Towards An Agent-Oriented Paradigm of Information Systems

Hong Zhu

Department of Computing, Oxford Brookes University,

Wheatley Campus, Oxford, OX33 1HX, UK

Tel: 0044 1865 484580, Fax: 0044 1865 484545

Email: hzhu@brookes.ac.uk

Abstract

This chapter presents a meta-model of information systems as a foundation for the methodology of caste-centric agent-oriented software development, which is suitable for applications on the Internet/Web platform and the utilisation of mobile computing devices. In the model, the basic elements are agents classified into a number of castes. Agents are defined as active computational entities that encapsulate (a) a set of state variables, (b) a set of actions that the agents are capable of performing, (c) a set of behaviour rules that determine when the agents will change their states and when to take actions, and (d) a definition of their environments in which they operate. Caste is the classifier of agents and the modular unit of the systems. It serves as the template that defines the structure and behaviour properties of agents as class does for objects. Agents can be declared statically or created dynamically at runtime as instances of castes. This chapter also illustrates the advantages of agent-oriented information systems by an example.

Introduction

The recent years has seen a rapid change in the hardware infrastructure and software platforms on which information systems operate. Notably, the Internet/Web as well as mobile devices such as notebook computers, PDA and 3G mobile phones and wireless networks, etc. are becoming ubiquitous. Proposals for effective utilisation of such flexible devices and the Internet infrastructure have been advanced, such as web services, semantic web and grid computing, and so on. These techniques provide a bright vision for the future computer applications, especially for management information systems. However, a big problem remains open, i.e. how software should be developed.

In the past two decades, object-orientation (OO) has been a successful mainstream paradigm for the analysis, design and implementation of software, especially information systems. However, software engineers are currently confronted with a number of challenges in the development of web-based information systems, especially in the construction of service-oriented systems, due to the new features of the Internet and WWW. One of the main challenges comes from the autonomous feature of the hardware and software resources on the Internet/Web. It is unnatural to model autonomous resources within the OO meta-model, which considers everything as objects.

In the past two decades, agent technology has been developed mostly as an artificial intelligence endeavour; Cf. (Huhns & Singh, 1997). It is partly inspired in the observations and modelling of autonomous and emergent behaviours in the societies of human beings or insects and animals. It has long been regarded as a viable solution to the development of complicated applications in dynamic environments such as the Internet (Jennings & Wooldridge, 1998). However, existing agent-based systems have been developed in *ad hoc* methods without proper methodology, language and tool supports. It is widely recognised that the lack of rigour has hampered the wide adoption of agent technology in IT industry.

In this chapter, we adapt and extend the principles of OO and propose a new meta-model of information systems based on the concept of agents. We will first present a meta-model of such agent-oriented information systems (AOIS), and then demonstrate the features of AOIS with an example and compares agent-orientation (AO) with traditional approaches. The readers are referred to (Shan, *et al*. 2006) for the aspects on the methodology, languages and tools that support the development of such AOIS.

The remainder of the chapter is organised as follows. We will first give an informal introduction to AOIS. It is followed by a formal definition of the meta-model. We will then illustrate the features of AOIS with an example and compares it with traditional approaches to the development of information systems. Finally, we conclude the chapter with a discussion of the related works and further work.

Basic concepts

In our conceptual model, the basic unit that forms an information system is agent. Because there is no widely accepted definition of the concept of agent and multi-agent systems (MAS), it is worthy spending a few words to clarify what we mean by agent and MAS and how such systems work. Our conceptual model can be characterized by a set of pseudo-equations. Each pseudo-equation defines a key feature of MAS.

The structures and operations of agents and multi-agent systems

Pseudo-equation (2.1) states that agents are defined as real-time active computational entities that encapsulate data, operations and behaviours, and situate in their designated environments.

$$Agent = \langle Data, Operations, Behaviour \rangle_{Environment} \qquad (2.1)$$

Here, data represent an agent's state. Operations are the actions that the agent can take. Behaviour is described by a set of rules that determine how the agent behaves in the context of its designated environment. By encapsulation, we mean that an agent's state can only be changed by the agent itself. Figure 1 illustrates the control structure of agent's behaviour.

```
Begin
    Initialise state;
    Loop
        Perceive the visible actions and states of the agents in its
            environment;
        Take actions and change state according to the situation in
            the environment and its internal state;
    end of loop;
end
```

There is a fundamental difference between objects and agents. In the structure of objects, there is no explicitly programmed behaviour rule. Instead, there is a fix behaviour rule for all objects, i.e. to execute a method if and only if it receives a message that invokes the method. In contrast, agents' behaviours are not simply driven by messages, although they can be so. For example, an agent can have a behaviour rule that enables it to take an action when it has not received any message for a certain period of time. Moreover, when an agent receives a message that requests the agent to take a specific action, the agent can decide whether to do so, for instance, according to its internal state or the origin of the request. In other words, the agent can decide 'when to go' and 'whether to say no' according to an explicitly specified set of behaviour rules. In this sense, an agent can be not only reactive as driven by the external events, but also be proactive, i.e. being able to initiate interactions with the outside.

