Modelling Web Services in the Agent-Oriented Modelling Language and Environment CAMLE

Hong Zhu*
Department of Computing
Oxford Brookes University, Oxford OX33 1HX, UK
Email: hzhu@brookes.ac.uk
*Corresponding author

Lijun Shan
Department of Computer Science
National University of Defence Technology, Changsha, 410073, P.R. China
Email: lijunshancn@yahoo.com

Abstract. Web services (WS) provide a technology for integrating applications over the Internet, which may be performed during execution through dynamic service discovery and invocation. A particular difficulty in the development of WS applications is caused by the lack of communications between developers from different vendors. This paper investigates how modelling can help solve the problem in the context of model-driven software development using the caste-centric agent-oriented modelling language and environment CAMLE. One of the main features of the method is the separation of perspectives so that models are built separately by different groups of software developers for service providers and requesters with different perspectives. Each model only represents the software system in one particular perspective, but it is self-contained so that it can be checked for consistency, transformed into formal specifications, and further used to derive implementations, test its validity and prove its properties, etc. Connections between the models are realised through common castes. The paper illustrates the method by an example of online auction service. The use of the modelling language and environment CAMLE in model consistency check and specification generation is also discussed. It is shown that semi-formal 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' behaviours. Such semantic information is crucial for the success of dynamic integration of WS.

Keywords: Web Services, Service-oriented computing, Software development methodology, Agent-oriented software engineering, Model-driven software development, Modelling, Modelling languages and tools, Generation of formal specifications, Consistency check.

Biographical notes: H. Zhu received his BSc, MSc and PhD degrees in Computer Science from Nanjing University, P.R. China, in 1982, 1984 and 1987, respectively. He is currently a professor of computer science at the Department of Computing, Oxford Brookes University, England. His current research interests include agent-oriented software development methodology, Web-based applications, software testing and quality assurance, etc.

L. Shan received her BSc and MSc degrees in Computer Science from the National University of Defence Technology, P.R. China, in 2001 and 2004, respectively. She is currently a PhD candidate at the Department of Computer Science at the National University of Defence Technology, P.R. China. Her current research focuses on software development methodology for Web Services.

1 INTRODUCTION

Web services (WS) is characterised by the dominant of program-to-program interactions (Gottschalk et al., 2002). In view of the infrastructure of WS becoming pervasive, a new paradigm of service-oriented computing is emerging. It is widely recognised that WS technologies will profoundly change the ways that computer systems and software are developed and used. This paper is concerned with the methodology for the development of WS applications with focus on the modelling and specifications of such systems.

1.1 Motivation

In comparison with other distributed computing techniques such as CORBA, Java RMI and DCOM, WS technology offers more flexibility and looser coupling so that it is more suitable for internet computing (Lau and Ryman, 2002). However, as Stal (2002) pointed out, it is fundamentally different from the others. The components of WS applications, such as service providers, are autonomous, active and persistent computational entities that control their own resources and their own behaviours. They have social ability and collaborate with each other through dynamic discovery and invocation of services. They cannot be simply considered as objects. Instead, entities with these features have been studied in AI community as agents; c.f. (Huhns and Singh, 1997). However, software engineering for developing such systems has been only a recent research topic with the recognition of the inadequacy of object-oriented software development methodology (Jennings, 1999, 2000). A particular difficult issue in the development of WS applications is the problem of trustworthiness. Because a service provider and its service requester software systems are often developed, operated and maintained by different vendors, the service requesters have to reply on the service provider to ensure the correct behaviour of the application. The software systems that provide the services are commonly open to the public. The correct execution of a service provider's software system depends on the requesters' software to behave as expected. How to ensure both service provider's and requester's software systems behave as expected, how to detect, deny and recover from unexpected behaviours are the key issues for the successful collaborations between them. The foundation to solve the trustworthy problem is the specification of what are expected form both sides of service providers and service requesters. This is the theme of the paper.

