# Agent-Oriented Modelling and Specification of Web Services

Hong Zhu
*Dept of Comp., Oxford Brookes Univ.*
*Oxford OX33 1HX, UK*
*Email: hzhu@brookes.ac.uk*

Lijun Shan
*Dept of Comp. Sci., National Univ. of Def. Tech.*
*Changsha, 410073, China*
*Email: lijunshancn@yahoo.com*

## Abstract

*Web services (WS) provide a technology for integrating applications over the Internet. This paper proposes a multi-agent conceptual model of WS and an agent-oriented modelling and formal specification method to address the difficulties in developing WS applications. The paper presents a graphic model of the general architecture of WS in agent-oriented modelling language CAMLE and an abstract specification in the formal specification language SLABS. It also illustrates how agent-oriented modelling and formal specification can be applied to the development of WS applications by an example of online auction WS and its requester application. It is shown that models and formal specifications enable software engineers to specify not only the service provider's functionality and behaviour, but also the requirements and restrictions on service requesters' behaviour. Such semantic information is crucial for the success of dynamic integration of WS.*

## 1. Introduction

Web services (WS) is characterised by the dominant of program-to-program interactions [1]. In view of the infrastructure of WS becoming pervasive, a new paradigm of service-oriented computing is emerging. In comparison with other distributed computing techniques, such as CORBA, Java RMI and DCOM, it does not only offer more flexibility and looser coupling so that it is more suitable for internet computing [2], but it is also fundamentally different from them [3]. The components of WS applications, such as service providers, are autonomous, active and persistent computational entities that control their own resources and their own behaviour. They have social ability and collaborate with each other through dynamic discovery and invocation of services. Entities having these features have been studied in AI community as agents, c.f. [4]. In this paper, we propose a conceptual model of WS in which computational entities that provide or request services are regarded as agents. Here, the word 'agent' has the same meaning as in 'real estate agents' where agents provide services to clients for buying or selling properties, and 'travel agents' where agents provides clients with the services of purchasing transportation tickets and booking hotels, etc.

The description of the semantics of WS components is of particular importance. First, the components in a WS application are usually developed by different vendors. Developers of service providers and those of the service requesters are usually separated by space and time. The most effective channel of communication between them is perhaps through documentation. Second, WS technology enables dynamic software integration at runtime. It does not only require the interface between integrated entities syntactically compatible, but more importantly, the interactions must be semantically correct. To enable dynamic search of services, informal documentation of services is insufficient. It has been recognised that in addition to the descriptions of the syntactical aspects of WS, such as the formats of the messages and the parameter types of each service, the description of semantic aspects, such as business processes, is of significant importance for the success of WS technology [5, 6]. However, the existing standards are inadequate to describe the semantics of WS. Solutions to the problem proposed in the literature rely on ontology for taxonomic descriptions of the functionality of each service, and workflow descriptions for the restrictions on the orders that services are called; see e.g. [7, 8]. Questions remain that whether ontology and workflow descriptions are expressive enough to provide the required semantic information for the development and dynamic discovery and invocation of WS. In [9], we proposed the uses of an agent oriented formal specification language SLABS to formally specify the semantics of WS components [10]. In this paper, we further develop the method by using an agent-oriented modelling language CAMLE [11] to develop WS applications. Graphic models are easier to construct and more readable than formal specifications. They are then automatically transformed into formal specifications in SLABS by our modelling tools.

The paper is organised as follows. Section 2 outlines the conceptual model of multi-agent systems (MAS) and argues that MAS is suitable for WS. Section 3 discusses the modelling of WS applications in CAMLE and illustrates our approach with an example of online auction services. Section 4 presents the formal specifications in SLABS automatically generated by CAMLE tool. Section 5 concludes the paper with a summary of the proposed method and a discussion on the directions for future research.

## 2. Conceptual model of WS

In this section, we argue that WS can be regarded as MAS and outline a conceptual model for WS.

### 2.1. The notion of agents

Agent is the most important but controversial notion in agent-based computing. It is often characterised by certain properties, see e.g. [12, 13]. The following have been widely considered as the most important ones.
*(a) Autonomy*: the capability of performing actions without explicit commands and having control over their state as well as their behaviour [12, 14, 15, 16].
*(b) Pro-activity*: the capability of exhibiting opportunistic and goal-directed behaviour and taking initiative.
*(c) Responsiveness*: the capability of perceiving the environment and responding in a timely fashion.
*(d) Sociality*: the capability of interacting with other agents and humans to complete their own tasks and to help others.

These properties match the features of software systems that run on the servers linked to the Internet and constitute a WS application. Service providers, requesters and registries perform their tasks autonomously in the sense that none of them should be considered as commanding the others. For example, a provider can refuse a service request. A requester can also stop further participation in the service process if the service provider does not satisfy the requester's business criteria. Each side has no control over the other. The interactions between two components of WS are essentially collaborations. A service requester may initiate the interaction with a service request. However, a service provider by no means has to be passive during the whole process of service. It may also take initiative actions from time to time. Therefore, at this very abstract level, agent technology is suitable for WS applications.