Despite of this fundamental difference, objects can be considered as agents in a degenerate form as argued in (Zhu, 2001a). In particular, object is a special case of agent in the sense that it has a fixed rule of behaviour, i.e. "execute the corresponding method when receives a message". Consequently, in our conceptual model, a MAS consists of agents and nothing but agents as stated in pseudo-equation (2.2).

$$MAS = \{Agent_n\}, n \in Integer \tag{2.2}$$

Organisations of multi-agent systems

In our conceptual model, the classifier of agents is called *caste*. Caste is the basic building blocks in the design and implementation of MAS. As a modular language facility, a caste serves as a template that describes the structure and behaviour properties of agents. Pseudo-equation (2.3) states that a caste at time *t* defines a set of agents that have the same structural and behavioural characteristics.

$$Caste_t = \{Agent \mid Structure \,\&\, Behaviour\ properties\} \tag{2.3}$$

While a collection of castes represent various types of participants in the problem domain, the structure of the problem domain is captured with certain relationships between castes.

Caste membership and migration relations

Agents are classified into various castes in the way similar to that data are classified into types and objects are classified into classes. In other words, agents are instances of castes just like objects are instances of classes. In the development of AOIS, caste is the modular programming unit that serves as the template of agents so that agents can be created as instance of a caste at runtime or declared statically. When an agent is created as an instance of a caste, it will have all the structural and behavioural features defined by the caste.

However, different from the notion of class in OO, caste allows dynamic classification. That is, an agent can change its caste membership (called *casteship* in the sequel) at run-time. The weakness of static object-class relationship in current mainstream OO programming has been widely recognized. Proposals have been advanced, for example, to allow objects' dynamic reclassification (Drossopoulou *et al.*, 2002). In our model, dynamic classification is an integral part of agents' behaviour capability, which can be naturally represented through agents' behaviour rules to change its casteship. An agent can take an action to join a caste or retreat from a caste at run-time. When an agent retreats from a caste, it will lose the structural and behavioural features of the caste. When it joins a caste, it will obtain the structural and behavioural features. Dynamic casteship allows users to model the real world with MAS naturally and to maximize the flexibility and power of agent technology. For example, Zhu and Lightfoot (2003) demonstrated that agents' ability to dynamically change their roles can be naturally represented by dynamic casteship.

A migration relation that represents agents' dynamic casteship can, therefore, be defined on castes. There are two types of migration relations, 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 then quit from caste *A*. For example, in a university information system, we would have castes MSc_Student, PhD_Student and Staff, etc. to represent the components that collect the information about and deliver the services to various types of users. A participate relation from PhD_Student to Staff can be defined to represent the possibility that a PhD student may be employed as a staff, such as a teaching assistant, while registered as a student. When an agent of PhD_Student joins the Staff caste, it will obtain all the structural and behavioural features of the caste Staff without losing its original structural and behavioural features defined in the caste PhD_Student. For example, suppose that the Staff caste defines a state variable Salary while PhD_Student does not. When an agent of PhD_Student joins Staff, it will have an additional state variable Salary as defined in the Staff caste. A migrate relation can be defined from caste MSc_Student to PhD_Student since a MSc student may become a PhD student after graduation, but it cannot be a MSc student and a PhD student at the same time. The agent will lose all the structural and behavioural features defined in the caste MSc_Student, but obtain all the structural and behavioural features that are defined by the PhD_Student caste. Suppose that the PhD_Student caste has a state variable Office_Address, while the MSc_Student caste has a state variable Laboratory. When an agent of MSc_Student moves to PhD_Student, it will lose the variable Laboratory and obtain a new variable Office_Address.

Inheritance relation

Inheritance relations can be specified between castes. A caste *A* inherits caste *B* means that any agents of caste *A* have all structural, behavioural and environmental features of caste *B*. Our model also allows multiple classifications, i.e. an agent can belong to more than one caste at the same time. Consequently, a caste can inherit more than one caste.

Whole-part relations

In our model, an agent may contain a number of components that are also agents. The former is called compound agent of the latter. In such a case, there exists a whole-part relationship between the compound and the component agents. We identify three types of whole-part relationships between agents according to the ways a component agent is bound to the compound agent.

The strongest binding between a compound agent and its components is *composite.* A composition relation from caste *A* to caste *B* means that an agent *b* in caste *B* contains an agent *a* in caste *A*. The compound agent *b* is responsible for creation and destruction of its component *a*. If the compound agent *b* no longer exists or quits from caste *B*, the component agent *a* will be destroyed and hence not exist.

The weakest binding is *aggregate*. An aggregate relation from caste *A* to caste *B* means that, although an agent *b* in caste *B* contains a component agent *a* of caste *A*, the component agent *a* is independent of the compound agent *b* in the sense that *b* does not affect *a*'s existence or casteship. If agent *b* is destroyed or quits from caste *B*, agent *a* can still survive and be a member of caste *A*.

