run r of MAS, a mapping r_a from T to S_a is a run of agent a in the context of r, where for any moment t, $r_a(t)$ is the restriction of $r(t)$ on S_a. In the sequel, we use $R_a = \{r_a \mid r \in R\}$ to denote the behaviour of agent a in the system.

We assume that agent takes actions step by step, which means if agent takes action *act* at moment t, then the action will be completed at $(t + 1)$ moment.

4.3 Formal definition of the adaptive casteship mechanism

In this section, we rigorously define the adaptive casteship mechanism and the semantics of caste operations. Formally, we write '$M \models_{r,t} \varphi$' to denote that the model M of MAS and its run r satisfies formula φ at moment t, and '$\models \varphi$' to denote that formula φ is valid for any model of MASs and their runs at all time. Let '< >' be the dynamic operator to represent action execution, intuitively, '<a: *act*>' indicates that agent a takes action *act*. '$\boldsymbol{U}$' is the *until* temporal operator and '$\psi \boldsymbol{U} \varphi$' means that φ will be eventually satisfied and before that ψ is satisfied. '●' denotes the next temporal operator.

Definition 4: The semantics of the above temporal operators are defined as follows.

- $M \models_{r,t} \psi \boldsymbol{U} \varphi$ iff $\exists\, t' \in T$: $(t \leq t')$ and $(M \models_{r,t'} \varphi)$ and $(\forall t''$: $t \leq t'' < t' \Rightarrow M \models_{r,t''} \psi)$
- $M \models_{r,t} \bullet\ \varphi$ iff $M \models_{r,t+1} \varphi$.

A number of special predicates are introduced to specify the castes of agent and their status. '*BindCaste*(a, c)' means that agent a holds a casteship to caste c, '*Active*(a, c)' denotes that agent a's casteship to caste c is active. '*Inactive*(a, c)' means that agent a's casteship to caste c is inactive. Formally, their semantics are defined as follows.

Definition 5: For all agents a, castes c and moments t,

1 $M \models_{r,t} BindCaste(a, c)$ iff $c \in CASTE(a, t)$

2 $M \models_{r,t} Active(a, c)$ iff $c \in CASTE^A(a, t)$

3 $M \models_{r,t} Inactive(a, c)$ iff $c \in CASTE^I(a, t)$.

Definition 6: $M \models_{r,t}$ <a: *join*(c)> iff

$c \notin CASTE(a, t)$ and $CASTE^A(a, t + 1) = CASTE^A(a, t) \cup \{c\}$ and $CASTE^I(a, t + 1) = CASTE^I(a, t)$

Definition 6 means that agent a executes action '*join*(c)' at moment t successfully, if and only if, at moment t, c is not the caste of agent a, and at moment $(t + 1)$ agent a's casteship to caste c is active, and the action execution will not change the inactive castes of agent a.

Theorem 1: *join operation has the following properties.*

1 $\models$ <a: *join*(c)> $\rightarrow \bullet(Active(a, c))$

2 $\models$ <a: *join*(c)> $\wedge\ Inactive(a, c_1) \rightarrow \bullet Inactive(a, c_1)$.

Property (1) in Theorem 1 means that if agent *a joins* some caste c at moment t, then the agent will hold a casteship to caste c actively in the next moment. Property (2) means that the *join* operation will not change the inactive castes of agent. The theorem follows Definitions 5 and 6 straightforwardly.

Definition 7: $M \models_{r,t}$ <a: *quit*(c)> iff

$c \in CASTE^A(a, t)$ and
$CASTE^A(a, t + 1) = CASTE^A(a, t) \backslash \{c\}$ and
$CASTE^I(a, t + 1) = CASTE^I(a, t)$.

Agent a *quits* caste c at moment t, if and only if, agent a holds a casteship to caste c actively at moment t, and at moment $(t + 1)$ agent a does not hold a casteship to castes c and the action execution will not change the inactive castes of agent a.

Theorem 2: *quit operation has the following properties.*

1 $\models$ < a: *quit*(c)> $\rightarrow \bullet\ (\neg BindCaste(a, c))$

2 $\models$ < a: *quit*(c)> $\wedge$ Inactive(a, $c1$) $\rightarrow \bullet Inactive(a, c1)$

Property (1) in Theorem 2 manifests that if agent *a quits* some caste c, then the agent will unbind to the caste c when action is completed. Property (2) means that the *quit* operation will not change the inactive castes of agent. The theorem follows Definitions 5 and 7 directly.

Definition 8: $M \models_{r,t}$ <a: *deactivate*(c)> iff

$c \in CASTE^A(a, t)$ and
$CASTE^A(a, t + 1) = CASTE^A(a, t) \backslash \{c\}$ and
$CASTE^I(a, t + 1) = CASTE^I(a, t) \cup \{c\}$.

Agent *a deactivates* caste c at moment t, if and only if, agent a holds a casteship to caste c actively at moment t, and at moment $(t + 1)$ the status of caste c will be changed from active to inactive.

Theorem 3: *deactivate operation has the following properties*

1 $\models$ <a: *deactivate*(c)> $\rightarrow Active(a, c)$

2 $\models$ <a: *deactivate*(c)> $\rightarrow \bullet\ (Inactive(a, c))$.

Property (1) in Theorem 3 states that if agent a *deactivates* a caste c, then the agent must hold a casteship to c actively before deactivation. Property (2) shows that if agent a *deactivates* some caste c, then the state of its casteship to c will be changed to inactive when the action is completed. The theorem follows Definitions 5 and 8 directly.

Definition 9: $M \models_{r,t} \langle a\text{: } activate(c)\rangle$ iff

$c \in CASTE^I(a, t)$ and $CASTE^I(a, t+1) = CASTE^I(a, t) \backslash \{c\}$ and $CASTE^A(a, t+1) = CASTE^A(a, t) \cup \{c\}$.

Agent a *activates* some caste c at moment t, if and only if, agent a holds a casteship to caste c inactively at moment t, and at moment $(t+1)$ the state of its casteship to c will be changed from inactive to active.

Theorem 4: *deactivate operation has the following properties.*

1 $\models \langle a\text{: } activate(c)\rangle \rightarrow Inactive(a, c)$

2 $\models \langle a\text{: } activate(c)\rangle \rightarrow \bullet(Active(a, c))$.

Property (1) in Theorem 4 states that if agent a activates its casteship to caste c, then the agent must hold an inactive casteship to caste c before the activation. Property (2) shows that if agent a activates caste c, then the state of its casteship to c will be changed from inactive to active. The theorem follows Definitions 5 and 9. The proof is straightforward.

5 Reasoning about adaptive behaviours

This section is devoted to the reasoning about adaptive behaviours as specified in adaptive casteship mechanism to ensure adaptive agents are well-designed. We will first introduce some notions about the structure of adaptive agents so that the reasoning about well-defined adaptive behaviour can be performed.

Definition 10: For any $c_1, c_2 \in CASTE$, if the behaviour rules of c_1 enables an agent that hold a casteship to caste c_1 to join caste c_2, then we call c_1 can directly reach c_2, write $c_1 \Rightarrow c_2$.

