The Dynamic Casteship Mechanism for Modeling and Designing Adaptive Agents

Xinjun Mao[1], Zhiming Chang[1], Lijun Shang[1], Hong Zhu[2] and Ji Wang[1]
[1]*Dept. of Computer Science, National University of Defense Technology, Changsha, China, 410073*
Email: xjmao@nudt.edu.cn, wj@nudt.edu.cn
[2]*Dept. of Computing, Oxford Brookes University Oxford OX33 1HX, UK*
Email: hzhu@brookes.ac.uk

Abstract. It is still a challenge to develop complex multi-agent systems with agents that typically exhibit adaptation to various behaviors in different situations. In this paper, we propose an extension of the dynamic casteship mechanism for modeling and designing adaptive agents. In our approach, caste is the modular unit in the design and implementation of multi–agent systems and provides an abstraction of the ways that agents adapt to different behaviors in different situations. The dynamic behaviors of agents are realized as the change of castes that agents bind during execution by "*join*" and "*quit*" operations on agent's casteships. The extended mechanism also enables agents to change the status of its castes to be either *active* or *inactive* at run-time. Accordingly, two new operations '*deactivate*' and '*activate*' on agent's casteships are introduced. In the paper, the formal semantics of these operations is rigorously defined. The properties of dynamic binding mechanism are investigated.
Keywords: adaptive agent, caste, dynamic binding

1. Introduction

Agent orientation has recently emerged as a new software development paradigm for developing complex software systems that operates in dynamic environment such as the Internet [6]. 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 development methodologies of complex multi-agent systems (MAS). Agent-oriented software engineering (AOSE) has become an active research area in agent-based computing. A number of methodologies, programming languages and CASE tools or environments have been proposed, such as Gaia[5], Tropos[11], CAMLE[3], etc.

Nevertheless, 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 [9] that are typically unpredictable, open, dynamic, hierarchically structured but confined by global constraints [1]. We believe that, to be a successful general-purpose software engineering paradigm, language facilities 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 [4]. Moreover, the methods must be universally applicable. This paper investigate the language facility caste proposed in [3] and [12] for the design and implementation of dynamic behavior adaptation in MAS. In the sequel, we will use *adaptive agents* to denote agents that are capable of adapting to different behaviors at runtime.

Agents executing in a dynamic environment often have to behave differently in different situations. For example, frequent role changes are common for agents in human societies and organizations. However, existing agent-oriented methodologies have not provided enough supports to develop such systems that naturally take such advantages, especially in the modeling, specification, design and implementation of such dynamic behaviors in a systematic way. It is no surprise that it is extremely difficult to develop MAS with adaptive behaviors [12]. This paper addresses this problem by the dynamic binding mechanism based on the language facility caste proposed in [4,12].

The remainder of the paper is organized as follows. Section 2 introduces an example of adaptive agents. In section 3, caste is defined and dynamic binding mechanism is informally discussed. Section 4 rigorously defines the formal model of MAS and the formal semantics of four atomic operations on castes. The properties of dynamic binding mechanism are also investigated in this section. Finally section 5 concludes the paper.

2. An example of adaptive dynamic MAS

In order to understand the requirements on the dynamic binding mechanism, in this section, we examine an example of information system to illustrate the basic characteristics of complex systems that consist of adaptive agents.

Suppose that an information system is to be developed for a university to support the management of the university's students, staff and related affairs such as registration, course selection, etc. and to 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, a member can take

certain actions in the operation of the information systems, access certain subset of the information stored in the systems, and must obey certain set of rules that confines the member to fulfill his tasks. Each member of the university is then supplied with a 'personal assistant', which is a software agent that stores the personal information about the member, the permitted actions that the member can take according to his role, as well as the personal preferences and private information. The personal assistant agent must also support the collaborations between the members.

As in almost all organizations, a member of the university may change its roles while remaining as a member of the organization. For example, an undergraduate student, say Alex, who is going to graduate from the university. After graduation, he gets an offer from the university to study for a Master degree, which will take him two years to complete. The graduation does not mean that he will change his identity. It just manifests that he will play a new role in the organization of the university. By changing role, he 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 organization such as his personal tutor, etc. This will also give him some new capability of actions, such as access to the master degree student 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, and the set of permitted actions, etc. What is 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.