The existing software engineering methodologies are inadequate to address the problems due to the following features of WS. First, the components in a WS application are usually developed by different vendors. Developers of the service providers and those of the service requesters are usually separated geographically and temporally. There is typically a lack of communication between them. The most effective channel of communications between them is perhaps through documentation. Second, WS technology enables dynamic software integration at runtime. It does not only require the interfaces between integrated entities syntactically compatible, but more importantly, the interactions must be semantically correct. For example, the meaning of a message passed between a service provider and a service requester must be interpreted exactly the same. The effects of an action taken by either the provider or a requester must be exactly the same as both sides expected. The order of the events happened in the interactions between a service provider and a requester (or requesters) must also be the same as all parties expected. To enable dynamic search of services, informal documentation of services is insufficient because currently natural language processing techniques are not mature enough to understand informally presented documents that specify a software system. 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 is of significant importance for the success of WS technology (Lambros, 2001; Leymann *et al.* 2002). Even for manual composition of WS applications, informal documentation suffers from the weakness of ambiguity, incompleteness and inconsistency. This weakness could be a major problem in the development of trustworthy WS applications even for composition by human developers. From the perspective of a service provider, a clear, unambiguous and consistent specification of what the service provides and what are expected of the service requesters' behaviours is the foundation to ensure the correctness of the provided services. From the perspectives of a service requester, in addition to the functional and non-functional requirements of the requester application, a well-written specification of the services that it uses and what are expected in its interactions with the service providers is the foundation of their successful uses of the services. Moreover, the specifications from these two perspectives must be consistent to enable correct interactions at runtime, and must also be flexible enough to give the other side the freedom in implementation. Formal documentation can facilitate not only more rigorous development of WS applications, but also facilitate more systematic and automatic validation and verification of the systems. Hence, it is helpful to address the trustworthy problem of WS.

1.2 Related work

There have been several efforts of defining languages and/or standards to enable software to use WS without much explicit a priori knowledge on how to use them. Proposals to the description of semantic aspects of WS have been advanced in the literature that 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. Leymann at IBM (2001) has published WSFL based on Petri Net theory, which can be used to aggregate Web Services. As a counterpart, Microsoft rejuvenated the Pi-Calculus model with its XLANG (Thatte, 2001). These two approaches are unified in BPML 1.0 (BPML.org, 2004). Later on, BEA, IBM, and Microsoft published BPEL4WS. Other organizations advocated radically different approaches for business process modelling, such as OWL-S (DAML.org, 2004), and its predecessor DAML-S (DAML.org, 2001).

WSFL, XLANG, BPML and BPEL4WS can be categorized as workflow description standards that aim at automating the execution of multiple interrelated WS that can be aggregated to form a business process. WSFL consist of two types of models: the flow model and the global model. A flow model describes how to use the functionality provided by a collection of composed WS. A global model describes how the composed WS interact. These models enable the separation of the abstract definitions of a workflow process (the flow model) from the implementation details of the process (the global model). XLANG is very similar to WSFL in its functionality. It has become a part of the Microsoft BizTalk infrastructure. BPML and BPEL4WS share the same roots in SOAP, WSDL and UDDI. They take the same advantage of XML technologies, especially XPath and XSDL. They are designed to leverage other specifications such as WS-Security and WS-Transactions. Beyond these areas of commonality, BPML also provides support for advanced semantics such as nested processes and complex compensated transactions, which are capabilities that BPEL4WS has yet to address. OWL-S provides a machine-interpretable, ontology-backed semantic description of both atomic and composite WS. As described above, WSFL and XLANG, etc. are designed to define the flow of a composition of services. Similarly, OWL-S has the expressive power to encapsulate the composition of several services within a single service description. In OWL-S, a composite service can be recursively decomposed into a set of atomic services. Control constructs, such as *Sequence*, *Concurrent*, *Split+Join*, *If-Then-Else*, are provided to orchestrate the services that compose a workflow. However, having not achieved the full power of formal specification techniques, ontology and workflow descriptions are not expressive enough to provide all the required semantic information for dynamic discovery and invocation of WS. Second, their representations are not readable and precise enough to be used as the vehicle to bridge the gap between the service providers and requesters. In particular, they do not support modularity in the definition of the semantic aspects of WS in the way that is required by the features of WS applications especially when the developers from different vendors are separated geographically and temporally. Moreover, as Tsai *et al.* pointed out (2004), existing WS technologies have not taken into consideration of the real-time and dependability requirements. A framework of service-oriented dynamic reconfiguration was proposed by Tsai *et al.* (2004). It explored the feasibility of developing a new service specification technique ISC (Interface, Scenarios and Constraints) that extends WSDL to specify the static and dynamic structure of services. Based on ISC, a framework for dynamic service reconfiguration (DRS) was proposed to enable service registration, de-registration, look up, verification, binding, execution, monitoring at runtime, especially the re-selection and rebinding of services in case of failures or overload.