We assume that the directly reachable relationship between castes is irreflexive, which means any caste is not permitted to be joined again by an agent when it has already had a casteship to the caste. We write $DReach(c) = \{c' \mid c \Rightarrow c'\}$ to denote the directly reachable castes set of c, and $DReach(a, t) = \cup_{\{c \in CASTEA(a,t)\}} DReach(c)$ to denote the directly reachable caste set of agent a at moment t.

For example, if the behaviour rule of caste *undergraduate* explicitly declares that when undergraduate student passes the entrance examination, he may join the caste of *postgraduate*, then *postgraduate* $\in$ *DReach*(*undergraduate*). The directly reachable relationship between castes does not necessarily satisfy transitive property. If c_1 can directly reach c_2, the agent that has casteship to caste c_1 is possible to *join* caste c_2 in its run when the scenario and the pre-condition specified in the behaviour rule are satisfied.

Definition 11: Let $c \in CASTE$, the reachable castes set $Reach(c)$ of caste c is recursively defined as follows.

1 if $c_1 \in DReach(c)$, then $c_1 \in Reach(c)$.

2 if $c_1 \in Reach(c)$ and $c_2 \in Reach(c_1)$, *then* $c_2 \in Reach(c)$.

Obviously, the reachable relationship between castes is transitive. It defines the possible castes that agent can hold casteships to during its lifecycle. If c_1 can reach c_2, then the agent that has the casteship to caste c_1 is possible to *join* caste c_2 in its run. Let $Reach(a, t) = \cup_{\{c \in \mathrm{CASTEA}(a,t)\}} Reach(c)$ the reachable caste set of agent a at moment t.

Definition 12: Let $c_1, c_2 \in CASTE$. If caste c_1 and c_2 are strictly not permitted for any agent a to hold casteships at the same time to govern the agent's behaviours simultaneously, then we say that caste c_1 and c_2 are conflict, written as $c_1 \uparrow c_2$. Let $V \subseteq CASTE$ be a subset of castes, if for all castes $c_1, c_2 \in V$, c_1 and c_2 are not conflict to each other, i.e., $c_1 \uparrow c_2$ is not true, then we say that the caste set V is consistent. For an agent a and moment t, if $CASTE^A(a, t)$ is consistent, we say that agent a is consistent on its casteships at moment t. If agent a is always consistent on its casteships at all moments in its run, we say that agent a is coherent.

The conflict relationship between castes is dependent on the applications. For example, if the university does not permit any student to be an *undergraduate* and *postgraduate* simultaneously, then the castes *undergraduate* and *postgraduate* are in conflict. Hence, the caste set {*undergraduate*, *postgraduate*} is not consistent. It is obvious that the conflict relationship between castes is reflexive, symmetric, but not transitive.

As the casteship of an agent can change from time to time and agent is possible to *join* any caste in its reachable castes set, agents should avoid being inconsistent on its castes during its run. Therefore, when developing adaptive MAS based on the adaptive casteship mechanism, the conflict requirements should be explicitly specified and the conflict relationship of castes should be detected and eliminated. For example, since {*undergraduate*, *postgraduate*} is inconsistent according to application requirement, an agent that has casteship to *undergraduate* must firstly quit the caste *undergraduate* before it joins the caste *postgraduate*.

Definition 13: For agent a in *MAS*, a is called *adaptive*, if and only if, there are time moments t_1 and t_2 in T $(t_1 \neq t_2)$ such that $CASTE^A(a, t_1) \neq CASTE^A(a, t_2)$ or $CASTE^I(a, t_1) \neq CASTE^I(a, t_2)$. Otherwise, agent a is called *static*.

The above definition manifests that the adaptive agent will change its casteships in its lifecycle. In the example defined in Section 2, it is obvious that agent Alex is a typical adaptive agent.

Lemma 1: *Let t_0 be the moment that agent a is created, if CASTE(a, t_0) is consistent and a is a static agent, then agent a is coherent.*

The proof is straightforward, because according to the Definition 13, the caste set of static agent will never be changed in agent's lifecycle.

We assume that there is no other action in agents except '*join*', '*quit*', '*deactivate*' and '*activate*' that can change the casteships of agent. Formally, the assumption is specified as follows. For any agent a, caste c, action act and time moment t, if $M \models_{r,t} <a: act>$ and $act \notin \{join, quit, deactivate, activate\}$, then $CASTE^I(a, t) = CASTE^I(a, t + 1)$ and $CASTE^A(a, t) = CASTE^A(a, t + 1)$).

Definition 14: An agent a is *rational* about caste operations, if and only if, it satisfies the following properties.

1 $M \models_{r,t} <a: join(c)> \rightarrow M \models_{r,t} \neg BindCaste(a, c)$

2 $M \models_{r,t} <a: quit(c)> \rightarrow M \models_{r,t} Active(a, c)$

3 $M \models_{r,t} <a: deactivate(c)> \rightarrow M \models_{r,t} Active(a, c)$

4 $M \models_{r,t} <a: activate(c)> \rightarrow M \models_{r,t} Inactive(a, c)$.

Formula (1) in Definition 14 means that when an agent intends to *join* a caste, it should have no casteship to the caste. Formula (2) states that when an agent intends to *quit* a caste, the agent should have already an active casteship to the caste. Formula (3) means that when an agent intends to *deactivate* a caste, the agent should have already held an active casteship to the caste. Formula (4) states that when an agent intends to *activate* a caste, then the agent should have an inactive casteship to the caste.

Definition 15: An agent a is faithful to caste operations, if and only if, agent a intending to *join* caste c at some moment t implies that the caste c is directly reachable for agent a at moment t. Formally, if $M \models_{r,t} <a: join(c)>$, then $c \in DReach(a, t)$.

The above definition shows that a faithful agent will depend on the directly reachable caste set to join a new caste. Note that, an agent can only take actions defined by the castes that the agent has active casteship to, i.e., for any agent a, action act and moment t, $M \models_{r,t} <a: act> \Rightarrow \exists c$: $c \in CASTE^A(a, t)$ and $act \in A_c$.

Lemma 2: *For any faithful agent a, caste c and moment t, if $c \notin Reach(a, t)$ and $c \notin CASTE(a, t)$, then for any $t' > t$: $c \notin CASTE(a, t')$.*

Proof: Assuming that there is $t' > t$ such that $c \in CASTE(a, t')$. Then, according to our assumption that there is no other action except '*join*', '*quit*', '*deactivate*' and '*activate*' that can change the casteships of agent and definition 6, there must be a moment $t < t'' < t'$, $M \models_{r,t''} <a: join(c)>$. As agent is faithful, according to Definition 15, $c \in DReach(a, t'')$ which is conflict with the pre-condition of the lemma.

The lemma states that if a caste is not reachable for agent a at moment t, then the agent cannot hold a casteship to the caste in its future run.

Definition 16: An agent a is consistent about caste operations, if and only if, it satisfies the following conditions.

1 if agent a intends to join caste c at moment t, there is no caste $c' \in CASTE^A(a, t)$: $c \uparrow c'$;

2 if agent a intends to activate caste c at moment t, there is no caste $c' \in CASTE^A(a, t)$: $c \uparrow c'$.

The above definition shows that a consistent agent will consider and avoid the conflict problem when it changes the casteships in its lifecycle.