Role changes of occur not only as moving from one role to another, but also as in the form of temporarily leaving from a post and then returning to the role to continue the job. For example, after one year's study, Alex decides to take one year 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 for ever. Instead, during the work placement year, he suspends his behavior 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. The information system should support Alex to suspend and resume his study by keeping all personal information such as the study program, scores of passed or failed modules, credits earned, etc. suspending the permitted activities as well as some restrictions on his performances, etc. while he is on leave, and resuming his normal postgraduate student behavior after returning to the university. The dynamic binding mechanism that supports the changes between roles must also be able to support such temporary suspension of a role and resumptions of a role after suspensions.

As in almost all organizations, a member of the university may also play multiple roles at the same time. For example, when Alex becomes a postgraduate, he also takes a teaching assistant (TA) job, 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 on two different roles. The dynamic binding mechanism should be able to support the taking of new roles while stay in a role.

3. Caste and Dynamic Binding Mechanism

The notion of caste was originally introduced in SLABS. It helps to deal with the limitation of object technology [4]. We regard caste as basic abstraction to specify agents' behaviors and the modular unit to design and implement multi-agent systems.

In our meta-model of MAS, agent is defined as an autonomous and persistent computational entity situated in certain environment. Each agent consists of four parts: data, actions, behaviors and environments. That is, the structure of an agent is a 4-tuple, i.e., *Agent* = <*D*, *A*, *B*, *E*>, were *D* represents an agent's state space, *A* is the actions that agent can take, *B* is the behavior rules that determine how agent behaves in the context of its designated environment *E*. It is worth noting that these parts may change during the execution of the agent. Such changes are realized by the dynamic binding of an agent to castes.

A caste defines a set of structural features and behavior patterns for agents. A caste 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 behavior 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, which consists of a set of agents that have influence on the agent's behavior [12]. When an agent binds to a caste, it obtains the state space and the capability to take an action defined by the caste, becomes in collaboration with and influenced by the agents specified in the caste definition and must obey the behavior rules specified in the caste. For example, in the above example, we can define two castes to represent the roles of undergraduate and postgraduate students, respectively.

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 that binds to the caste 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, which are also agents. Similarly, the value of a visible data can be observed by other agents, while the value of an internal

state can only be observed by its components. For the sake of simplicity, in the sequel, we assume all the actions and state variables are visible because, in this paper, we are not concerned with the visibility issues.

Although there are similarities with the relationship between object and class and the relationship between data and type, none of the terms like *instance*, *member*, *classifier* and *type* accurately represents the relationship between agent and caste. The main difference is that an agent can take actions to join a caste or retreat from a caste at run-time. Consequently, it obtains (or lose) the structural and behavioral features defined by the caste. Hence, the relationship is called casteship.

It is worth noting that an agent's action of joining or quitting a caste is also determined by the behavior rules defined by the caste that the agent currently binds to. Therefore, the behavior rules specified in each caste explicitly declare what castes the agent can *join* and *quit* and in what situation to do so. In the above example, Alex graduates and becomes a postgraduate student must be the result of executing a behavior rule that enables an undergraduate to quit from the caste *Undergraduate* and to join the caste *Postgraduate*.

However, the existing mechanism of dynamic binding of agents to castes is still insufficient in the modeling of adaptive agents as discussed in section 2. For example, it can not distinguish the castes in active status from ones in inactive status. To overcome the drawbacks, this paper extend the dynamic binding mechanism by introducing the notion of active binding and inactive binding of an agent to a caste.

(1) Active binding

An agent's binding to a caste is active means that the agent obtains all the structural and behavioral features defined by the caste. In other words, the agent can take actions defined in the action part of the caste according to the behavior 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.

(2) Inactive binding

An agent's binding to a caste can also be inactive, which means that the agent can not change the state variables and cannot take actions defined in an inactive caste while it still maintains its values of the state variables defined by the caste. The behavior rules of the inactive caste will not affect the agent's behavior. 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 binding to the caste again, the values of the state variables are resumed to that when the agent's binding to the caste last changed to inactive. This is different from that agent retreats a caste.

An agent can change its casteship status by taking action *deactivate* to become inactive and *activate* to become active. When agent takes action to *join* a caste, the casteship will be in active state.

4. Formal model of dynamic casteship mechanism

In this section, we formally define the extended dynamic binding mechanism. We will first define the model of multi-agent systems.