However, not all agent models developed in AI researches are suitable for WS. For example, in the BDI models, agents have mental states consisting of belief, desire and intension that control their behaviours [17, 18]. Game theory models define agents as computational entities that aim at maximising their utility functions. It is questionable if ordinary programmers can produce WS systems productively through thinking of belief, desire and intention, or games and utility functions. Moreover, WS has been considered as an attractive technology for wrapping existing IT assets so that new solutions can be deployed quickly and recomposed to address new opportunities [1]. Few of existing IT assets can be considered as agents in these models.

### 2.2. Meta-model of agents and MAS

Our agent model is from a software engineering perspective [19, 20, 21].

We define *agents* as active and persistent computational entities that encapsulate data, operations and behaviours and situate in their designated environments. Here, data represents an agent's state. Operations are the actions that an agent can take to modify its state and/or to affect the environment. Behaviours are sequences of state changes and operations performed by the agent in the context of its environment. By encapsulation, we mean that an agent's state can only be changed by the agent itself, and an agent has its own rules that govern its behaviour. Each agent must also have an explicit specification of its designated environment. Therefore, agents have a structure containing the following elements.

- *Agent name*. It is the identity of the agent.
- *Environment description*. It specifies a set of agents that influence the agent.
- *State space*. It defines the states that the agent can be in. It is divided into two parts. The *visible part* consists of a set of variables whose values are visible but cannot be changed by other computational entities. *The internal part* consists of a set of variables which are not visible by other entities.
- *Actions*. They are the atomic actions that the agent can take. Each action has a name and may have parameters. An action can be either visible or internal. *Visible actions* generate events visible by other agents, while *internal actions* are not visible to any other agent.
- *Behaviour rules*. It is the agent's body that determines its behaviour and has the following structure.

```
Begin
    Initialisation of internal state;
    Loop
        Perception of the situation in its environment;
        Decision on the action to take, which can be
            (1) visible or internal actions;
            (2) changes of visible or internal state;
            (3) joining into or retreating from a caste;
    end of loop;
end
```

In the context of WS, the components in a WS application can be modelled by a number of agents. For example, a WS provider can be considered as an agent, whose services as the visible actions. The information that a WS publishes on the Internet can be considered as visible state, while an invisible state represents the internal state of the software system. The behaviour rules determine the way that the WS fulfils its tasks.

A MAS consists of a set of interactive agents that are grouped into *castes*. Caste is a new concept first introduced by SLABS. It is a natural evolution of the concepts of classes in object-orientation and data types in procedural programming. It can play a significant role in agent-oriented software development [22]. The notion of caste is defined as a set of agents with the same structural and behavioural characteristics. Agents are instances of castes. It has the structure and behaviour characteristics defined by the caste. An example of behaviour characteristics is that an agent follows a specific communication protocol to communicate with other agents. Therefore, such a communication protocol can be specified by defining a caste with the protocol as behaviour characteristic. For example, we can define the caste *WS Agent* as those using TCP/IP protocols with messages encoded compliant with SOAP. The relationship between agents and castes is similar to what is between objects and classes. What is different is that an agent can join a caste or retreat from a caste at run-time dynamically. In modelling language CAMLE, how agents change their casteship is described by migration relations.

Inheritance relationships can also be defined between castes. A sub-caste inherits the structure and behaviour from its super-castes. But, a sub-caste cannot overwrite the structures and behaviour rules of its super-castes. Multiple inheritances are allowed to enable an agent to belong to more than one society and play more than one role at the same time.

Our model of agents also allows agents to be formed from a group of other agents. The former are called compound agents and the latter component agents. In such a case, a whole-part relationship exists between the compound and the component agents, which is represented as an aggregate relation between castes in CAMLE. In the design of CAMLE language, we identified three types of commonly used whole-part relationships between agents according to the ways a component agent is bound to the compound agent and the ways a compound agent controls its components. The strongest binding is *composition* in which the compound agent is responsible for creation and destruction of its components. If the compound agent is destroyed, the components no longer exist. The weakest binding is *aggregation*, in which the lifetimes of the compound and the component are independent, so that the component agent will not be affected at all when the compound agent is destroyed. Between these two is the *congregation* whole-part relation. With such a relation, when the compound agent is destroyed, the component agents will still exist, but they will lose the membership to the component caste. This is a novel type of whole-part relationship that has not been investigated in the literature so far to our knowledge.