Lemma 3: *For any agent a, caste c and moment t,*

1 *if agent a executes action 'join(c)' and there is no caste $c' \in CASTE^A(a, t)$: $c \uparrow c'$, then the caste set of agent a is consistent when the 'join' action is completed;*

2 *if agent a executes action 'activate(c)' and there is no caste $c' \in CASTE^A(a, t)$: $c \uparrow c'$, then the caste set of agent a is consistent when the 'activate' action is completed.*

The lemma shows that if an agent intends to *join* or *activate* a caste and the caste to be joined or activated does *not* conflict with any castes that agent a already has active casteships, then when the operation is completed the caste set of agent is consistent. The lemma follows Definition 12, Definitions 6 and 9 directly.

Lemma 4: *For any agent a that is consistent about caste operations and moment t, if Reach(a, t) is consistent, then the agent will be consistent in its future run.*

The lemma is a corollary of Lemma 2 and Definition 16. The proof is straightforward.

Definition 17: We call that an agent a is reachable at moment t, if and only if, for any caste $c \in CASTE(a, t)$, $\exists t' < t$: $M \models_{r,t'} <a: join(c)>$ and $c \in DReach(a, t')$.

The definition manifests that reachable agent always binds to castes in its directly reachable caste set.

Definition 18: For any caste operation *act* and agent *a*, we say that *act* is safe for agent *a*, if and only if, agent *a* is always consistent and reachable when performing *act*.

Theorem 5: *If agent* a *is rational, faithful and consistent about caste operations, then the 'join', 'quit',* 'activate' *and* 'deactivate' *caste operations are all safe for agent* a, *i.e., for any caste operation* $act \in \{join, quit, activate, deactivate\}$, *caste c and moment t,*

1 if $M|=_{r,t} <a: act(c)>$ and agent *a* is consistent at moment *t*, then when *act* is completed agent *a* is still consistent;

2 if $M|=_{r,t} <a: act(c)>$ and agent *a* is reachable at moment *t*, then when *act* is completed agent *a* is still reachable.

Proof 1: For any model M of MAS and its run *r*, moment *t* and caste *c*, if $M|=_{r,t} <a: act(c)>$ and $act \in \{join, activate\}$, then according to Definition 16, there is no caste $c' \in \mathrm{CASTE}^{A}(a, t)$: $c \uparrow c'$. Therefore based on Lemma 4, agent *a* is consistent when the '*join*' or '*activate*' action is completed. For $act \in \{quit, deactivate\}$, as agent *a* is consistent at moment *t*, then for any castes c_1, $c_2 \in CASTE^{A}(a, t)$, there is no $c_1 \uparrow c_2$. Therefore, according to the formal semantics definitions of *quit* and *deactivate* operations, when *quit* and *deactivate* operations are completed at moment t', the active castes of agent *a* at moment t' is $C^{A}(a, t') = C^{A}(a, t) \setminus \{c\}$. It is obvious that for any castes $c_1, c_2 \in CASTE^{A}(a, t')$, there is no $c_1 \uparrow c_2$. Therefore, agent *a* is consistent when *act* is completed.

Proof 2: For any model M of MAS and its run *r*, moment *t* and caste *c*, if $M |=_{r,t} <a: act(c)>$ and agent *a* is reachable at moment *t*, then for $act \in \{activate, deactivate\}$, according to semantics definitions of *activate* and *deactivate* operations, when *activate* and *deactivate* operations are completed at moment $t + 1$, the caste set of agent *a* at moment $t + 1$ is $CASTE(a, t + 1) = CASTE(a, t)$. As agent *a* is reachable at moment *t*, therefore agent *a* is still reachable at moment $(t + 1)$. For $act \in \{quit\}$, according to semantics definitions of *quit* operation, when *quit* operation is completed at moment $t + 1$ the caste set of agent *a* at moment $t + 1$ is $CASTE^{A}(a, t') = CASTE^{A}(a, t) \setminus \{c\}$. As agent *a* is reachable at moment *t*, therefore agent *a* is still reachable at moment $(t + 1)$. For $act \in \{join\}$, according to semantics definitions of *join* operation, as agent *a* is faithful about caste operations, therefore according to Definition 15, $c \in DReach(a, t)$. So, when *join* operation is completed at moment $t + 1$, agent *a* is reachable at moment $t + 1$.

6 Modelling and analysing adaptive agents

Adaptive agents and MAS typically have complex behaviours. A visual modelling language will be helpful to model and analyse the dynamic behaviours in such systems. In this section, we propose a diagrammatic notation called transition diagrams, which describes dynamic change of agent's casteships during its lifecycle. The semantics of the visual notation is based on the adaptive casteship mechanism defined in Section 4. This section also introduces the constraints on transition diagrams for specifying and analysing adaptive agents with a number of model properties.

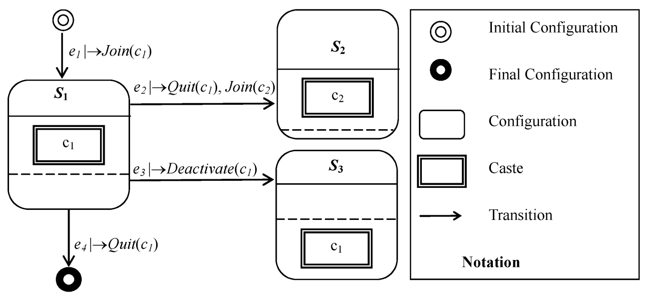
6.1 Transition diagram

In this section, we introduce transition diagram, present its graphic notation and its meanings. The abstract syntax is then formally defined to facilitate the definition of constraints in Section 6.2.

6.1.1 The graphic notation

The notation of transition diagrams is an extension to CAMLE language. In CAMLE, information about agents' dynamic transition is partly depicted in caste diagram and behaviour diagrams. Caste diagram describes the possible transition paths between castes. Behaviour diagrams define behaviour rules for each caste to describe how agents change their casteships in what scenario. But, when software engineers have particular interests on some adaptive agent, a transition diagram is helpful to explicitly describe the *configuration* of the agent at various time moments and the *transitions* between the configurations by changing the agent's casteships. Figure 4 shows the notation and format of transition diagram.

Figure 4 Notation and format of transition diagram



Configuration specifies the state of an agent at a time moment in terms of its active and inactive casteships. In a transition diagram, a configuration is depicted as a round-corner rectangle that comprises three parts partitioned by a solid line and a dash line. The top part is annotated with the *name* of the configuration. The middle part contains the set of *active casteships* of the agent, and the bottom part contains the set of *inactive casteships*. There are two special kinds of configuration nodes. One is *initial configuration* represented by hexagon, which specifies the start point of an agent's lifecycle. The other is *final configuration* represented by dark hollow circle, which denotes the end point of the agent's lifecycle. When modelers need to depict some configurations as the start or end of the transition chain in the diagram but do not care about their details, initial configuration and end configuration can also be used.