4.1. Model of Multi-Agent Systems

In [12], the caste-centric formal model of MAS has been formally defined. The following extends the formal model to enable active and inactive dynamic binding of agents to be specified. For the sake of simplicity, we have omitted some aspects, such as the inheritance relationship between castes and scenario definition, of the original model so that we can focus on the dynamic casteship mechanism.

Definition1. A model M of MAS consists of two parts < MAS, CASTE>, where MAS = $\{a_1, a_2, ..., a_n\}$ is a finite set of agents and CASTE = $\{c_1, c_2, ..., c_m\}$ is a finite set of castes.

Agents in MAS 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 dynamic binding 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 assumes 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.

Agent's state space at some moment depends on the casts that agent binds to at that time moment and their status. Let $D^A_{a,t} = \cup_{\{c \in CASTE^A(a,t)\}} Dc$ denote the state spaces of agent *a* at moment *t* based on active castes that it binds to at moment *t*, $D^I_{a,t} = \cup_{\{c \in CASTE^I(a,t)\}} Dc$ 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 $A^A_{a,t} = \cup_{\{c \in CASTE^A(a,t)\}} A_c$ the action set of agent *a* at moment *t* based on active castes that it binds to at moment *t*, $A^I_{a,t} = \cup_{\{c \in CASTE^I(a,t)\}} A_c$ 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 $B^A_{a,t} = \cup_{\{c \in CASTE^A(a,t)\}} B_c$ the behavior rule set of agent *a* at moment *t* based on active castes that it binds to at moment *t*, $B^I_{a,t} = \cup_{\{c \in CASTE^I(a,t)\}} B_c$ the behavior rule set of agent *a* at moment *t* based on inactive castes that it binds to at moment *t*; $E^A_{a,t} = \cup_{\{c \in CASTE^A(a,t)\}} E_c$ the environment of agent *a* at moment *t* based on active castes that it binds to at moment *t*, $E^I_{a,t} = \cup_{\{c \in CASTE^I(a,t)\}} E_c$ the environment of agent *a* at moment *t* based on inactive

castes that it binds to at moment *t*.

Definition2. The state 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 $s^A_{a,t} = D^A_{a,t} \times A^A_{a,t} \times B^A_{a,t} \times E^A_{a,t}$ denoting the state of agent *a* at moment *t* based on active castes that it binds to, and $s^I_{a,t} = D^I_{a,t} \times A^I_{a,t} \times B^I_{a,t} \times E^I_{a,t}$ denoting the state 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 state of MAS at moment *t* is $S_{MAS,t} = \prod_{(a \in MAS)} s_{a,t}$. We write $S_{MAS} = \cup_{(t \in T)} S_{MAS,t}$ the set of all possible configurations of MAS.

Definition3. A run *r* of a MAS is a mapping from time *T* to the set S_{MAS}. The behavior 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 behavior 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.

Definition4. For any c_1, $c_2 \in$ *CASTE*, if the behavior rules of c_1 permit an agent that binds to caste c_1 to *join* caste c_2, then we call c_1 can directly reach c_2, denoted as $c_1 \Rightarrow c_2$.

We assume that the directly reachable relationship between castes is irreflexive, which means any caste is not permitted to be bound again when it has already bound. We write $DReach(c) = \{c' \mid c \Rightarrow c'\}$ to denote the directly reachable castes set of *c*, and $DReach(a,t) = \cup_{\{c \in \text{CASTE}^A(a,t)\}} DReach(c)$ to denote the directly reachable caste set of agent *a* at moment *t*. For example, if the behavior rule of caste *undergraduate* explicitly declares that when undergraduate student passes the entrance examination, he will join the caste of postgraduate, then *postgraduate* $\in$ *DReach(undergraduate).* The directly reachable relationship between castes does not satisfy transitive property. If c_1 can directly reach c_2, the agent that binds to caste c_1 is possible to *join* caste c_2 when the scenario and the pre-condition specified in the behavior rule are satisfied.

Definition5. 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 bind during its run. If c_1 can reach c_2, then the agent that binds to caste c_1 is possible to *join* caste c_2 in its run. Let $Reach(a,t) = \cup_{\{c \in \text{CASTE}^A(a,t)\}} Reach(c)$ the reachable caste set of agent *a* at moment *t*.

Definition6. Let c_1, $c_2 \in$ *CASTE*. If caste c_1 and c_2 are strictly not permitted for any agent *a* to bind to at the same time to govern the agent's behaviors 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 time moments in its run, we say that agent *a* is coherent.