The third whole-part relation is called *congregate*. It means that if the compound agent is destroyed, the component agent will still exist, but it will lose the casteship of the component caste. For example, a university consists of a number of individuals as its members. If the university is destroyed, the individuals should still exist. However, they will lose the membership of the university. Therefore, in the university information system, the whole-part relationship between the caste University and the caste University Member is a congregation relation. This relationship is different from the relationship between a university and its departments. Departments are components of a university. If a university is destroyed, its departments will no long exist. The whole-part relationship between the castes University and Department is therefore a composition relation. The composition and aggregation relation in our conceptual model is similar to the composition and aggregation in UML, respectively. However, congregation is a novel concept, which has not been recognized in the research on OO modelling of whole-part relations; cf. (Barbier, 2003).

Communications and environment

In our conceptual model, an agent's state variables and actions are 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, which are also agents. Similarly, the value of a visible state variable can be observed by other agents, while the value of an internal state can only be observed by its components. Notice that, our use of the term 'visibility' is different from the traditional concept of scope used in OO languages.

The concept of visibility of an agent's actions and state variables forms the basic communication mechanism in our conceptual model. Agents communicate with each other by taking visible actions and changing visible state variables, and by observing other agents' visible actions and visible states, as shown in pseudo-equation (2.4).

$$A \to B = A.Action \,\&\, B.Observation \qquad (2.4)$$

An agent's visible actions are not necessarily observed by all agents in the system. They are only observed by those agents who are interested in the agent's behaviour and regard the agent as a part of their environments. The environment of an agent in a MAS at time moment *t* is a subset of the agents in the system. As illustrated in pseudo-equation (2.5), from a given agent's point of view, only those in its environment are visible. In particular, from agent *A*'s point of view, agent *B* is visible means that agent *A* can perceive the visible actions taken by agent *B* and obtain the value of agent *B*'s visible state variables at run-time.

$$Environment_t(Agent, MAS) \subseteq MAS - \{Agent\} \qquad (2.5)$$

In our model, the environment of an agent is required to be explicitly defined so that the agents in a MAS are designed and implemented with a *designated environment*. In other words, the environment of an agent is specified but allowed to vary within a certain range when an agent is designed. We can describe the environment of a caste, for example, as the set of agents in a number of particular castes. An environment such described is neither closed, nor fixed, nor totally open. Once an agent joins a caste, its environment is combined with the specified environment of the caste. Hence, the agents in the caste's environment become visible. The environment also changes when other agents join the caste in the agent's environment.

Formal definition of the meta-model

We now formally define the conceptual model using mathematical notions and notations. Readers can skip over this section if not interested in the formal treatment of the subject.

Multi-agent systems

Agents behave in real-time concurrently and autonomously. A time moment is an element in a time index set *T*, which is defined as a subset of real numbers $[t_0, \infty)$, i.e. $T = \{t \mid t \in R \,\&\, t > t_0\}$, where t_0 can be any real number. The structure of agents consists of four elements, i.e. each agent A is 4-tuple $\langle S_{A,t}, \Sigma_{A,t}, R_{A,t}, E_{A,t} \rangle$, where

(a) $S_{A,t}$ *is the state space*. We also write $S^V_{A,t}$ and $S^I_{A,t}$ to denote the visible and internal parts of the state space $S_{A,t}$, respectively. Thus, $S_{A,t} = S^V_{A,t} \times S^I_{A,t}$.

(b) $\Sigma_{A,t}$ *is the set of actions* that the agent is capable of performing. We write $\Sigma^V_{A,t}$ and $\Sigma^I_{A,t}$ to denote the sets of visible and internal actions, respectively. Thus, $\Sigma_{A,t} = \Sigma^V_{A,t} \cup \Sigma^I_{A,t}$, where $\Sigma^V_{A,t} \cap \Sigma^I_{A,t} = \varnothing$.

(c) $R_{A,t}$ *is the set of behaviour rules* that determine how the agent changes its state and which action to take at what circumstances.

(d) $E_{A,t}$ *is the designated environment of the agent.*

In general, the set of agents in a MAS may change during execution as agents may be quit from the system or join the system at runtime. Let $\{A_1, A_2, \cdots, A_n\}_t$ be the set of agents in the system at time moment *t*. These agents are members of castes $C_1, C_2, \cdots, C_m$. The *casteship* of agent *A* at time moment *t* to caste *C* is denoted by $A \in_t C$. We write $Caste_t(A)$ to denote the set of castes that agent *A* belongs to at time moment *t*, i.e. $Caste_t(A) = \{C \mid A \in_t C\}$. An agent can join a caste *C* by taking the action of *JOIN*(*C*) and quit from a caste *C* by taking the action *QUIT*(*C*).