A transition describes how an agent transforms from one configuration to another when some condition is satisfied in its lifecycle. In transition diagrams, a directed edge represents a transition relation between a pair of configurations. A transition is annotated with a *transition rule*, which represents the condition that results in the transition. The format of a transition rule is "*pattern* $|\rightarrow$ *transition-actions* [if *scenario*] [where *pre-cond*]", which means that when the agent's previous behaviour matches the *pattern*, behaviour of the agent's environment matches the *scenario* and the *pre-condition* is satisfied, the agent will take the *transition-actions* which are a sequence of actions in the atomic action set {*join, quit, deactivate, activate*}. Transition rule is a special type of behaviour rule of caste. The execution of the rule will result in the change of agent's configuration from the source to the destination of the edge. For example, Figure 4 shows that when the pattern matches e_2, the agent will execute '$quit(c_1), join(c_2)$' , which leads to the agent's configuration change from S_1 to S_2. Note that, a transition rule in a transition diagram is not necessarily a complete behaviour rule. It is acceptable as long as the transition-actions part is specified, while the other parts stating why the transition is invoked are optional. Transition diagrams, as one viewpoint to MAS, aim at correct design of adaptive agents with respect to certain properties. Analysing the properties of an adaptive agent can be realised through checking certain properties of the agent's configurations and the transitions between the configurations in transition diagrams. Therefore transition modelling does not care the detailed structure and behaviour rules of castes or agents. The complete behaviour rules of each caste including transition rules should be specified in the behaviour diagram for the caste.

In order for transition diagrams to be well organised with acceptable complexity, each transition diagram is regulated to model one certain caste of agents. Thus a transition diagram usually contains one non-trivial configuration (not initial situation or final situation) as the start-point of a set of transition chains, and the transition chains have only one hop. For example, in the transition diagram in Figure 4, the start configuration is S_1, thus the 'leading role' in the diagram is caste C_1. The diagram describes that an agent may join caste C_1 under condition e_1, (i.e., the transition from the initial situation to S_1), then agents of caste C_1 may migrate to C_2 under condition e_2 (i.e., from S_1 to S_2), or deactivate C_1 under condition e_3 (i.e., from S_1 to S_3), or quit C_1 under condition e_4 (i.e., from S_1 to S_2). Note that how the agents of C_2 will change its casteship to other castes is not concerned in this diagram.

6.1.2 The abstract syntax

Graphic modelling with transition diagrams not only helps to explicitly describe specific design information about each adaptive agent, but also facilitates to further investigate special properties of the agents. To achieve this purpose, we use the GEBNF (Graphically Extended BNF) proposed in Zhu and Shan (2006)to rigorously define the type system and abstract syntax of the transition diagrams. Table 1 shows the notation of GEBNF.

Table 1 GEBNF notation

Notation	*Meaning*	*Examples and explanation*
<X>	X is a concept or a type of entities in the model	<*Model*> and <*Diagram*> represent the concepts of models and diagrams, respectively
X ::= Y	X is defined as Y	<*Model*> *::=* <*Diagram*>* : a model is defined as a number of diagrams
X*	Repetition of X (include null)	<*Diagram*>*: the entity consists of a number *N* of diagrams, where $N \geq 0$
X+	Repetition of X (exclude null)	<*Diagram*>+: the entity consists of a number *N* of diagrams, where $N \geq 1$
X | Y	Choice of X and Y	<*Actor node*>|<*Use case node*> means that the entity is either an actor node or a use case node
X , Y	X and Y, the union of X and Y	<*Actor node*>,<*Use case node*>: an entity that consists of an actor node and a use case node
[X]	X is optional	[<*Actor*>]: element of actor is optional
X Y	Order pairs consists of X and Y	<Actor node> <Use case node>: an element that consists of an order pair of an actor node and a use case node
/X/	An annotation field named as X	/*Use case name*/: the annotation field called use case name
X : Y	The type of X is Y	/*Use case name*/*: Text* : the type of the annotation use case name is text
(X)	Parenthesis	It is used to change the preferences of the expression
'abc'	Terminal element, the literal value of a string	'*extends*': the literal value of the string 'extends'
Text [!*F*]	Predefined type Text with syntax specified by F, where F is a BNF	*Text*: a text in any format; *Text ! <object name> ':' <class name>* : the text that consists of an object name and a class name separated by a colon

In GEBNF, we can define the type system and abstract syntax of transition diagram as follows.

6.2 Constraints on transition diagrams

Based on the adaptive casteship mechanism described in Section 4, we define some constraints for transition diagrams to support analysing and checking the properties of adaptive agents. Modellers can analyse adaptive agents' properties, such as rational, faithful, consistent, and reachable properties, through checking the agents' transition diagrams against the constraints. Diagrams satisfying such constraints represent well-defined agents with the desirable properties.

```
<Transition diagram>::= /Title/: Text ! <Caste Name>, <Configuration>+, <Transition>*
<Configuration>::= <General Configuration> | <Initial Situation> | <Final Situation>
<General Configuration>::= /Name/: Text ! <Configuration Name>, <Active Casteship>,
                           <Inactive Casteship>
<Active Casteship>::= <Caste Node>*
<Inactive Casteship >::= <Caste Node>*
<Caste Node>::= /Name/: Text ! <Caste Name>
<Transition>::= /Guard/: Text ! <Transition Rule>, <Configuration> <Configuration>
<Transition Rule>::= <Event>, <Action>+
<Action>::= ('Join' | 'Quit' | 'Activate' | 'Deactivate' ) '(' <Parameter> ')'
<Parameter>::= <Caste Name>
```

6.2.1 Notation for defining constraints

We use the formal notation proposed in Zhu and Shan (2006) to define the constraints based on the type system and abstract syntax given in Section 6.1. The following summarises the formal notation, which is a first-order logic for defining structural properties of graphic models.

ML is the modelling language. Let φ be an n-ary operator defined on the type $t_1, t_2, \ldots, t_n$, that results in a value of type t. Let ρ be an n-ary relation defined on the type $t_1, t_2, \ldots, t_n$.

- *Expressions* are formed by finite applications of the following constructions.
  - *Variables* of various types are expressions of their own types.
  - *Constants* are expressions of their own types.
  - $\varphi(e_1, e_2, \ldots, e_n)$ is an expression of type t, if $e_1, e_2, \ldots, e_n$ are expressions of types $t_1, t_2, \ldots, t_n$, respectively.
  - $e.f$ is an expression, if e is an expression of type t and f is a field defined by the language ML for the type t. The type of $e.f$ is ft, if the type for field f is defined to be of type ft by ML.
  - $e.t$ is an expression, whose value is the set of the elements of type t in e, where type t is defined in ML.
  - *Type*(e) is an expression if e is an expression. The value of *Type*(e) is the type of e.
- *Statements* are formed by finite application of the following constructions.
  - $\rho(e_1, e_2, \ldots, e_n)$ is a statement, if $e_1, e_2, \ldots, e_n$ are expressions of types $t_1, t_2, \ldots, t_n$, respectively; in particular, $e_1 = e_2$ and $e_1 \in e_2$ are statements, if e_1 and e_2 are expressions.
  - *Type*(e) = t is a statement, if e is an expression and t is a type name.
  - $\neg\rho$, $\rho_1 \Rightarrow \rho_2$, $\rho_1 \Leftrightarrow \rho_2$, $\rho_1 \wedge \rho_2$, and $\rho_1 \vee \rho_2$ are statements, if ρ, ρ_1 and ρ_2 are statements. Here the symbols $\neg$, $\Leftrightarrow$, $\Rightarrow$, $\wedge$ and $\vee$ denote their respective logic relations as usual, namely $\neg$ for 'not', $\Rightarrow$ for 'imply', $\wedge$ for 'and', $\vee$ for 'or' and '$\rho_1 \Leftrightarrow \rho_2$' for '$(\rho_1 \Rightarrow \rho_2) \wedge (\rho_2 \Rightarrow \rho_1)$'.
  - $\forall X \in E.S$ and $\exists X \in E.S$ are statements, if X is a free variable in statement S.