For example, if the university does not permit any student to be an *undergraduate* and *postgraduate* simultaneously, then the castes *undergraduate* and *postgraduate* are exclusive to each other. Hence, the caste set {*undergraduate*, *postgraduate*} is not consistent. It is obvious that the exclusiveness 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, since {*undergraduate*, *postgraduate*} is inconsistent, the agent that binds to *undergraduate* must firstly quit the caste *undergraduate* before it joins the caste *postgraduate*.

Definition7. 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 $\text{CASTE}^A(a,t_1) \neq \text{CASTE}^A(a,t_2)$ or $\text{CASTE}^I(a,t_1) \neq \text{CASTE}^I(a,t_2)$. Otherwise, agent *a* is called *static*.

In the above example, agent Alex is a typical adaptive agent.

Lemma1. 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. [1]

4.2. Formalizing Dynamic Binding Mechanism

In this section, we formally define the dynamic binding mechanism and investigate its properties. 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*. "***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.

Definition8. Formally, 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 $M \models_{r,t+1} \varphi$

Definition9. $M \models_{r,t} <a: join(c)>$ iff $c \notin \text{CASTE}(a,t)$ and $\text{CASTE}^A(a,t+1) = \text{CASTE}^A(a,t) \cup \{c\}$ and $\text{CASTE}^I(a,t+1) = \text{CASTE}^I(a,t)$

Definition 9 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* actively

[1] For the sake of space, the proofs of the theorems and lemma in the paper are omitted.

binds to castes *c*, and the action execution will not change the inactive castes of agent *a*. A number of special predicates are introduced to specify the castes of agent and their status. "BindCaste(*a*, *c*)" means that agent *a* binds to caste *c*, "Active(*a*, *c*)" denotes that agent *a* binds to caste *c* and it is in active status. "Inactive(*a*, *c*)" means that agent *a* binds to caste *c* and it is in inactive status. Formally, their semantics are defined as follows.

Definition10. For all agents *a*, castes *c* and moments *t*,

- $M \models_{r,t} \text{BindCaste}(a, c)$ iff $c \in \text{CASTE}(a, t)$
- $M \models_{r,t} \text{Active}(a, c)$ iff $c \in \text{CASTE}^A(a, t)$
- $M \models_{r,t} \text{Inactive}(a, c)$ iff $c \in \text{CASTE}^I(a, t)$

Theorem1. *join* operation has the following properties.

(1) $\models \langle a: join(c) \rangle \rightarrow \bullet(\text{Active}(a, c))$

(2) $\models \langle a: join(c) \rangle \wedge \text{Inactive}(a, c_1) \rightarrow \bullet\text{Inactive}(a, c_1)$

Property (1) means that if agent *a joins* some caste *c* at moment *t*, then the agent will bind caste *c* actively in the next moment. Property (2) means that the *join* operation will not change the inactive castes of agent.

Definition11. $M \models_{r,t} \langle a: quit(c) \rangle$ iff $c \in \text{CASTE}^A(a, t)$ and $\text{CASTE}^A(a, t+1) = \text{CASTE}^A(a,t) \backslash \{c\}$ and $\text{CASTE}^I(a, t+1) = \text{CASTE}^I(a,t)$

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

Theorem2. *quit* operation has the following properties.

(1) $\models \langle a: quit(c) \rangle \rightarrow \bullet(\neg\text{BindCaste}(a, c))$

(2) $\models \langle a: quit(c) \rangle \wedge \text{Inactive}(a, c_1) \rightarrow \bullet\text{Inactive}(a, c_1)$

Property (1) 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.

Definition12. $M \models_{r,t} \langle a: deactivate(c) \rangle$ iff $c \in \text{CASTE}^A(a, t)$ and $\text{CASTE}^A(a,t+1) = \text{CASTE}^A(a,t) \backslash \{c\}$ and $\text{CASTE}^I(a,t+1) = \text{CASTE}^I(a,t) \cup \{c\}$

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

Theorem3. *deactivate* has the following properties

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

(2) $\models \langle a: deactivate(c) \rangle \rightarrow \bullet(\text{Inactive}(a, c))$

Property (1) means if agent *a deactivates* some caste *c*, then the caste *c* must be bound actively. Property (2) shows if agent *a deactivates* some caste *c*, then the state of caste *c* will be changed to inactive when the action is completed.