In the context of WS, service providers and service requesters are grouped into castes. Different castes represent different types of service requesters and different types of service providers. An agent can join a caste to become a valid requester and quit from the caste after receiving the services or when it is unsatisfied with the services. When it is a member of the caste, it must obey the behaviour rules in order to obtain the required services. However, it has no obligations to follow the rules after quitting from the caste. The organisational structure of a MAS is depicted in a caste model in CAMLE. It describes the castes and their inheritance, whole-part and migration relations. Figure 1 shows the architecture of WS in a caste model. It states that a WS application may consist of a number of WS providers, WS requesters and a set of business agents that implement business rules and processes. A business agent can participate in service provider and/or service requester castes. The providers and requesters must be WS agents that comply with SOAP protocol.

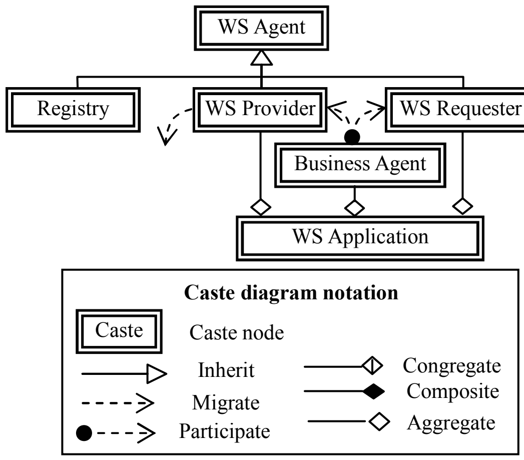


**Figure 1. Caste diagram of WS architecture**

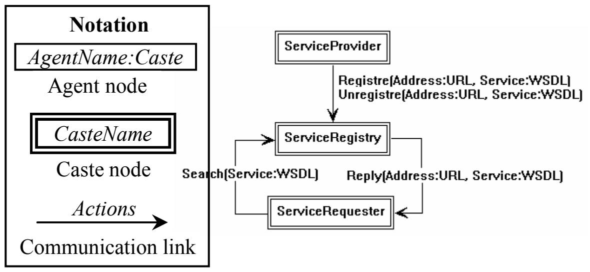


**Figure 2. Collaboration model of WS architecture**

Communication plays a crucial role in MAS as well as in WS. Components of a WS application must communicate to collaborate with each other. There are two means of communication in our meta-model: visible actions and visible states. Communication by visible actions is similar to sending a message through the Internet (or broadcasting a message on a network), which requires the sender to take an action (i.e. to send the message) and the receiver(s) to observe the action (i.e. to catch the message). Communication by visible states matches the way of communication by publishing information on the web. These are the basic modes of communications through the Internet. Our meta-model does not give details about the communication protocols, syntactic formats and their semantics. Such details are important in the development of WS. The modelling language provides software engineers with collaboration diagrams to specify such details about communications. It enables engineers to work at a very high level of abstraction and to focus on the functionality and behaviour aspects rather than on syntactic details.

The power of agent-based systems has been best demonstrated in dynamic environment [23, 24], which is also a basic property of Internet-based computing. Usually, the environment of an agent consists of a set of different types of entities, such as software objects, equipments, devices, human beings, and software systems, etc. As argued in [10, 25], all of these types of entities can be considered as agents as defined above. Therefore, in our agent model, the environment of an agent contains a subset of the agents that may affect the behaviour of the agent. Moreover, we emphasizes that agents are situated in their *designated* environments, which is specified as the set of agents in a caste, or a specific agent in a caste, or a parameter the represent an agent in a caste, or a combination of the above. It differs from a completely *open environment*, where every element in the system can always affect the behaviour of an agent. It also differs from a *fixed environment*, where an agent can only be affected by a fix set of entities in the system. In either fixed or open environments, the agent cannot change its environment. The concept of designated environment gives software developers more power of control over the environment so that software agents have more protection in dynamic environments. It is worth noting that both open and fixed environments are special cases of the designated environments.

In our models of MAS, the behaviours of agents are defined in terms of agents' responses to environment scenarios. A scenario represents the observation of an agent towards its environment at a particular time.

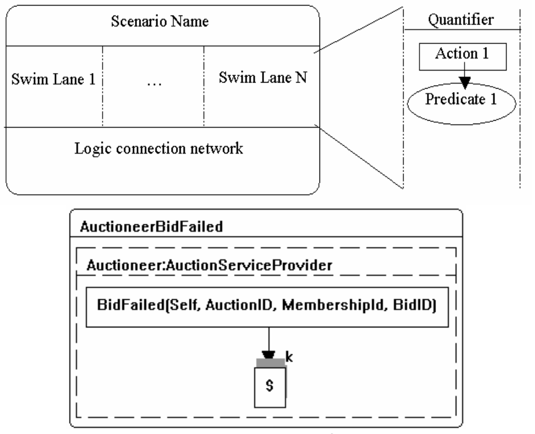


Figure 3. Format and example of scenario diagram

Figure 3 gives the format and an example of scenario diagrams, which depicts the situation that an auctioneer informs the agent that its bid failed. Behaviour diagrams describe agents' designed behaviour in certain scenarios. Figure 4 shows an example of behaviour diagram. It specifies a simple behaviour rule of the Service Registry that when there is a WS requester that sends a Search request to the registry with description of services C, the registry will reply with a list of services that matches the description.