There is only a little effort that has directly gone into the research on methodologies for developing WS applications. Among the most closely related works are agent-oriented software development methodologies; see e.g. (Dam and Winikoff, 2003; Zambonelli and Omicini, 2004; Ciancarini and Wooldridge, 2001; Weiß and Ciancarini, 2002; Giunchiglia, Odell and Weiß, 2003; Giorgini, Muller and Odell, 2004; Odell, Giorgini and Muller, 2005; Garcia *et al.*, 2003; Lucena, *et al.*, 2004; Choren, *et al.*, 2005). Existing agent-oriented methodologies vary in how to describe agents and MAS at a higher abstraction level as well as how to obtain such a description. For example, Gaia (Zambonelli, *et al.*, 2003) provides software engineers with the organization-oriented abstraction in which software systems are conceived as organized societies and agents are seen as role players. Although Gaia was regarded as probably the most mature agent-oriented software development method, it has no modelling language and no modelling tools. Tropos (Bresciani, 2002) emphasizes the use of notions related to mental states during all software development phases. The notions like belief, desire, intention, plan, goals, etc., represent the abstraction of agent's state and capability. It is less directly applicable to WS although they may have potential to be adapted. A number of proposals of extending UML for the development of agent-based systems have also been advanced. Among them the most notable one is the work by FIPA's Agent UML Technical Committee and known as AUML (http://www.auml.org). It extends UML with notations to represent agents (Bauer et al., 2001; Odell et al., 2001). However, the semantics of the notation is left open. There is no well-defined meta-model of the modelling language. How to use the notation in software development is also an open problem. Tsai *et al.* (2005) proposed the WebStrar framework for developing trustworthy WS. The framework starts with the development of specification of WS in OWL-S. It applies completeness and consistency analysis, model checking and software testing techniques to ensure the quality of WS. Tsai *at el.* (2003, 2005b) have also been researching on the techniques of developing trustworthy WS including testing, simulation, and verification, etc.

1.3 Proposed approach

Addressing the problems discussed above, based on their caste-centric approach to agent-oriented software development methodology, Zhu *et al.* (2004) used an agent-oriented formal specification language SLABS (Zhu, 2001a, 2003) to formally specify the semantics of WS applications. Such a formal specification can be used as a solid foundation to implement WS in agent-oriented programming languages such as SLABSp (Wang, Shen and Zhu 2004, 2005a, 2005b). The correctness of the implementations as well as other properties of WS application systems such as emergent behaviours can be formally proved by reasoning about the specifications (Zhu, 2005). However, formal specifications are difficult to develop without tool support. This problem was addressed in (Zhu and Shan, 2005a), which proposed the use of the agent-oriented modelling language CAMLE and its automated tools (Shan and Zhu, 2004a, 2005; Zhu and Shan, 2005b) to develop formal specifications of WS. Graphic models are easier to construct and more readable than formal specifications. They are then automatically transformed into formal specifications in SLABS by using CAMLE's automated tools. This naturally leads to a model-driven development methodology for WS applications. In this paper, we further develop the approach by investigating the structure of agent-oriented models of WS applications and the uses of automated consistency check tools to ensure the quality of such models. Our approach has the following distinctive features.

A. Caste-centric agent-orientation

In (Zhu and Shan, 2005a), we studied the conceptual model of WS. At a high level of abstraction, the computational entities that constitute a WS application and provide or request services can be regarded as agents in our meta-model of multi-agent systems (MAS) (Zhu, 2001a). In our meta-model, the notion of agent is defined as the computational entities that encapsulate states, operations, and behaviour rules and situate in their designated environments; also see section 2. Intuitively, our definition of the word 'agent' has the same meaning as in the context of 'real estate agents' where agents provide services to clients for buying or selling properties, and 'travel agents' where agents provide clients with the services of purchasing transportation tickets and booking hotels, etc. In this sense, our agent-oriented methodology is actually a service-oriented methodology with emphasis on the entities that provide and use services rather than the functionality of these entities, i.e. the services. This enables us to model, analyse, specify, design, and implement WS at a very high level of abstraction rather than focusing on syntactic details of messages, etc. It is also worth noting that our meta-model is an extension of object-orientation with a uniformed semantics. Our model differs from existing ones in the way that caste plays the central role in the modular construction of software systems. Caste is the classifier of agents like class is the classifier of objects. It is the modular unit in a MAS. It can be used to implement various notions in MAS such as roles, agent societies, normative behaviours, interaction protocols, communication languages, etc. (Zhu, 2001b). As we will see later in this paper, caste also provides a nice language facility for modular construction of WS applications.