An inheritance relation between castes is defined as partial ordering relation denoted by $\prec$. We have that at all times *t*, $A \in_t C \wedge C \prec C' \Rightarrow A \in_t C'$. We assume that the set of castes and the inheritance relations between them do not change at runtime. Each caste describes its agents' structure, behaviour and environment in the form shown in Figure 2.

```
Caste $C$ <= $C_1, ..., C_k$;  (* inheritance relationship*)
    ENVIRONMENT $E_1, ..., E_w$;  (*description of the environment*)
    VAR       *$v_1$:$T_1$, ..., *$v_m$:$T_m$;  (* visible state variables *)
              $u_1$:$S_1$, ..., $u_l$:$S_l$; (* invisible state variables *)
    ACTION    *$A_1(p_{1,1}, ..., p_{1,n1})$, ..., *$A_s(p_{s,1}, ..., p_{s,ns})$;  (* visible actions *)
              $B_1(q_{1,1}, ..., q_{1,m1})$, ..., $B_t(q_{t,1}, ..., q_{t,mt})$; (* invisible actions*)
    RULES     $R_1, R_2, ..., R_h$  (* Behaviour rules *)
End $C$.
```

Figure 2. Structure of caste description

In Figure 2, the clause '$C \Leftarrow C_1, C_2, \cdots, C_k$' specifies that caste *C* inherits castes $C_1, C_2, \cdots, C_k$. Therefore, $C \prec C_i, i = 1, \cdots, k$. The VAR clause declares a set of state variables of the caste in addition to what it inherits, where the visible variables $S^V(C)$ and invisible variables $S^I(C)$ of the caste *C* are:

$$S^V(C) = \{v_1 : T_1, \cdots, v_m : T_m\} \cup \bigcup_{i=1}^{k} S^V(C_i) \tag{3.1}$$

$$S^I(C) = \{u_1 : S_1, \cdots, u_j : S_j\} \cup \bigcup_{i=1}^{k} S^I(C_i). \tag{3.2}$$

The ACTION clause defines a set of actions of the caste. The visible actions $\Sigma^V(C)$ and invisible actions $\Sigma^I(C)$ of caste *C* are:

$$\Sigma^V(C) = \{A_1(p_{1,1}, \cdots, p_{1,n_1}), \cdots, A_s(p_{s,1}, \cdots, p_{s,n_s})\} \cup \bigcup_{i=1}^{k} \Sigma^V(C_i) \tag{3.3}$$

$$\Sigma^I(C) = \{B_1(q_{1,1}, \cdots, q_{1,m_1}), \cdots, B_t(q_{t,1}, \cdots, q_{t,m_t})\} \cup \bigcup_{i=1}^{k} \Sigma^I(C_i). \tag{3.4}$$

It is assumed that variables and action identifiers are unique in each caste declaration. Duplicated declarations of identifiers in a caste are not allowed.

The environment of the agents of a caste is explicitly specified in the ENVIROMENT clause in the following forms:

(a) 'agent name' indicates a specific agent in the system;

(b) 'All: caste-name' means all the agents of the caste;

(c) 'agent-variable: caste-name' is a variable that ranges over the caste. It can be assigned to any agent in the caste.

Let $ENV_t(C)$ denote the environment for caste *C* at time moment *t*, we have that

$$ENV_t(C) = \bigcup_{i=1}^{w} [\![E_i]\!]_t \cup \bigcup_{i=1}^{k} ENV_t(C_i), \tag{3.5}$$

where $E_i, i = 1, \cdots, w$ are the environment description clauses in caste *C*'s definition, $[\![E]\!]_t$ denote the semantics of an environment description clause *E* at time moment *t*. $[\![E]\!]_t$ is defined as follows.

$$[\![agent]\!]_t = \begin{cases} \{agent\}, & \text{if } agent \text{ is in the system at time } t; \\ \varnothing, & \text{if } agent \text{ is not in the system at time } t. \end{cases} \tag{3.6}$$

$$[\![All : Caste]\!]_t = \{X \mid X \in_t Caste\}\ ; \tag{3.7}$$

$$[\![x : C]\!]_t = \begin{cases} \{A\}, & \text{if } x = A \text{ and } A \text{ is in the system at time } t; \\ \varnothing, & \text{otherwise.} \end{cases} \tag{3.8}$$

The set of rules *RULE*(*C*) that agents of caste *C* must obey is

$$RULE(C) = \{R_1, R_2, \cdots, R_h\} \cup \bigcup_{i=1}^{k} RULE(C_i)\,. \tag{3.9}$$

Let *A* be any given agent and $Caste_t(A) = \{C_1, C_2, \cdots, C_n\}$, the following equations define the structural and behavioural properties of the agent at each time moment *t*.

$$S^V_{A,t} = \bigcup_{i=1}^{n} S^V(C_i)\,, \text{ and } S^I_{A,t} = \bigcup_{i=1}^{n} S^I(C_i)\,. \tag{3.10}$$

$$\Sigma^V_{A,t} = \bigcup_{i=1}^{n} \Sigma^V(C_i)\,, \text{ and } \Sigma^I_{A,t} = \bigcup_{i=1}^{n} \Sigma^I(C_i)\,. \tag{3.11}$$

$$E_{A,t} = \bigcup_{i=1}^{n} ENV_t(C_i)\,. \tag{3.12}$$

$$R_{A,t} = \bigcup_{i=1}^{n} RULE(C_i)\,. \tag{3.13}$$

Dynamic semantics

A *run r* of a MAS is a mapping from time *T* to the set $\prod_{i=1}^{n} S_{A_i,t} \times \Sigma_{A_i,t}$. The behaviour of a MAS is defined by the set *R* of possible runs. For any given run *r* of the system, a mapping *h* from *T* to $S_{A,t} \times \Sigma_{A,t}$ is a run of agent *A* in the context of *r*, if $\forall t \in T.h(t) = r_A(t)$, where $r_A(t)$ is the restriction of *r*(*t*) on $S_{A,t} \times \Sigma_{A,t}$. 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 a MAS has the following properties.