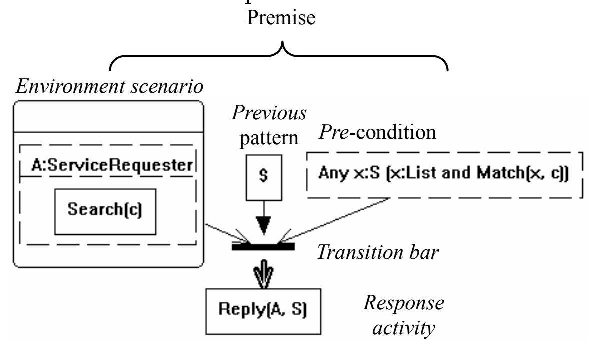


Figure 4. Example behaviour rule of service registry

Readers are referred to [10, 11, 26, 27] for more details of the meta-model and CAMLE language.

## 3. Modelling WS Application Systems

In this section we discuss how WS application systems can be modelled in CAMLE. We will illustrate our method with an example of online auction services. We will demonstrate how developers of service provider model the service provider system without over-restricting the development of the users of the services, and how the design knowledge is specified in the model for the developers of the requesters.

### 3.1. Service provider's perspective

The first step is to identify the types of agents that participate in the operation of the system and specify them as castes. From the online auction service provider's point of view, there are two types of agents that will interact with their software. Sellers can ask for the service provider to set up an online auction to sell its goods with certain conditions. Buyers can then bid for the goods online. Therefore, we have three different castes in this application: (a) Auction Service Providers, (b) Sellers, (c) Buyers. Sellers and buyers are service requesters; hence they are sub-castes of Service Requesters defined in the previous section. Auction service providers are service providers for sellers and buyers, hence, a sub-caste of Service Providers. This leads to the caste diagram from the auction service provider's perspective shown in Figure 5. Of course, the Auction Service Providers can be compound and more complicated than what is depicted in the diagram. However, its internal structure is hidden from the users of the system.

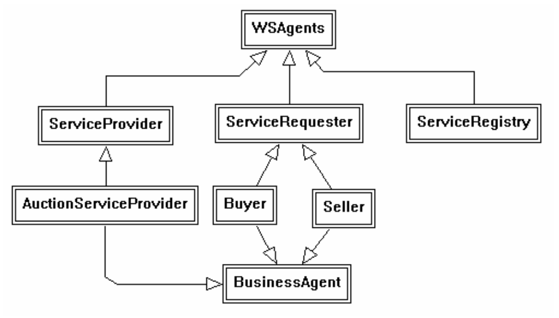


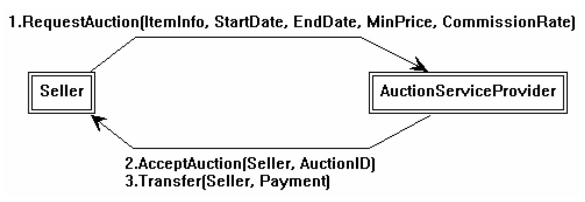
**Figure 5. Caste diagram from provider's perspective**

There are two types of services that an online auction provider provides to different types of service requesters. It sets up online auctions according to a seller's request. It also conducts online auctions via accepting bids from buyers. The communications with the sellers and buyers are depicted in the generic collaboration diagram in Figure 7.

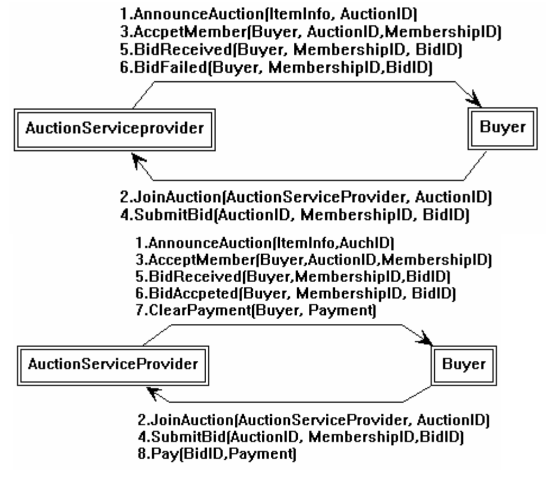
There are two scenarios in the collaboration between the service provider and its requesters. The first is to set an auction for a seller. The second is to run the online auction to sell the item to buyers. Each scenario is modelled by a scenario-specific collaboration diagram, as shown in Figure 6.

While collaboration diagrams are expressive enough to describe workflow in the form of action sequences, it is not capable of expressing the semantics of business rules. For example, in the interaction with buyers, an auction service provider must follow the following rules.