B. Model-driven process

In our method, modelling is the driving force of the development process shown in Figure 1. The construction of a model of the required system is the most important milestone of requirements elicitation and analysis, design and specification, etc. Models are the most important document at various stages of software development. It is also one of the most important software artefacts for various development activities. For example, models that represent the requirements can be transformed into design models and formal specifications (Shan and Zhu, 2004a, 2005) so that efficient implementations can be derived. Models can be automatically checked for its consistency (Shan and Zhu, 2004b), validated and tested against the users' requirements, and formally proved for its dynamic and static properties such as emergent properties (Zhu, 2005), etc. However, due to the limitation of space, this paper will focus on the construction of models for WS applications, checking models' consistency and the transformation of models into formal specifications. These development activities are effectively supported by the CAMLE language and automated tools. The implementation, testing, verification, validation, and maintenance issues will be addressed separately in other papers.

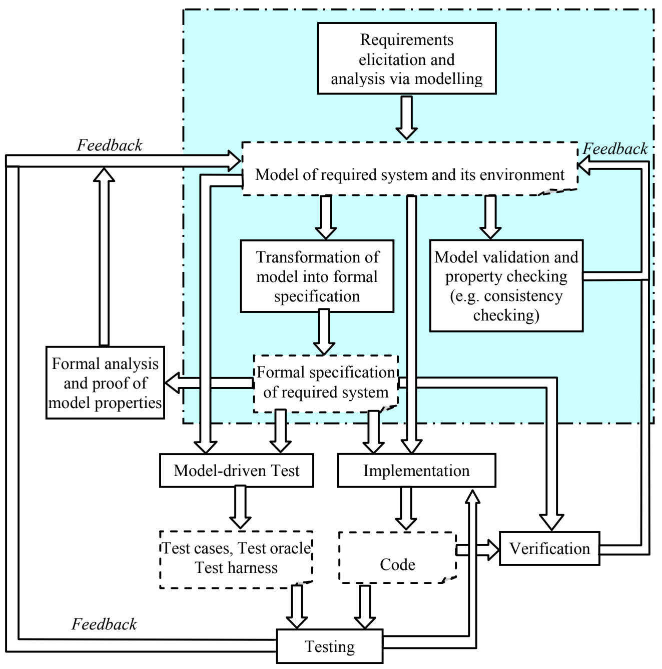

Requirements elicitation and analysis via modelling
Feedback
Model of required system and its environment
Feedback
Transformation of model into formal specification
Model validation and property checking (e.g. consistency checking)
Formal analysis and proof of model properties
Formal specification of required system
Model-driven Test
Implementation
Test cases, Test oracle Test harness
Code
Verification
Feedback
Testing

Figure 1 *Process of model-driven development of WS applications*

C. Separation of Perspectives

As discussed in subsection 1.1, the lack of communications is the heart of the problem in the development of WS applications. Our approach directly addresses the heart of the problem through *the separation of perspectives*. We recognise that the perspective from a service provider is different from those of service requesters. Even different types of service requesters may have different perspectives as well. Models thus developed from one perspective may not be complete in the traditional sense. Internal detailed information may need to be hidden from the users of different perspectives. Yet, sufficient information must be provided to the developers of other perspectives so that such partial models make sense and facilitate implementation, validation, verification and testing.

In order to achieve this objective, we propose the following two principles to organise the structure of models and specifications. As it will be shown in the paper, the modelling language and environment CAMLE supports such separation of perspectives.

- *The autonomy principle*

As discussed in section 1.1, software systems on both provider and requester sides are autonomous. They should be regarded as and implemented to be agents so that the interaction between them can be established dynamically as discussed above. Moreover, the developers of a WS provider and the developers of its service requesters may be from different vendors and separated by space and time. These groups of developers are also autonomous in the sense that they are not under the same management control structure. In other words, the developers of a WS provider are not always able to enforce how requester software is to be developed and how their service will be used. Similarly, developers of a service requester are not always able to enforce the developers of a service provider to change the service according to their requirements. The developers from different perspectives may have their own concerns and design decisions in their development processes. Models from both service provider and requester's perspectives, therefore, must allow flexibility for the other to make their own design decisions. The collaboration between these developers may have to be postponed to runtime and rely on the collaboration between the software systems. Our methodology respects the autonomy of the groups of WS developers as well as the autonomy of the software systems in WS.