Definition13. $M \models_{r,t} \langle a: activate(c) \rangle$ iff $c \in \text{CASTE}^I(a, t)$ and $\text{CASTE}^I(a,t+1) = \text{CASTE}^I(a,t) \backslash \{c\}$ and $\text{CASTE}^A(a,t+1) = \text{CASTE}^A(a,t) \cup \{c\}$

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

Theorem4. *deactivate* has the following properties.

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

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

Property (1) means that if agent *a activates* caste *c*, then the caste *c* must be bound to inactively. Property (2) shows that if agent *a activates* caste *c*, then the state of caste *c* will be changed from inactive to active. We assume that there is no other actions in agents except *join*, *quit*, *deactivate* and *activate* that can change the casteships of an agent to any caste. Formally, for any agent *a*, caste *c*, action *act* and time moment *t*, if $M \models_{r,t} \langle a: act \rangle$ and $act \notin \{join, quit, deactivate, activate\}$, then $\text{CASTE}^I(a, t) = \text{CASTE}^I(a,t+1)$ and $\text{CASTE}^A(a, t) = \text{CASTE}^A(a,t+1)$).

Definition14. An agent *a* is *rational* about caste operations, if and only if, it satisfies the following properties.

(1) $M \models_{r,t} \langle a: join(c) \rangle \rightarrow M \models_{r,t} \neg\text{BindCaste}(a, c)$

(2) $M \models_{r,t} \langle a: quit(c) \rangle \rightarrow M \models_{r,t} \text{Active}(a, c)$

(3) $M \models_{r,t} \langle a: deactivate(c) \rangle \rightarrow M \models_{r,t} \text{Active}(a, c)$

(4) $M \models_{r,t} \langle a: activate(c) \rangle \rightarrow M \models_{r,t} \text{Inactive}(a, c)$

Formula (1) means that when an agent intends to *join* a caste, then the caste should not be bound by the agent. Formula (2) states that when an agent intends to *quit* a caste, the agent should have already actively bound to the caste. Formula (3) means that when an agent intends to *deactivate* a caste, the agent should have already actively bound to the caste. Formula (4) states that when an agent intends to *activate* a caste, then the caste should be already bound to inactively by the agent.

Definition15. An agent *a* is faithful, 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*.

Note that, an agent can only take actions defined by the castes that the agent actively binds, i.e., for any agent *a*, action *act* and moment *t*, $M \models_{r,t} \langle a: act \rangle \Rightarrow \exists c: c \in \text{CASTE}^A(a,t)$ and $act \in A_c$.

Lemma2. 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 $t' > t$: $c \notin CASTE(a, t')$.

The lemma states that if a caste is not reachable for agent *a* at some moment *t*, then the agent is impossible to bind the caste in its future run.

Definition16. 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*, then there is no caste $c' \in \text{CASTE}^A(a,t)$: $c \uparrow c'$; (2) if agent *a* intends to activate caste *c* at moment *t*, then there is no caste $c' \in \text{CASTE}^A(a,t)$: $c \uparrow c'$.

Lemma3. For any faithful agent *a* and moment *t*, if *Reach(a,t)* is consistent, then the agent will be consistent in its future run.

Lemma4. For any agent *a*, caste *c* and moment *t*, if agent *a* execute action "*join(c)*" and there is no caste $c' \in \text{CASTE}^A(a,t)$: $c \uparrow c'$, then agent *a* is consistent when the "*join*" action is completed; if agent *a* executes action "*activate(c)*" and there is no caste $c' \in \text{CASTE}^A(a,t)$: $c \uparrow c'$, then 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* has already been actively bound to, then when the operation is completed the agent is consistent.

Definition17. An agent follows the dynamic binding mechanism in its run, if and only if, the agent is rational, faithful and consistent about the caste operations. If all agents in MAS follow the dynamic binding mechanism, then we say that the MAS follows the dynamic binding mechanism, such MAS is abbreviated as MAS_{DBM}.

Definition18. We call that an agent *a* is reachable at moment *t*, if and only if, for any caste $c \in$ CASTE*(a, t)*, $\exists t' < t$: $M|=_{r,t'} <a: join(c)>$ and $c \in DReach(a,t')$.

Definition19. For any caste operation $act \in \{join, quit, activate, deactivate\}$ and an agent *a*, if agent *a* are always consistent and reachable based on the *act* operation, then we say that the caste operation *act* is safe for agent *a*.