(1) When a buyer requests to join the auction, its credit must be checked and the membership issued if its credit is OK;
(2) When receives a bid from a member buyer, the auctioneer must acknowledge the receipt of the bid with a unique bid identifier;
(3) Every received bid must be compared with the current best bid. If the new bid beats the current best bid, the new bid becomes the current best bid; otherwise, a failure message is sent to the bidder;
(4) By the scheduled finish time of the auction, an acceptance message must be sent to the winner;
(5) Payment from the bid winner must be cleared and fund transferred to the seller with commission charged with the agreed commission rate.


1.RequestAuction(ItemInfo, StartDate, EndDate, MinPrice, CommissionRate)
Seller
AuctionServiceProvider
2.AcceptAuction(Seller, AuctionID)
3.Transfer(Seller, Payment)

(a) Scenario of setting up an auction for a seller


1.AnnounceAuction(ItemInfo, AuctionID)
3.AccpetMember(Buyer, AuctionID,MembershipID)
5.BidReceived(Buyer, MembershipID, BidID)
6.BidFailed(Buyer, MembershipID,BidID)
AuctionServiceprovider
Buyer
2.JoinAuction(AuctionServiceProvider, AuctionID)
4.SubmitBid(AuctionID, MembershipID, BidID)
1.AnnounceAuction(ItemInfo,AuchID)
3.AcceptMember(Buyer,AuctionID,MembershipID)
5.BidReceived(Buyer,MembershipID,BidID)
6.BidAccpeted(Buyer, MembershipID, BidID)
7.ClearPayment(Buyer, Payment)
AuctionServiceProvider
Buyer
2.JoinAuction(AuctionServiceProvider, AuctionID)
4.SubmitBid(AuctionID, MembershipID,BidID)
8.Pay(BidID,Payment)

(b) Scenarios of running an auction

**Figure 6. Scenario-specific collaboration diagrams**

The protocol is specified by a behaviour diagram for the Auction Service Providers caste. A behaviour diagram contains the behaviour rules for the agents. For example, the rule (1) above is specified in Figure 88.

In the development of a service, certain assumptions on the service requesters' behaviour must be made. For example, in the interactions with the auction service provider, the buyers must follow an interaction protocol.

(1) A buyer must join an auction before the scheduled start date of the auction and become a member of the auction before it submits any bid;
(2) A buyer's bid for an item must be better than the current best bid for the item;
(3) By the scheduled finish time of the auction, only the best bid is accepted and its buyer must buy the item;
(4) If a buyer's bid is beaten by another bid, the beaten bid is failed;
(5) A buyer can quit from the auction only after its bid becomes failed.

The complete protocol is expressed as two sets of rules; one for the auctioneer and one for the buyers. Thus, a behaviour diagram is also associated to the Buyer caste as a part of the model from the provider's perspective. Similarly, there is a set of behaviour rules for the Seller caste. The details are omitted for the sake of space.

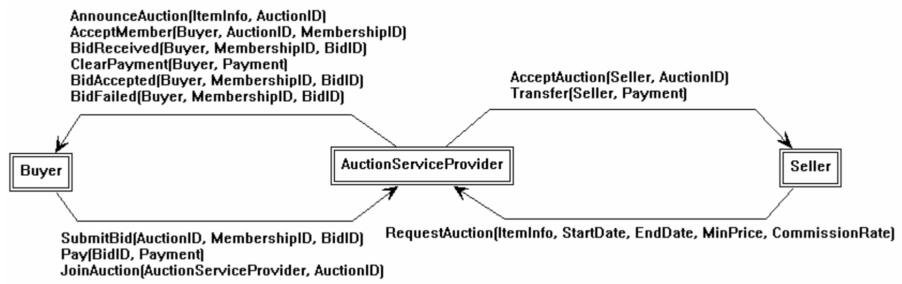

AnnounceAuction(ItemInfo, AuctionID)
AcceptMember(Buyer, AuctionID, MembershipID)
BidReceived(Buyer, MembershipID, BidID)
ClearPayment(Buyer, Payment)
BidAccepted(Buyer, MembershipID, BidID)
BidFailed(Buyer, MembershipID, BidID)
AcceptAuction(Seller, AuctionID)
Transfer(Seller, Payment)
Buyer
AuctionServiceProvider
Seller
SubmitBid(AuctionID, MembershipID, BidID)
Pay(BidID, Payment)
JoinAuction(AuctionServiceProvider, AuctionID)
RequestAuction(ItemInfo, StartDate, EndDate, MinPrice, CommissionRate)

**Figure 7. Generic collaboration diagram from auction service provider's perspective**

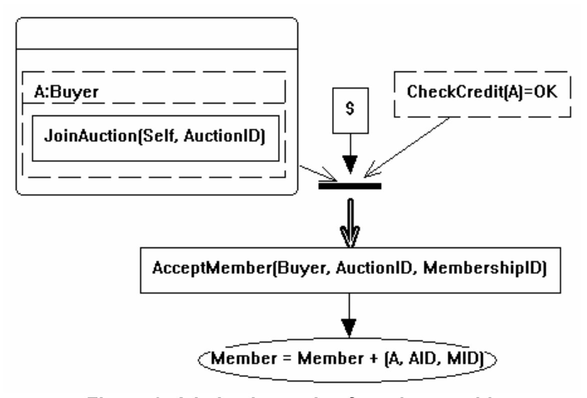

A:Buyer
JoinAuction(Self, AuctionID)
$
CheckCredit(A)=OK
AcceptMember(Buyer, AuctionID, MembershipID)
Member = Member + (A, AID, MID)

**Figure 8. A behaviour rule of service provider**

### 3.2. Service requester's perspective

We now discuss how CAMLE can be used to develop models from the requester's perspective.