- *The explicitness principle*

The assumptions made about the service requesters must be explicitly specified and published with the service itself. By '*the assumptions made by the service provider'* we meant, for example, what the service provider expects a requester to respond to the service provider's actions, the orders of the events in the interactions with the requesters, the interpretations of the meanings of the messages, etc. Some of such semantic correctness conditions can be formally specified by traces, invariant conditions and pre/post-conditions, and their combinations. Some may relate to the real world events, such as to switch on the intruder detector of a house, which is probably not as simple as to maintain an invariant of the program state or to satisfy a pre/post-condition of a procedure/method. In some cases, traditional formal specification techniques may not be powerful enough to specify such assumptions. For example, in English auction, a buyer must submit a bid with a price that is higher than all other buyers' bidding prices made so far. This condition cannot be specified straightforwardly as a pre/post-condition pair of the input/output of a 'bidding' method/procedure in the service requester's program or an invariant condition of the internal state of the requester's program, because it involves other buyers' behaviours, in particular, what have been done by all other buyers. What important is that these assumptions must be grouped and encapsulated into a modular unit so that developers from different perspectives only need to access the related parts. The structure of models and formal specifications of WS proposed in this paper and supported by CAMLE language and tools enables explicit specifications of the service provider's assumptions about the service requesters' behaviours in such a way. Hence, the environment of the service provider software can be clearly stated for software developers. The same specification can also be used by developers on the service requester side so that the application can be smoothly integrated without too much demand of technique supports from the service provider side.

There are two most important differences between our approach and the existing works on WS techniques. First, WSFL, BPML and OWL-S focus on the workflow management of multiple Web Services, where the basic elements are individual services and the relationships between them, e.g. the execution orders and transactional issues. CAMLE and SLABS can specify these issues as well as the semantic information of each single WS, for example, through the uses of patterns, scenarios, behaviour rules. Second, while the related works are on a more operational level that develops enabling technology for the declaration of the orchestration among multiple services, CAMLE and SLABS are on a more abstract level of software development methodology aiming at effective uses of such technology. The features of our approach in comparison with existing software development methods include agent-oriented development methodologies have been discussed above. In comparison with Tsai, *et al.*'s WebStrar framework, the work reported in this paper focuses on the development of models and formal specifications of WS applications, which form the foundation for other quality assurance activities, while WebStrar emphasises the testing, completeness and consistency analysis, verification and validation and other quality assurance activities.

1.4 Organisation of the paper

The paper is organised as follows. Section 2 briefly reviews the modelling language CAMLE and its modelling environment as well as the underlying meta-model of MAS. Section 3 is devoted to the construction of models of WS applications in CAMLE. The process of model construction is illustrated using the example of online auction services. Section 4 is concerned with the consistency between service providers and service requesters. It discusses the use of the modelling environment CAMLE in the consistency check of the models. Section 6 demonstrates the use of CAMLE's automated tools to generate formal specifications from graphic models. Section 6 concludes the paper with a summary of the proposed method and a discussion of the directions for future research.

2. MODELLING LANGUAGE AND ENVIRONMENT CAMLE

In this section, we briefly review the modelling language CAMLE, its modelling environment and the underlying meta-model in the context of WS. More details can be found in (Shan and Zhu, 2004a, 2005; Zhu and Shan, 2005b).

2.1 Modelling language CAMLE

Agent is the most important but controversial notion in agent-based computing. It is often characterised by certain properties, see e.g. (Jennings, 2000; Lange, 1998). The following properties have been widely considered as the most important ones.

- *Autonomy*: the capability of performing actions without explicit commands and having control over their state as well as their behaviour (Jennings, 1999, 2000; Bauer, Muller, and Odell, 2001; Odell, Parunak and Bauer, 2001).
- *Pro-activity*: the capability of exhibiting opportunistic and goal-directed behaviour and taking initiative.
- *Responsiveness*: the capability of perceiving the environment and responding in a timely fashion.
- *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 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 research are suitable for WS. For example, in the BDI models, agents have mental states consisting of belief, desire and intension that control their behaviours (Rao and Geogreff, 1991; Wooldridge, 2000). 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 (Gottschalk, *et al.*, 2002). Few of existing IT assets can be considered as agents in these models.