- Actions are instantaneous, i.e. for all $t_1, t_2 \in T$, if $t_1 \neq t_2, r_A(t_1)$ is regarded as different from $r_A(t_2)$.
- An agent may take no action at a time moment *t*, i.e. the agent is silent at time *t*. We use τ to denote silence.
- The actions taken by an agent are separable, i.e. for all runs *r*, all agents *A*, there exists a real number ε>0 such that for all *t*, $r^C_A(t) \neq \tau \Rightarrow (\forall x \in T.(t < x \leq t + \varepsilon \Rightarrow r^C_A(x) = \tau))$, where $r^C_A(t)$ denotes the action taken by agent *A* at time moment *t* in the run *r*.

With above assumptions, we can prove that an agent can take at most a finite number of non-silent actions in any finite period of time and a countable number of non-silent actions in its lifetime.

Notice that, the global state S_g of the system at time t is the value of $\prod_{i=1}^{n} S_{A_i,t} \times \Sigma_{A_i,t}$. However, each agent A can only view the visible states and actions of the agents in its environment, i.e. the part of S_g in the space $\prod_{X \in E_{A,t}} S^V_{X,t} \times \Sigma^V_{X,t}$.

A behaviour rule R for agent A defines a predicate P_R on the set of all possible execution histories of the agent in the context of the system's runs. Here, the context is the agent's view of the history of the environment. Therefore, such a possible history is a mapping from the time $t \in T$ to the set $S_{A,t} \times \Sigma_{A,t} \times \prod_{X \in E_{A,t}} \left(S^V_{X,t} \times \Sigma^V_{X,t} \right)$. For a possible history $\psi_{A,t}$ of agent A up to time moment t, $P_R(\psi_{A,t}) = true$ means that agent A's behaviour at time moment t satisfies the behaviour rule R. An execution r_A of agent A in the context of a run r is valid, if for all time moment t and all behaviour rules $R \in R_{A,t}$, agent A's behaviour at time t satisfies rule R. A run r of a MAS M is valid, if all agents A's behaviours are valid in r.

The meta-model does not define how a behaviour rule should be defined. In fact, languages at different levels of abstraction can have their own ways of defining behaviour rules. For example, in the formal specification language SLABS, a behaviour rule are defined in the following form.

<Pattern> | [<Probability>] —> <Action>, [***if*** <Scenario>]; [***where*** <Pre-condition>]

where <Pattern> defines a pattern of the agent's behaviour so far, <Scenario> specifies the scenario in the environment, <Pre-condition> specifies the pre-condition that the rule applies, <Action> specifies the action to be take by the agent, and <Probability> defines the probability that the agent will take the action as specified. The formal semantics of such behaviour rules and a formal system for reasoning about the behaviour of MAS can be found in (Zhu, 2005). In modelling language CAMLE, behaviour rules are defined in the form of behaviour diagrams (Shan & Zhu, 2004, 2005). In programming language SLABSp, the behaviour rules take the form of a conditional statement that uses a scenario as the guard condition (Wang, Shen & Zhu, 2005).

Illustrative example

Generally speaking, information systems are systems that collect, store, process and use information to fulfil certain tasks and provide certain services. Different paradigms of software and information system development methods differ in the way how the information processing functions and storage facilities are structured, organised and used, and how such systems are constructed and evolved accordingly.

From the structure point of view, traditional *structured methods* separate the storage of information from their processing methods. Functions are hierarchically decomposed into sub-functions and then sub-sub-functions, and so on. In *object-oriented methods*, a set of related data and their processing methods are encapsulated into one computational entity called object. The relationships between objects resemble their counterparts in the real world. The classification of objects into classes represents the structural similarity between objects. Inheritance relation represents specialisation and dissimilarities. Method invocation association represents functional dependences. Whole-part relations represent hierarchical structural decomposition. These relations help the maintenance and evolution of information systems as their real world environment evolves as discussed in the introduction of the chapter. However, objects in the mainstream OO paradigm are passive entities. Models of information systems that consist of a large number of active and autonomous information processing components cannot be represented naturally and close to the real world counterparts in the structure. In the meta-model of *caste-centric AO method* proposed in this chapter, a set of data, their processing methods and the rules on how the processing methods are to be used are encapsulated into one computational entity called agent to represent an active information processing element. The relationships between such elements and their environments are represented in the classification of agents for the similarity in their structural and behaviour features, in inheritance relations for the dissimilarity and specialisation, in whole-part relations for structural decomposition, in visibility in the environment for communications and collaborations with each other, and so on. This model inherits many feature of OO, but further captures the features of active information processing elements in modern information systems enabled by the availability of mobile computing devices and the Internet/Web-based software platform.

In this section, we will present an illustrative example to highlight the differences between traditional approaches (such as structured and OO approaches) and the AO approach.