Consider an online flight ticketing service that sells air tickets via an e-commerce website. For each flight, it will try to sell the unsold tickets by online auction when the time reaches 7 days before the scheduled date of flight. Suppose the normal business rules and process of the software is specified as a caste Ticket Seller. For the sack of space, the detail of the caste is omitted in this paper. When a ticket seller wants to sell air tickets by auction, it will become a member of the Seller caste and obey the behaviour rules specified in the service provider's model. Such agents, which are called Sell By Auction, must satisfy all the structure and behaviour requirements specified in the castes Ticket Sellers and Sellers. They can be specified as the sub-caste of Ticket Sellers and Seller to model their behaviours. As shown in Figure 9, an alternative caste model that enables a Ticket Seller to use the auction WS is to have a participation relation from Ticket Seller to Seller. In this case, an agent of the caste Ticket Seller joins the Seller caste dynamically and uses the auction service after joining the caste. Such dynamic integration is what WS meant to be. Hence, it is a better model than the static model.

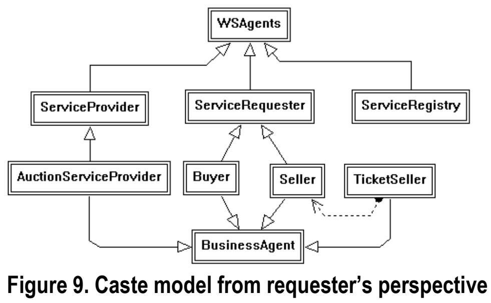


**Figure 9. Caste model from requester's perspective**

For the castes shown in Figure 9, the behaviour diagrams for caste WS Agents, Service Provider, Service Requester and Service Registry are common to all WS applications. Hence, they should be treated as the public information and ideally as a part of the WS standard. The models and specification of caste Auction Service Provider, Buyer and Seller are provided by the WS providers. They define the syntax, semantics as well as the pragmatics of the auction WS. They are also public and should be stored with the WS registration.

## 4. Generation of Formal Specifications

The modelling environment developed for CAMLE contains a number of automated tools, which include tools for consistency checking [28] and generation of formal specifications in SLABS. In SLABS, each caste is specified in the following syntax, which can also be in the equivalent graphic format shown in Figure 10.

```
Caste-description ::=
    Caste name [ <= { caste-name [ ( instantiation ) ] , }+ ; ]
        [ environment-description ; ]
        [ structure-description ; ] [ behavior-description ; ]
    end name
```

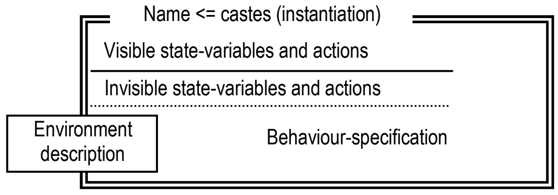

Name <= castes (instantiation)
Visible state-variables and actions
Invisible state-variables and actions
Environment description
Behaviour-specification

**Figure 10. Format of caste description in SLABS**

The SLABS language uses transition rules as a facility to explicitly specify how observations of the environment are related to the agent's behaviour. Each rule consists of a description of a scenario of the environment, the action to be taken by the agent in the scenario and a condition of the agent's internal state and previous behaviour.

```
Behaviour-rule ::= [ < rule-name >: ] pattern | [ prob ] –> event ,
                        [ if Scenario] [where pre-cond] ;
```

The formal specifications of castes Service Provider and Buyer given in Figure 11 are generated from CAMLE models. Similarly, the specifications of other castes can be generated. Details are omitted.

With the formal specification, we can formally prove the properties of a WS if it satisfies the specification. For example, the following are some examples of the properties that can be inferred from the specification of the auction service system.

- If a buyer submits a bid, the auctioneer will send an acknowledgement message 'BidRecieved'.
- Buyer only submits a bid that beats the current best bid.
- By the end of auction, the current bid must be the best bid, and that bid will be accepted by the auctioneer. In that case, an acceptance message 'BidAccepted' will be send to the buyer who submitted the bid.
- Once a buyer receives an acceptance message, it will pay for the item.
- Any bid submitted to the auction that failed must be beaten by at least one bid.