6.2.2 Constraints for adaptive properties

Based on the properties of adaptive agents defined in Section 4.4, constraints on transition diagrams can be formally specified as statements of the first order language defined above. The abstract syntax of transition diagrams specified in Section 6.1 defines the notation for representing constructs of transition diagrams in the following constraints.

(*A*) *Constraints for rational agents*

Recall that Definition 14 defines rational agents as the agents that satisfy four properties. The four properties of agents are represented as the following four constraints on transition diagrams. A transition diagram describes a rational agent if each transition T in the diagram satisfies the constraints. We use *Begin* (T) to denote the configuration where T starts.

- *Join* (C) $\in$ *T.<Transition rule>.<Action>* $\Rightarrow$ $C \notin$ *Begin* (T). (*<Active Casteship>* $\cup$ *<Inactive Casteship>*)

  This constraint means that for transition T, if action *Join* (C) is among the action part of its transition rule, its start configuration must not contain C as active or inactive casteship.
- *Quit* (C) $\in$ *T.<Transition rule>.<Action>* $\Rightarrow$ $C \in$ *Begin* (T). *<Active Casteship>*

  This constraint means that for transition T, if action *Quit* (C) is among the action part of its transition rule, its start configuration must contain C as active casteship.

- *Deactivate* (C) ∈ *T*.<*Transition rule*>.<*Action*> ⇒ *C* ∈ *Begin* (*T*) . <*Active Casteship*>

  This constraint means that for transition *T*, if action Deactivate (*C*) is among the action part of its transition rule, its start configuration must contain *C* as active casteship.
- Activate (*C*) ∈ *T*.<*Transition rule*>.<*Action*> ⇒ *C* ∈ *Begin* (*T*) . <*Inactive Casteship*>

  This constraint means that for transition *T*, if action Activate (*C*) is among the action part of its transition rule, its start configuration must contain *C* as inactive casteship.

(*B*) *Constraints for faithful agents*

Let CASTES denote the set of castes in the system to be developed. Based on the definition of *DReach*() for individual caste, Definition 10, we define a 2-tuple relation ***DirectReach*** = {<c_1, c_2> | $c_1 \Rightarrow c_2$, c_1, c_2 ∈ *Castes*}. ***DirectReach*** is an irreflexive, non-symmetric, non-transitive relation that represents the set of directly reachable pairs of castes in the system. From the global perspective to the system, using ***DirectReach*** rather than *DReach*() simplifies the definition of constraints on transition diagrams. When developing adaptive systems, modelers should explicitly define ***DirectReach*** for the system at early stage of software development based on the analysis of targeted systems. Such information that restricts certain types of relations between castes can be used to regulate the design and to detect errors in system models. Recall the definition of faithful agents, Definition 15. A transition diagram *D* for an agent *a* must satisfy the following constraint in order for *a* to be a faithful agent.

- ∀*T* ∈ *D*.<*Transition*>. (*Join*(*c*) ∈ *T*.<*Transition rule*>.<*Action*> ⇒ ∃*s* ∈ *Begin* (*T*).<*Active Casteships*>. (<*s*, *c*> ∈ ***DirectReach***))

This constraint means that for any transition *T*, if action *Join* (*c*) is among the action part of its transition rule, there must be an active casteship *s* in the start configuration so that <*s*, *c*> bears ***DirectReach*** relation.

(C) Constraints for consistent agents

Based on the definition of *conflict* for individual caste, Definition 12, we define a 2-tuple relation ***Conflict*** = {<c_1, c_2> | $c_1 \uparrow c_2$, c_1, c_2 ∈ *Castes*}. ***Conflict*** is irreflexive, symmetric and non-transitive relation that denotes the set of conflict pairs of castes in the system. Modelers should also explicitly define ***Conflict*** for the system at the early stage of software development. Recall that Definition 16 defines the agents that are consistent about caste operations. A transition diagram *D* for an agent *a* must satisfy the following constraints in order for *a* to be a consistent agent about caste operations. We use *End*(*T*) to denote the configuration where a transition *T* is directed to.

- ∀*T* ∈ *D*.<*Transition*>.(*Join*(*c*) ∈ *T*.<*Transition rule*>.<*Action*> ⇒ ∀*s* ∈ *End* (*T*).<*Active Casteships*>. (<*s*, *c*> ∉ ***Conflict***)).

This constraint means that for transition *T*, if action *Join* (*c*) is among the action part of its transition rule, there must be no such active casteship *s* in the end configuration that <*s*, *c*> bears ***Conflict*** relation.

- ∀*T* ∈ *D*.<*Transition*>.(*Activate*(*c*) ∈ *T*.<*Transition rule*>.<*Action*> ⇒ ∀*s* ∈ *End* (*T*).<*Active Casteships*>. (<*s*, *c*> ∉ ***Conflict***)).

This constraint means that for transition *T*, if action *Activate* (*c*) is among the action part of its transition rule, there must be no such active casteship *s* in the end configuration that <*s*, *c*> bears ***Conflict*** relation.

Besides the consistency with respect to actions, each configuration of a consistent agent must also be consistent by itself, i.e. do not contain conflict pair of castes. Formally, for a transition diagram *D*,

- ∀*S* ∈ *D*.<*Configuration*>.∀*a*, *c* ∈ *S*.<*Active Casteships*>. (<*a*, *c*> ∉ ***Conflict***).

(*D*) *Constraints for reachable agents*

Recall that Definition 17 defines reachable agents. In the context of graphic modelling, the set of agents of a caste is reachable if all directly reachable relations from the caste are described in the transition diagram for the caste. Given the ***DirectReach*** set for a system, a transition diagrams *D* defines a reachable caste when it satisfies the following constraint.

- ∀<*s*, *c*> ∈ ***DirectReach***. (*s* ∈ *D*.<*StartConfiguration*>. <*Active Casteship*> ⇒ ∃*T* ∈ *D*.<*Transition*>. (*Begin*(*T*) = *D*.<*StartConfiguration*> ∧ *c* ∈ *End*(*T*). <*Active Casteships*>))

where *D*.<*StartConfiguration*> denotes the non-trivial (not initial situation or final situation) start configuration in diagram *D*. This constraint means that if *D* is for agents of caste *s*, all possible direct casteship transition from *s* must be described in *D*.

The constraints defined above can be used individually or in combination to check whether the models of adaptive MAS are correct with respect to certain desirable properties. The constraints are apt to be implemented as automatic tool so that the checking of models can be automated. In our previous study on agent-oriented modelling with the CAMLE language, we have developed a consistency checking tool based on the type system, syntax and consistency constraints of CAMLE (Shan and Zhu, 2004). The effectiveness of our consistency checking method has been verified (Shan and Zhu, 2006). The implementation of automated tool for checking transition diagrams is in our research agenda.

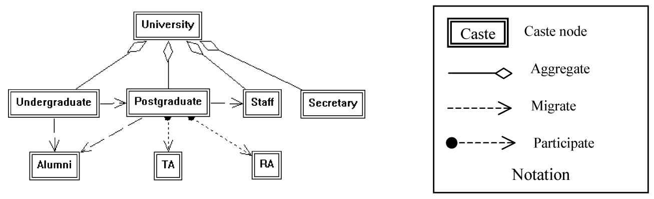
7 Case study: university information system

This section illustrates the use of transition diagram for modelling adaptive MAS with the example of the university information system specified in Section 2. Using the constraints of transition diagram defined in Section 6.2 for checking CAMLE model and transition diagrams is also demonstrated.