Suppose that a summer school is organised to teach a number of classes. Some classes are scheduled to be held at the same time, but at different classrooms. Students can choose the classes to attend. Students may be unfamiliar with the site where classes are. An information system is required to help students go to the right classrooms.

Solution of structured methods

In a structured methodology such as stepwise refinement, one would decompose the functionality of the system and find the steps to calculate the results from the input. For example, suppose the system is to be used by an instructor for helping student in his/her class to find the next classroom. One might come up with a solution similar to the following.

```
Begin
        1. Get the times and locations of the classes
        2. Get the class lists of the students
        3. Input the current class
        4. Get a list of students in the class
        5. For each person on the list Do:
                5.1. Find the next class that student is attending
                5.2. Find the location of the class
                5.3. Find the route from the classroom to the person's next class
                5.4. Tell the student how to get their next class
End
```

Figure 3. Pseudo-code in Structured Programming

Using a structured analysis and design method, one might obtain the following data flow diagram shown in Figure 4.

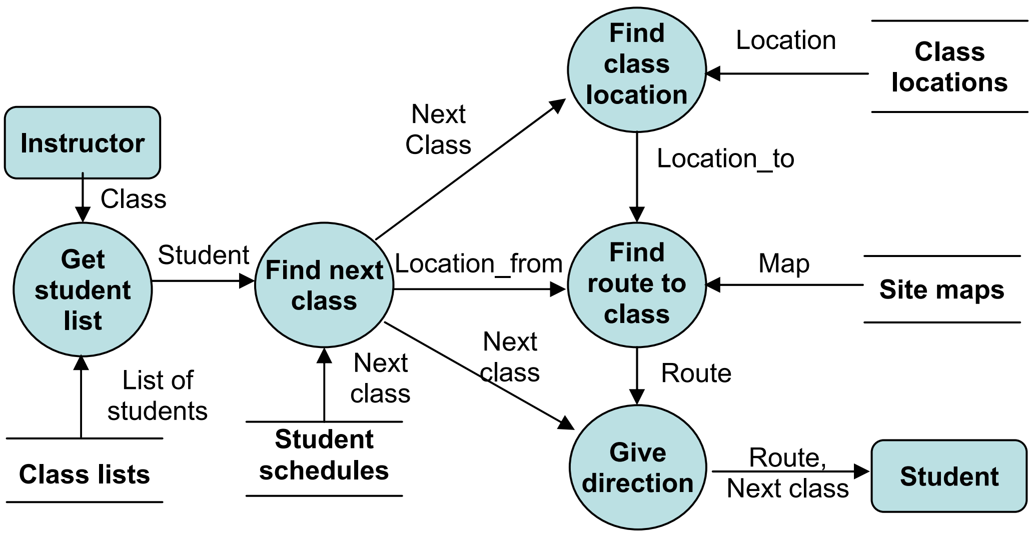
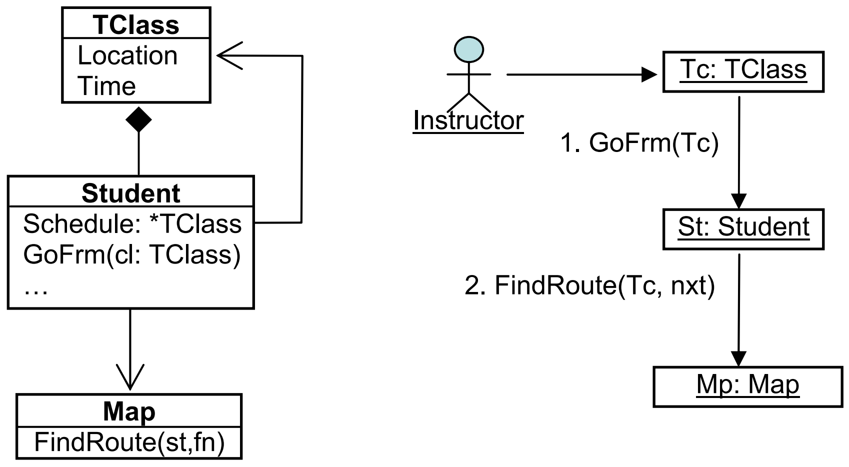


Figure 4. Data flow diagram

Object-oriented solution

As Shalloway and Trott (2002) pointed out, although the above may be a good solution for computer systems, it is unnatural for a person to do. In other words, the model given in Figure 4 does not represent the real world. A better solution would be that the instructor post directions to go from this classroom to the other classrooms and then inform everybody that the directions are posted at the back of the classroom and tell everybody to follow the direction and go to the next classroom. The alternative solution can be represented in an OO model shown in Figure 5.



(a) Class diagram

(b) Collaboration diagram

Figure 5. Object-Oriented model in UML

Shalloway and Trott (2002) analysed the differences between the above two solutions and their impacts on software development. In addition to their analysis, we can also identify the following weakness of the OO solution. First, modelling students as objects means that they are passively driven by messages to perform actions of finding out route and then go to the next classroom. They are controlled by other objects. However, in the real world, students may have autonomous behaviours. They are not directly controlled by anybody. Therefore, the model still does not exactly represent the real world. Second, it is the developer's responsibility to implement all the components of the system although the implementation can be done by programming the code, reusing existing code, using COTS components, etc. Once implementation is done, it is not to be changed during execution of the system.