These properties are important for the auction service requesters. However, they are deeply related to the semantics of the service description that are inadequately specified in existing of WS descriptions.

```
Service Providers
VAR
     ServiceDescription: WSDL;
ACTION
     Register(R: Service Registries, service: WSDL);
     Unregister(R: Service Registries, service: WSDL);
----------------------------------------------------
VAR
     State: {Start Service, In Service, Stop Service}
- - - - - - - - - - - - - - - - - - - - - - - - - -
[!State=Start Service ] |→
        Register(R, ServiceDescription)!State'=In Service;
[!State=In Service, !State=Stop Service ] |→
        Unregister(R, ServiceDescription);
```

```
Buyers <= Service Requesters
VAR  BusinessInfo: UDDI;
ACTION     Submit_Bid(AuctionID, MembershipID, BID);
    Pay(BID_ID, PAYMENT);
    Join_Auction(Auction Service Providers, AuctionID);
----------------------------------------------------
VAR  Membership: {Yes, No};  MID: MembershipID;
       Auction: AuctionID;        Bid_ID: BID_ID;
- - - - - - - - - - - - - - - - - - - - - - - - - -
<Join Auction>:
           [!Membership= No ]
Auctioneer:  |→ time: Join_Auction(Auctioneer, AID);
 Auction            if Auctioneer:[Announce_Auction(d, AID)];
 Service            where Auct∈ Auctioner.AuctionInfo
 Provider           & time < Auct.Start & Auct.ID=AID
<Get Membership ID>:
      [Join_Auction(Auctioneer, AID)]
        |→ !Membership'=Yes & Auction'=AID, MID'=mid
            if Auctioneer:[Accept_Member(Self, AID, mid)
<Submit Bid>:
      [!Membership=Yes] |→ Submit_Bid(Auction, MID, Bid);
            where Beat(Bid, Auctioneer.auct.Current_Bid)
            & Auct∈ Auctioneer.AuctionInfo
            & Auction.Auct.ID=Auction
<Receive Acknowledge Of Bid>:
      [Submit_Bid(Auction, MID, Bid)] |→!Bid_ID'=bidID;
        if Auctioneer:[ Bid_Received (Self, AID, mid, bidID)],
        where AID=Auction & mid = MID;
<Revise Bid After Failure>:
      [Submit_Bid(Auction, MID, Bid)]
        |→; Submit_Bid(Auction, MID, Bid2)
          If Auctioneer:[Bid_Failed(Self, AID, mid, bidID), $^k],
          where Auct ∈ Auctioneer.AuctionInfo
          & Auct.ID=Auction
          & Beat(Bid, Auct.Current_Bid)
          & Bid_ID = bidID & MID=mid;
<Pay Accepted Bid>:
      [Submit_Bid(Auction, MID, Bid)]
        |→; Pay(Bid_ID, Payment)
          If Auctioneer:[ Bid_Accepted (Self, AID, mid, bidID)],
          Where AID=Auction & Bid_ID=bidID & MID = mid
<Quit From Auction>:
   [!Membership=Yes]
        |→ Quit_Auction(AuctionID)!Membership'=No,
             if Auctioneer:[Bid_Failed(Self, AID, bidID), $^k]
             where Auction=AID & Bid_ID = bidID
```

**Figure 11. Specifications generated from models**

## 5. Concluding remarks

In this paper, we argued that WS applications should be understood as multi-agent systems. We proposed an approach to use the graphic agent-oriented modelling language CAMLE to model WS applications. It is illustrated by an example of online auction service to demonstrate how models of WS in CAMLE can help developers from both service provider and service requester's perspectives. Another advantage of modelling WS in CAMLE is that formal specifications can be automatically generated so that properties of a WS can be formally proved.

The structure of modelling and formal specification of WS proposed in this paper provides a modular description of the semantics of the services provided. It also enables explicit specifications of the service provider's assumptions on the service requester's behaviour. Hence, the designated environment of the service provider can be clearly stated for developers on both sides. The same specification can also be used by developers of service requesters so that the application can be smoothly integrated without too much demand of technique supports from the service provider.

There are a number of issues worthy further research. We are investigating how formal specifications of WS can be represented in a format that complies with XML standard and can be used to describe WS and facilitate the dynamic search and integration of WS applications. We are developing a method and formal logic that enables formal specifications to be used in service search and query. Formal specifications should also be helpful in quality assurance in the development of WS applications through validation, verification and testing. We are working on automated testing of WS applications based on formal specifications. We have already developed an experimental programming language based on our meta-model of MAS for implementing agent-oriented software systems. Further research is in progress to write WS in such a programming language. How to derive implementations from formal specifications is also an interesting topic for further research.

## Acknowledgement

The work reported in this paper is supported by China High-Technology Research and Development Programme under the grant 2002AA116070.

## References

[1] K. Gottschalk, *et al.*, "Introduction to Web services architecture", *IBM Systems Journal*, 41(2), 2002, pp. 170-177.