7.1 Caste diagram

To capture the structure of the university information system, the castes that constitute the system and the structural relations between the castes, a caste diagram in CAMLE is constructed as shown in Figure 5. The notation shown in Figure 5 is the part of the notation of CAMLE caste diagram that is involved in this example. It is worth noting that both *migrate* and *participate* represent casteship transition relations. The difference between them is that migrate relation between caste *A* and caste *B* denotes that an agent of *A* quits from *A* before it joins *B*, while participate relation between *A* and *B* denotes that an agent of caste *A* joins caste *B* while remaining its casteship to *A*.

Figure 5 Caste diagram for University and the notation



Castes in the University information system is represented by the set

CASTES = {Undergraduate, Postgraduate, Staff, Secretary, Alumni, TA, RA}

Note that only Undergraduate, Postgraduate, Staff and Secretary represent the members of a university. The caste Alumni is considered in the system because the university needs to keep a record of the alumni. TA and RA are positions that the university provides for postgraduates. Since the example aims to demonstrate the effectiveness of our adaptive casteship mechanism rather than give a complete model of the university system, we do not consider more refined classification for Staff or Secretary for the sake of space.

In the system, agents of Undergraduate and Postgraduate may be adaptive agents, while agents of Staff, Secretary and Alumni are static agents because they cannot change their casteships. To properly model adaptive agents, we must define the ***DirectReach*** and ***Conflict*** set for the system. Without loss of generality, suppose the university neither allows a student (undergraduate or postgraduate) to take part-time job as staff or secretary, nor a staff or secretary to be part-time student. A postgraduate may be TA or RA or both or neither of them during his study. A postgraduate, staff or secretary may belong to the alumni if he did graduate from the university. Therefore, we have that

DirectReach = {<Undergraduate, Postgraduate>, <Undergraduate, Alumni>, <Postgraduate, Alumni>, <Postgraduate, TA>, <Postgraduate, RA>, <Postgraduate, Staff>},

Conflict = {(Undergraduate, Alumni), (Undergraduate, Postgraduate), (Undergraduate, Staff), (Undergraduate, Secretary), (Undergraduate, TA), (Undergraduate, RA), (Postgraduate, Staff), (Postgraduate, Secretary), (Staff, Secretary), (Staff, TA), (Staff, RA), (Secretary, TA), (Secretary, RA)}.

To help define ***Conflict***, we can also define ***Consistent*** as the set of the 2-tuple castes without conflict relation. In other words, ***Consistent*** = $\{(x, y) \mid x, y \in \mathit{Castes}, x \neq y\}$ – ***Conflict***.

Consistent = {(Postgraduate, Alumni), (Postgraduate, TA), (Postgraduate, RA), (Staff, Alumni), (Secretary, Alumni), (Alumni, TA), (Alumni, RA), (TA, RA)}.

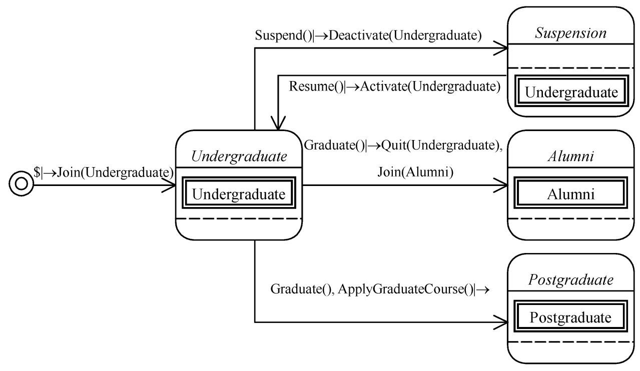
The caste diagram that describes the whole set of castes in the system and the migration relations between the castes helps to define the sets. Actually, ***DirectReach*** equals to the migration relations in the caste diagram. Note that ***Conflict*** relation is not directly related to the relations described in caste diagram.

7.2 Transition diagram

The definition of ***Conflict*** resulted from the analysis of the requirement and constraints of the system. ***Conflict*** can be used to detect errors in transition diagrams. For example, it is incorrect to design an agent that joins Staff and Postgraduate at the same time.

For agents of the Undergraduate caste, we develop a transition diagram as shown in Figure 6 to model the casteship transition within its lifecycle. The transition diagram shown in Figure 7 is for the agents of Postgraduate caste. Note that in each transition, the *event* part only describes the action or state of the agent that precedes the caste transition actions. It is not the complete condition that invokes the casteship transition, because the result action also depends on the scenario of the whole system. Take the transition from the configuration of *Undergraduate* to *Suspend* as an example. When an undergraduate applies to the university to suspend his study, his application may either get permitted or refused. Similarly, when an undergraduate applies graduate course from the university, he may either get an offer or be rejected. However, it should be the behaviour diagram in CAMLE rather than the transition diagram that specifies the complete set of behaviour rules of Undergraduate in various scenarios of the system.

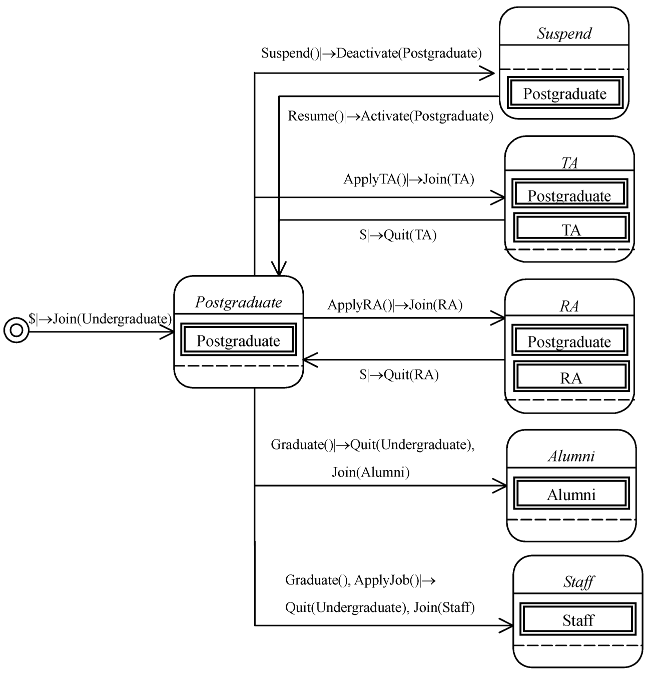
Figure 6 Transition diagram for undergraduate agents


Suspend()|→Deactivate(Undergraduate)
Suspension
Resume()|→Activate(Undergraduate)
Undergraduate
Undergraduate
Graduate()|→Quit(Undergraduate), Join(Alumni)
Alumni
$|→Join(Undergraduate)
Undergraduate
Alumni
Graduate(), ApplyGraduateCourse()|→
Postgraduate
Postgraduate

The two transition diagrams describe all possible configurations and casteship changing of agents of the two castes. With the constraints defined in Section 6.2, it is easy to check the two transition diagrams of the properties and to conclude that the agents of Undergraduate and Postgraduate are rational, faithful, consistent and reachable, i.e., safe.