Agent-oriented solution

A caste-centric AO solution would contain three castes to represent students, instructors and the organizers of the summer school as shown in Figure 6. The organizers will be responsible to set the class schedule and provide the local knowledge. They are also responsible to inform the instructors and the students of the availability of the schedule and local knowledge. The instructors will teach the students. Teachers may also access the class schedules and local knowledge to determine when and where to go to the classrooms and teach which subjects. The students will access the class schedule and local knowledge in order to determine where to go and how to get there. It is worthy noting that the schedule and local knowledge can be objects, which are a degenerated form of agents as discussed in the previous section.

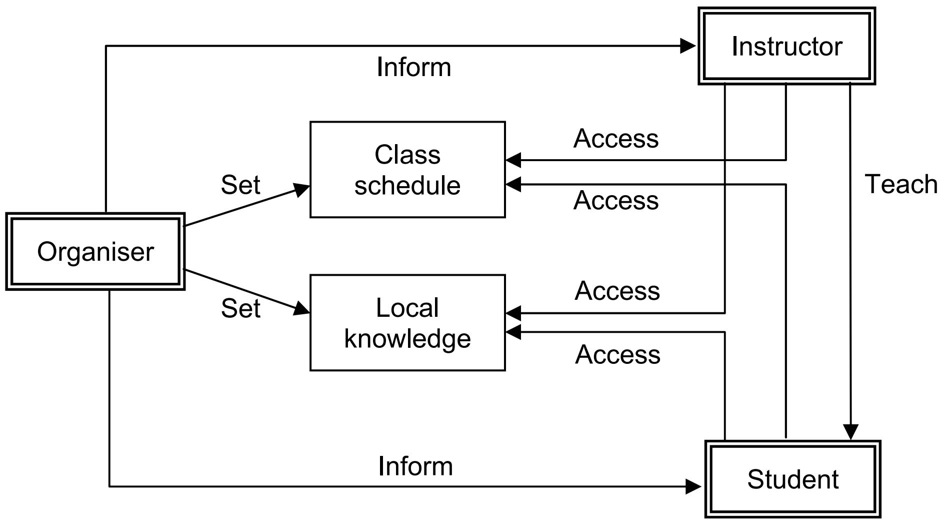


Figure 6. Collaboration diagram of AO solution

In comparison with the OO solution, the AO model is much closer to how the real world works. The main difference is in the release of instructors from the responsibility of controlling the students on when to find out the information about their next classes. The students can access the schedule and local knowledge whenever they like to do so. The impact of this difference is significant because this does not only gives the freedom to decide when to access the local knowledge and class schedule, but also the possibility to have freedom in how to access them because other parts of the system need to know much less details of the caste. As far as the local knowledge, such as a map of the campus, and class schedule are represented in a standard format, agents that represent students can be any program that understand them. Therefore, it can be programs running on students' notebook computers, hand-held computers, mobile phones, etc. The software running on these computing devices can be different products that each is suitable to the device that is owned by the student. This will significantly reduce the complexity of developing the system to make it suitable to run on many different hardware and software platforms. It will also enable the users to interact with the system in an interface that they are familiar with. It is unnecessary to integrate the programs that represent the student agents into the system before it is put into operation, because they can join the system at any time during the execution of the system. The software can be run on other servers on the Internet that provides, for example, services to display a map on the screen, to find routes in an electronic map and to provide scheduling and event reminder services. Therefore, the complexity and cost to develop the system can be significantly reduced. It is also easier for the system to evolve.

Conclusion

In this chapter, we presented a meta-model of AOIS, in which the basic elements are agents. The meta-model enables us to model software systems very close to information system in the physical world so that the software systems are easier to understand, more reusable and easier to modify. As illustrated in the example given in section 4, AO approach is suitable to the platform of Internet/Web and the utilisation of mobile devices.

In the literature of AO software engineering, there are a number of proposals to AO methodologies aiming at developing MAS (Zambonelli, Jennings & Wooldridge, 2003; Bresciani *et al*. 2004; Burrafato & Cossentino, 2002; Zambonelli & Omicini, 2004). A few formal models of agents and MAS have also been developed and investigated; Cf. (Myer & Schobbens, 1999). Among the most well-known formal models is the mentalistic model of BDI agents, in which each agent has mental states of belief, desire and intention (Rao & Georgreff, 1991). The logic properties of mental states were formally studied in the framework of modal logics, e.g. (Wooldridge, 2000). Such models are suitable for developing artificial intelligence applications. Application of them to information systems may need a revolutionary change in software development paradigm. d'Inverno and Luck (2003) formally defined various concepts related to agents and MAS in the formal specification language Z. They regard agents as special cases of objects. This is the opposite to our approach. Semi-formal definitions of meta-models have also been proposed by Bernon et al. (2005) and Odell, Nodine and Levy (2005). They provide syntactical descriptions of the structures of agents and related notions, but no definition of their semantics. These models tend to include a large number of concepts including, for example, roles, agent groups, agent societies, capabilities, responsibilities, goals, plans, organisations, etc. Further research on these concepts and their properties and interrelationships is necessary before they can be language facilities. Our approach is caste-centric, that is, caste plays the central role in our methodology. In comparison with other meta-models, our meta-model is much simpler and clearer. For example, most other methodologies have the notion of roles, which is not formally defined and is not a language facility. In our approach, agents that play the same role can be defined by a caste (Zhu, 2001b), which is a well-define language facility that can be implemented in a programming language (Wang, Shen & Zhu, 2005). It can also be used to implement other agent concepts naturally, such as agent societies, protocols and normative behaviours. The most important feature of our approach is that it is a natural evolution of the current mainstream paradigm of OO software development.