[2] C. Lau, and A. Ryman, "Developing XML Web services with WebSphere Studio Application Developer", *IBM Systems Journal*, 41(2), 2002, pp. 178-197.

[3] M. Stal, “Web Services: Beyond Component-Based Computting”, *C. ACM*, 45(10), 2002, pp. 71-76.

[4] M. Huhns, and M. P. Singh (Eds.), *Readings in Agents,* Morgan Kaufmann, San Francisco, 1997.

[5] F. Leymann, D. Roller, and M.-T. Schmidt, “Web services and business process management”, *IBM Systems Journal*, 41(2), 2002, pp. 198-211.

[6] P. Lambros, M.-T. Schmidt, and C. Zentner, *Combine Business Process Management Technology and Business Services to Implement Complex Web Services*, IBM Corporation, 2001.

[7] F. Leymann, *Web Services Flow Language*, IBM Corporation, 2001.

[8] S. Thatte, *XLANG-Web Services for Business Process Design*, Microsoft Corporation, 2001.

[9] H. Zhu, B. Zhou, X. Mao, L. Shan, and D. Duce, “Agent-Oriented Formal Specification of Web Services”, *Proc. of the AAC-GEVO’04 Workshop at GCC’04*, Springer, Oct. 2004.

[10] H. Zhu, “SLABS: A Formal Specification Language for Agent-Based Systems”, *Int. J. of Software Engineering and Knowledge Engineering*, 11(5), 2001, pp. 529-558.

[11] L. Shan, and H. Zhu, “CAMLE: A Caste-Centric Agent-Oriented Modelling Language and Environment”, *Proc. of SELMAS'04 at ICSE'94*, Edinburgh, UK., 2004, IEE, pp. 66-73.

[12] N.R. Jennings, “On agent-based software engineering”, *Artificial Intelligence*, 117, 2000, pp. 277-296.

[13] D.B. Lange, “Mobile Objects and mobile agents: The future of distributed computing?”, *Proceedings of The European Conference on Object-Oriented Programming*, 1998.

[14] N.R. Jennings, “Agent-Oriented Software Engineering”, *Multi-Agent System Engineering, Proceedings of 9th European Workshop on Modelling Autonomous Agents in a Multi-Agent World,* Valencia, Spain, 1999, Springer, Berlin, Heidelberg, New York, pp.1-7.

[15] B. Bauer, J.P., Muller, and J. Odell, “Agent UML: a formalism for specifying multiagent software systems”, *Agent-Oriented Software Engineering*, M. Wooldridge, (Eds), Springer, 2001, pp. 91-103.

[16] J. Odell, H. Van Dyke Parunak, and B. Bauer, “Representing Agent interaction protocols in UML”, *Agent-Oriented Software Engineering*, M. Wooldridge (Eds), Springer, 2001, pp. 121-140.

[17] A.S. Rao, and M.P. Georgreff, “Modeling Rational Agents within A BDI-Architecture”, *Proc. of the International Conference on Principles of Knowledge Representation and Reasoning*, 1991, pp. 473-484.

[18] M. Wooldrighe, *Reasoning About Rational Agents*, The MIT Press, 2000.

[19] P. Ciancarini, and M. Wooldridge, *Agent-Oriented Software Engineering*, LNCS 1957, Springer-Verlag, 2000.

[20] M. Wooldridge, G. Weiss, and P. Ciancarini, *Agent-Oriented Software Engineering II*, LNCS 2222, Springer, 2002.

[21] L. Shan, and H. Zhu, “CAMLE: A Caste-Centric Agent-Oriented Modelling Language and Environment”, *Proc. of SELMAS’04 at ICSE’94*, Edinburgh, UK, IEE, May 2004, pp. 66-73.

[22] H. Zhu, “The role of caste in formal specification of MAS”, *Proc. of PRIMA’2001*, Taipei, Taiwan, Springer, 2001, pp.1-15.

[23] N.R. Jennings, and M.J. Wooldridge, *Agent Technology: Foundations, Applications, And Markets*, Springer, Berlin, Heidelberg, New York, 1998.

[24] M. Huhns, and M.P. Singh, *Readings in Agents*, Morgan Kaufmann, San Francisco, 1997.

[25] H. Zhu, “Formal Specification of Agent Behaviour through Environment Scenarios”, *Formal Aspects of Agent-Based Systems*, Rash, J., *et al*. (Eds), Springer, 2001, pp. 263-277.

[26] L. Shan, and H. Zhu, “Modelling and specification of scenarios and agent behaviour”, *IEEE/WIC conference on Intelligent Agent Technology (IAT’03)*, Halifax, Canada, Oct. 2003.

[27] H. Zhu, *A Formal Specification Language for Agent-Oriented Software Engineering*, Department of Computing, Oxford Brookes University, 2002.

[28] L. Shan, and H. Zhu, “Consistency Check in Modeling Multi-Agent Systems”, *Proc. of COMPSAC’04*, IEEE CS, Sept., 2004, pp. 114-121.