Theorem5. If agent *a* follows the dynamic binding mechanism, then the *join*, *quit*, *activate* and *deactivate* caste operations are 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.

5. Conclusion

Dynamic agents that typically exhibit various behaviors in their lifetime are widespread in complex MASs. In the past years, many attempts have been made to support the development of such agents. However, it is still a challenge and open problem in the literature of AOSE. In this paper, we present an approach of dynamic binding mechanism to model and design dynamic agents. We adopt caste as abstraction to specify agents' behaviors and as modular unit to implement dynamic agents. Our approach permits agents to bind multiple castes and the caste that agent binds can be in active or inactive state. The dynamic behaviors of agents are interpreted and realized as the change of agents' casteships in their lifecycles, which can be specified and implemented by four atomic operations on castes. The semantics of dynamic binding mechanism and caste operations are rigorously defined based on the temporal logic integrating with dynamic operators. Some important properties of dynamic binding mechanism are formally specified and discussed. Our approach to dynamic agents differs from the approach proposed in [2] as we permit multiple castes to be bound at a moment and the *join* operation actually integrates with the *enact* and *activate* operations. There is no explicit gap between caste specification and agent design.

Acknowledgements. The authors acknowledge supports from Natural Science Foundation of China (60373022, 90612009), 973 project of China (2005CB321802), 863 project of China (2005AA113130), and science foundation of Huawei Corporation.

References

1. Xinjun Mao, Eric Yu, Organizational and Social Concepts in Agent oriented Software Engineering, Proceedings of AOSE, LNCS 3382, 1-15, 2005.
2. Mehdi Dastani, M. Birna van Riemsdijk, Joris Hulstijn, Frank Dignum, John-Jules Ch. Meyer, Enacting and Deacting Roles in Agent Programming, Proc. of AOSE V, LNCS 3382, 189-204, 2005.
3. Shan, L. and Zhu, H., CAMLE: A Caste-Centric Agent-Oriented Modelling Language and Environment, Proc. of Software Engineering for Multi-Agent Systems III: Research Issues and Practical Applications, LNCS 3390, 144-161, Springer, 2005.
4. Zhu, H., and Lightfoot, D., Caste: A step beyond object orientation, Proc. of JMLC, LNCS 2789, 59-62, 2003.
5. F. Zambonelli, N. R. Jennings, and M. Wooldridge, Developing Multiagent Systems: The Gaia Methodology, ACM Transactions on Software Engineering Methodology, 12(3), 317-370, 2003.
6. N.R.Jennings, An agent-based approach for building complex software systems, Communication of ACM, 44(4), 35-41, 2001.
7. E. Yu, Agent-Oriented Modelling: Software Versus the World, Proc. Of AOSE, LNCS 2222, 206-225, 2001.
8. J. Ferber and O. Gutknecht. A meta-model for the analysis and design of organizations in multi-agent systems, Proc. of 3rd International Conference on MultiAgent Systems, 128-135, 1998.
9. Franco Zambonelli, Andrea Omicini, Challenges and Research Directions in Agent-Oriented Software Engineering, Autonomous Agent and Multi-Agent Systems, 9, 253-283, 2004.
10. Michael Luck, Peter McBurney and Chris Preist, Agent Technology: Enabling Next Generation Computing, A Roadmap for Agent-based Computing, http://www. agentlink.org, 2005.
11. F Giunchiglia, John Mylopoulos and A Perini, The Tropos Development Methodology: Processes, Models and Diagrams, Proc. Of AAMAS, 35-36, 2002.
12. Zhu Hong, A formal specification language for agent-oriented software engineering, Proc. of AAMAS, 1174 -1175, 2003.
13. Thomas Juan, Adrian Pearce and Leon Sterling, ROADMAP: Extending the Gaia Methodology for Complex Open System, Proc. of AAMAS, 3-10, 2002.
14. Bauer, B., Muller, J.P., and Odell, J., Agent UML: a formalism for specifying multiagent software systems, Proc. Of AOSE, LNCS 1957, 91-103, 2001.
15. J. Odell, H. V. D. Parunak, S. Brueckner, J. Sauter. Temporal aspects of dynamic role assignment, Proc. Of AOSE, LNCS 2935, 201-213, 2003.
16. Wang, J., Shen, R., Zhu, H. Agent Oriented Programming based on SLABS, Proc. of COMPSAC, 2005.