We are currently further investigating the design and implementation of AO programming languages based on the meta-model to support the development of service-oriented computing such as web services (Zhu & Shan, 2005). We are also further developing automated software tools to support the whole development process to bridge the gaps between models, specifications and their implementations on existing software platforms.

References

Barbier, F., Henderson-Sellers, B., Le Parc A. & Bruel J-M. (2003). Formalization of the whole-part relationship in the Unified Modeling Language. IEEE Trans. Software Eng. 29(5), 459-470.

Bernon, C., Cossentino, M., Gleizes, M-P., Turci, P. & Zambonelli, F. (2005). A study of some multi-agent meta-models. AOSE 2004. Springer, LNCS 3382, 62-77.

Bresciani, P., Perini, A., Giorgini, P., Giunchiglia, F. & Mylopoulos, J. (2004) Tropos: an agent-oriented software development methodology. Autonomous Agents and Multi-Agent Systems, 8, 203-236.

Burrafato, P. & Cossentino, M. (2002). Designing a multi-agent solution for a bookstore with the PASSI methodology. Proc. of AOIS-2002 (May 27-28, 2002, Toronto, Canada) at CAiSE'02.

d'Inverno, M. & Luck, M. (2003). Understanding Agent Systems. Berlin: Springer-Verlag.

Drossopoulou, S., Damiani, F., Dezani-Ciancaglini, M. & Giannini, P. (2002). More dynamic object reclassification: FickleII. ACM Trans. on Programming Language and Systems, 24(2), 153-191.

Huhns, M. & Singh, M.P. (Eds.) (1997). Readings in Agents. San Francisco: Morgan Kaufmann.

Jennings, N.R. & Wooldridge, M.J. (Eds.) (1998). Agent Technology: Foundations, Applications And Markets. Berlin: Springer.

Myer, J-J. & Schobbens, P-Y. (Eds.) (1999) Formal Models of Agents - ESPRIT Project ModelAge Final Workshop Selected Papers. LNAI 1760, Springer.

Odell, J., Nodine, M. & Levy, R. (2005). A metamodel for agents, roles and groups. AOSE 2004. Springer LNCS 3382, 78-92.

Rao, A. S. & Georgreff, M. P. (1991). Modeling Rational Agents within A BDI-Architecture. Proc. of the International Conference on Principles of Knowledge Representation and Reasoning, 473-484.

Shalloway, A. & Trott, J. (2002). Design Patterns Explained. Addison and Wesley.

Shan, L., Shen, R., Wang, J. & Zhu, H. (2006) Caste-centric development of agent oriented information systems. Chapter ???, in this book.

Wang, J., Shen, R. & Zhu, H. (2005). Agent oriented programming based on SLABS. Proc. of COMPSAC'05 (July, 2005, Edinburgh, UK), 127-132.

Wooldridge, M. (2000). Reasoning About Rational Agents. The MIT Press.

Zambonelli, F. & Omicini, A. (2004). Chanllenges and research directions in agent-oriented software engineering. Autonomous Agents and Multi-Agents Systems, 9, 253-283.

Zambonelli, F., Jennings, N. R. & Wooldridge, M., Developing multiagent systems: the Gaia methodology, *ACM Transactions on Software Engineering and Methodology,* Vol. 12, No.3, 2003, pp317-370.

Zhu, H. & Lightfoot, D. (2003). Caste: A step beyond object orientation, in Modular Programming Languages. Proc. of JMLC'2003 (Aug. 2003, Austria), Springer LNCS 2789, 59-62.

Zhu, H. (2001a). SLABS: A Formal Specification Language for Agent-Based Systems. International Journal of Software Engineering and Knowledge Engineering, 11(5), 529-558.

Zhu, H. (2001b). The role of caste in formal specification of MAS. Proc. of PRIMA'2001 (July 2001, Taipei), Springer LNCS 2132, 1-15.

Zhu, H. (2003). A formal specification language for agent-oriented software engineering. Proc. of AAMAS'2003 (July, 2003, Melbourne, Australia), 1174- 1175.

Zhu, H. (2005). Formal Reasoning about emergent behaviour in MAS. Proceedings of SEKE'05 (July 14~16, 2005, Taipei), 280-285.

Zhu, H. and Shan, L. (2005) Agent-Oriented Modelling and Specification of Web Services, Proc. of WORDS'2005 (February 2-4, 2005, Sedona, USA), 152-159.