The development process described above takes the sets of ***DirectReach*** and ***Conflict*** as part of requirement specification of the targeted system, then models the system based on the specification, and checks the properties of the model against the constraints defined on transition diagrams. The process of developing adaptive MAS may also start from graphic modelling with caste diagrams and transition diagrams based on the analysis of the targeted system. Then the two sets ***DirectReach*** and ***Conflict*** can be defined according to the models to rigorously document certain design decisions. Anyway, the constraints on the transition diagrams help to check the consistency between the two types of artifacts and to analyse the properties of the system under development.

Figure 7 Transition diagram for postgraduate agents



8 Comparison with related work

In object-oriented paradigm, when an object is created by instantiating some class, it obtains the structural and behavioural features defined by the class. However, an object can not change its structure and behaviour in its lifecycle. The lack of support to dynamic re-classification of objects has been recognised to be a limitation of object oriented programming (Scheer and Pringle, 1998). In order to solve the problem, efforts have been made to extend object-oriented programming languages so that the class membership of an object can be changed at run-time, e.g., the language Fickle$_{\text{II}}$ (Drossopoulou et al., 2002).

In the literature of agent-oriented software engineering, many agent-oriented methodologies like MaSE, Tropos, etc., assume that agents in MAS are static in order to simplify the development processes and modelling languages (Mao and Yu, 2004). Some methodologies like ROADMAP (Juan et al., 2002), the recent version of Gaia (Zambonelli et al., 2003), etc., though claiming to support the development of open and dynamic systems, actually do not consider dynamic agents and therefore do not provide effective mechanisms and language facilities to support the modelling and designing of adaptive agents.

However, there are still a number of agent-oriented software development methodologies, modelling languages and programming languages, such as AALADIN (Ferber and Gutknecht, 1998), AUML (Odell et al., 2003), and 3APL (Dastani et al., 2005) etc., that have been proposed to support modelling role changes to some extents.

In Zhu (2003a) and Zhu and Lightfoot (2003), a dynamic caste mechanism is proposed as a language facility for the design and implementation of dynamic adaptation of behaviours in MAS. Its direct support for role-based models is discussed in Zhu and Lightfoot (2003). In this paper, we further analyse the requirements for a language facility that supports adaptive agents and extend the caste facility to meet these requirements.

In AALAADIN (Ferber and Gutknecht, 1998), the dynamic aspect of MAS is related to the institutionalised patterns of interactions that are defined within the roles, such as creation of groups, entering and leaving of a group by an agent, or acquisition of a role in relation, which can be specified by organisational sequence diagram, an extension of sequence diagram in UML. Therefore, the dynamic aspect in AALAADIN is considered in the structure level, i.e., group, which is different from our work that considers the dynamic aspect in the behaviour level.

AUML is a set of UML idioms and extensions for specifying multi-agent software systems (Bauer et al., 2001). Based on the insight that agent is an extension to active object, the core part of AUML is protocol diagram which describes how agent changes its roles in different contexts (Odell et al., 2003). However, role is an abstract concept closer to real world and can not act as design metaphor or modular unit to implement agents. In most of the existing agent-oriented methodologies and modelling languages, an concept of agent class or agent type is introduced in the design and implementation phase as a template or modular unit to construct agents, but few formally define the concept with any adaptive casteship mechanism. The relationship between agent class or agent type and agent is the same as that in object orientation, i.e., agent class is the template to instantiate agent and agent is the instance of agent class.

In 3APL (Dastani et al., 2005), four operations are presented such as 'enact', 'deact', 'activate' and 'deactivate', to capture the dynamics of roles played by agents. Enacting a role means internalising the specification of the role, while activating a role means reasoning with the (internalised) specification of the role. The semantics of the operations 'enact', 'deact', 'activate' and 'deactivate' in 3APL are actually different from that of the operations 'join', 'quit', 'activate' and 'deactivate' defined in this paper. Moreover, 3APL assumes that at any moment only one role can be active and that all roles other than the enacted one are inactive. The cognitive architecture of agents is assumed in 3APL, on which agents' enacting or deacting the roles are investigated. Obviously, these assumptions are too strong to tackle the whole aspects related to dynamic agents in complex systems. There is a huge gap between the role specification and agent cognitive architecture for developers to transfer the role model to cognitive model. Our approach to adaptive agents is different from 3APL as we permit multiple castes to be bound to at a moment and the *join* operation actually integrates with the *enact* and *activate* operations in 3APL. There is no explicit gap between requirement specification and software design in our approach as we use the unified concepts and abstractions in the whole development process. Therefore, our approach proposed in this paper simplifies the development of complex adaptive agents.

9 Conclusion and future work

Agent orientation provides high level abstractions and a natural modelling metaphor to develop software systems. However, in the literature of agent-oriented software engineering, it is still a challenge to develop complex MAS with agents that exhibit various and dynamic behaviours in their lifecycle. Such agents are desirable in many complex applications of MAS, such as internet-based applications, enterprise information systems, etc.

In this paper, we present the adaptive casteship mechanism, which is the extension to Mao et al. (2006), to model and design individual adaptive agents in MAS. It enables the supports for the execution of adaptive agents at run-time. We adopt caste as the abstraction to specify agents' behaviours and as the modular unit to design and implement adaptive agents. Our approach permits agents to hold casteship to multiple castes and the casteships that agent holds to be in active or inactive states. The dynamic adaptation of behaviours is realised as the change of agents' casteships in their lifecycles, which can be specified and implemented by four atomic operations on castes i.e., *join*, *quit*, *activate* and *deactivate*. The semantics of the adaptive casteship mechanism and the caste operations are rigorously defined based on the temporal logic integrated with dynamic operators. The properties of adaptive agent's behaviours are formally specified and proved. In order to support the modelling and analysing agents' dynamic adaptation of the behaviours with the adaptive casteship mechanism at design-time, a visual notation of caste transition diagrams is proposed and a number of model consistency rules are designed.

The adaptive casteship mechanism proposed in this paper provides an abstract and flexible way to develop complex MAS, especially ones that have adaptive agents. However, there are still a number of open problems that should be settled.

- Firstly, the work reported in this paper is actually at the micro-level, i.e., it is concerned with individual agents and their dynamic properties and behaviours. However, the adaptability can be manifested at the level of MAS. As discussed in Mao and Yu (2004), the structure of MAS in terms of the relationships between agents may also change dynamically in some complex systems when environment changes. The adaptability issueat the macro-level, i.e., the adaptability of MAS, is not dealt with in this paper and deserves further investigation. For example, for the sake of simplicity, we have not taken the inheritance relationships between castes into consideration in this paper. It is also assumed in this paper that caste operations are not executed concurrently.
- Secondly, the work reported in this paper set in the context of analysis and design of MAS. We believe that the adaptive casteship mechanism can be equally used in the design and implementation of agent-oriented programming languages that support the adaptive agents. Implementation of the adaptive casteship mechanism on runtime platforms is worth further investigation.
- Moreover, we are carrying out case studies of the adaptive casteship mechanism with a number of complex applications. For example, we intend to implement resource sharing utility by applying the adaptive casteship mechanism. In mobile computing area, a mobile agent can access the destination computer by executing the behaviour specifications provided by the destination computer. The adaptive casteship mechanism can also be an effective technical solution to developing complex application such as self-organisation and self-management application.
- Finally, the relation between caste transition diagrams and other diagrams in CAMLE is still an open problem. How to ensure the consistency between caste transition diagrams and CAMLE models, and whether it is possible to derive caste transition diagrams from caste diagram and behaviour diagrams are also problems to be investigated in our future work.

Acknowledgements

The authors gratefully acknowledge financial support from Natural Science Foundation of China under granted number 60773018 and 90612009, 863 Programme of China under granted number 2007AA01Z135, and 973 Programme of China under granted number 2005CB321802.

References

Bauer, B., Müller, J.P. and Odell, J. (2001) 'Agent UML: a formalism for specifying multiagent software systems', *International Journal of Software Engineering and Knowledge Engineering*, Vol. 11, No. 3, pp.207–230.

Dastani, M., Riemsdijk, M.B.V., Hulstijn, J., Dignum, F. and Meyer, J-J.C. (2005) 'Enacting and deacting roles in agent programming', *Proceedings of the 5th International Workshop on Agent-Oriented Software Engineering*, LNCS 3382, Springer-Verlag, Berlin, Germany, pp.189–204.

Drossopoulou, S., Damiani, F., Dezani-Ciancaglini, M. and Giannini, P. (2002) 'More dynamic object reclassification: FickleII', *ACM Transactions on Programming Languages and Systems*, Vol. 24, No. 2, pp.153–191.

Ferber, J. and Gutknecht, O. (1998) 'A meta-model for the analysis and design of organizations in multi-agent systems', *Proceedings of the 3rd International Conference on MultiAgent Systems*, IEEE Computer Society, Washington DC, USA, pp.128–135.

Giunchiglia, F., Mylopoulos, J. and Perini, A. (2002) 'The Tropos development methodology: processes, models and diagrams', *Proceedings of International Conference on Autonomous Agent and Multi-Agent System*, ACM Press, Washington DC, USA, pp.35, 36.

Jennings, N.R. (2001) 'An agent-based approach for building complex software systems', *Communication of ACM*, Vol. 44, No. 4, pp.35–41.

Juan, T., Pearce, A. and Sterling, L. (2002) 'ROADMAP: extending the Gaia methodology for complex open system', *Proceedings of the 1st International Joint Conference on Autonomous Agents and Multiagent Systems*, ACM Press, New York, USA, pp.3–10.

Luck, M., Mcburney, P. and Preist, C. (2005) *Agent Technology: Enabling Next Generation Computing, A Roadmap for Agent-Based Computing*, Available from http://www.agentlink.org

Mao, X. and Yu, E. (2004) 'Organizational and social concepts in agent oriented software engineering', *Proceedings of the 5th International Workshop on Agent-Oriented Software Engineering*, LNCS 3382, Springer-Verlag, Berlin, Germany, pp.1–15.

Mao, X., Chang, Z., Shan, L., Zhu, H. and Wang, J. (2006) 'The dynamic casteship mechanism for modeling and designing adaptive agents', *Proceedings of the 2nd International Workshop on Agent-oriented Software Development Methodologies (AOSDM'06) at SEKE'06*, Knowledge Systems Institute, Illinois, USA, pp.639–644.

Odell, J., Parunak, H.V.D. and Fleischer, M. (2002) 'The role of roles', *Journal of Object Technology*, Vol. 2, No. 1, pp.39–51.

Odell, J., Parunak, H.V.D., Brueckner, S. and Sauter, J. (2003) 'Temporal aspects of dynamic role assignment', *Proceedings of the 4th International Workshop on Agent-Oriented Software Engineering*, LNCS 2935, Springer-Verlag, pp.201–213.

Scheer, T. and Pringle, S. (1998) *Ten Practical Limitations of Object Orientation, OOPSLA Poster Session*, Available from http://www.ac.org/sigplan/oopsla//oopsla98/fp/posters/10.htm

Shan, L. and Zhu, H. (2004) 'Consistency check in modeling multi-agent systems', *Proceedings of COMPSAC*, IEEE Computer Society, pp.114–121.

Shan, L. and Zhu, H. (2005) 'CAMLE: a caste-centric agent-oriented modelling language and environment (Book Chapter)', in Choren, R., Garcia, A., Lucena, C. and Romanovsky, A. (Eds.): *Software Engineering for Multi-Agent Systems III.* Springer-Verlag, Berlin, Germany, pp.144–161.

Shan, L. and Zhu, H. (2006) 'Specifying consistency constraints for modelling languages', *Proceedings of the 18th International Conference on Software Engineering and Knowledge Engineering (SEKE'06)*, Knowledge Systems Institute, Illinois, USA, pp.578–583.

Shan, L., Shen, R., Wang, J. and Zhu, H. (2006) 'Caste-centric development of agent oriented information systems', in Rennard, J-P. (Ed.): *Handbook of Research on Nature Inspired Computing for Economy and Management*, Idea Group Inc., Hershey, USA, pp.692–707.

Wang, J., Shen, R. and Zhu, H. (2005) 'Agent oriented programming based on SLABS', *Proceedings of COMPSAC*, IEEE Computer Society, Edinburgh, Scotland, pp.127–132.

Zambonelli, F. and Omicini, A. (2004) 'Challenges and research directions in agent-oriented software engineering', *Autonomous Agent and Multi-Agent Systems*, Vol. 9, No. 3, pp.253–283.

Zambonelli, F., Jennings, N.R. and Wooldridge, M. (2003) 'Developing multiagent systems: the Gaia methodology', *ACM Transactions on Software Engineering Methodology*, Vol. 12, No. 3, pp.317–370.

Zhu, H. and Lightfoot, D. (2003) 'Caste: a step beyond object orientation', *Proceedings of Joint Modular Languages Conference*, LNCS 2789, Springer-Verlag, Berlin, Germany, pp.59–62.

Zhu, H. and Shan, L. (2005) 'Caste-centric modelling of multi-agent systems: the CAMLE modelling language and automated tools (Book Chapter)', *Model-driven Software Development, Research and Practice in Software Engineering*, Springer-Verlag, Berlin, Germany, pp.57–89.

Zhu, H. and Shan, L. (2006) 'Well-formedness, consistency and completeness of graphic models', *Proceedings of the 9th International Conference on Computer Modelling and Simulation (UKSim)*, United Kingdom Society for Modelling and Simulation, Cambridge, UK, pp.47–54.

Zhu, H. (2001) 'SLABS: a formal specification language for agent-based systems', *International Journal of Software Engineering and Knowledge Engineering*, Vol. 11, No. 5, pp.529–558.

Zhu, H. (2003a) 'A formal specification language for agent-oriented software engineering', *Proceedings of International Conference on Autonomous Agent and Multi-Agent System*, ACM Press, New York, USA, pp.1174–1175.

Zhu, H. (2003b) *Representation of Role Models in Castes*, Technical Report DoC-TR-03-02, Department of Computing, Oxford Brookes University, Oxford.

Zhu, H. (2006) 'Towards an agent-oriented paradigm of information systems (Book Chapter)', in Rennard, J-P. (Ed.): *Handbook of Research on Nature Inspired Computing for Economy and Management.* Idea Group Inc., Hershey, USA, pp.679–691.