Our agent model is from a software engineering perspective. 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 (Zhu, 2001). 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 the modelling language CAMLE, how agents change their casteship is described by migration relations. There are two kinds of migration relationships: migrate and participate. A migrate relation from caste A to B means that an agent of caste A can retreat from caste A and join caste B. A participate relation from caste A to B means that an agent of caste A can join caste B while retaining casteship of A.

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. The organisational structure of a MAS is represented in a caste model in CAMLE. It describes the castes and their inheritance, whole-part and migration relations. Figure 2 below summaries the graphical notation of caste diagrams in CAMLE.

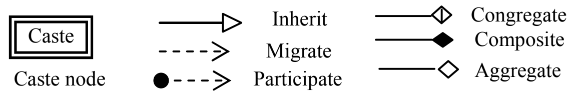


Figure 2 *Graphical notation of caste diagrams*

In the context of WS, service providers and 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. Figure 3 shows the architecture of WS in a caste model at a very high level of abstraction. 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.

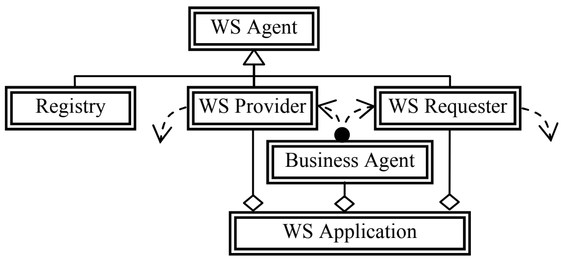


Figure 3 *Caste diagram of WS architecture*

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 of MAS: 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 the meanings of the messages. Of course, 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 at syntactic levels. The communications and collaborations between agents are described in collaboration models in CAMLE.

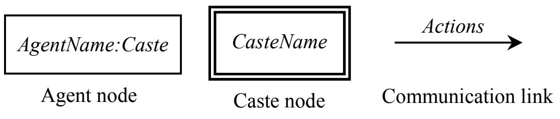


Figure 4 *Graphical notation of collaboration diagrams*

There are two types of nodes in a collaboration diagram. An agent node represents a specific agent. A caste node represents any agent in a caste. An arrow from node A to node B represents that the visible behaviour of agent A is observed by agent B. Therefore, agent A influences agent B. When agent B is particularly interested in certain activities of agent A, the activities can also be annotated to the arrow from A to B.

It is worthy noting that although this model looks similar to the collaboration diagrams in UML, there are significant differences in the semantics. In OO paradigm, what is annotated on the arrow from A to B is a method of B. It represents a method call from object A to object B, and consequently, object B must execute the method. In contrast, in CAMLE the action annotated on an arrow from A to B is a visible action of A. Moreover, agent B does not necessarily respond to agent A's action. The distinction indicates the shift of modelling focus from controls represented by the method calls in OO paradigm to collaborations represented by signalling and observation of visible actions. It fits well with the autonomous nature of agents.

One of the complications in the development of collaboration models is to deal with agents' various behaviours in different scenarios. By scenario, we mean a typical situation in the operation of the system. In different scenarios, agents may take different actions, pass around different sequences of messages even communicate with different agents. Therefore, it is better to describe different scenarios separately. The collaboration model in CAMLE supports the separation of scenarios by including a set of collaboration diagrams. Each diagram represents one scenario. In such a scenario-specific collaboration diagram, actions annotated on arrows can be numbered by their temporal sequence. In addition to scenario-specific collaboration diagrams, a general collaboration diagram is also associated to each compound caste to give an overall picture of the communications between all the component agents by describing all visible actions that the component agents may take and all possible observers of the actions. Figure 5 shows the general collaboration diagram of WS architecture.

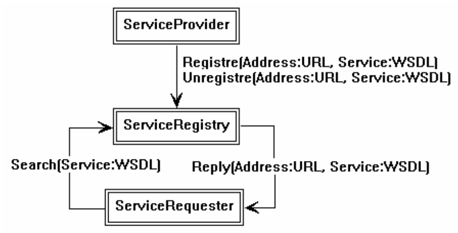

ServiceProvider
Registre(Address:URL, Service:WSDL)
Unregistre(Address:URL, Service:WSDL)
ServiceRegistry
Search(Service:WSDL)
Reply(Address:URL, Service:WSDL)
ServiceRequester

Figure 5 *Collaboration model of WS architecture*

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 6 gives the format and an example of scenario diagrams, which depicts the situation that an auctioneer informs the agent that its bid failed.

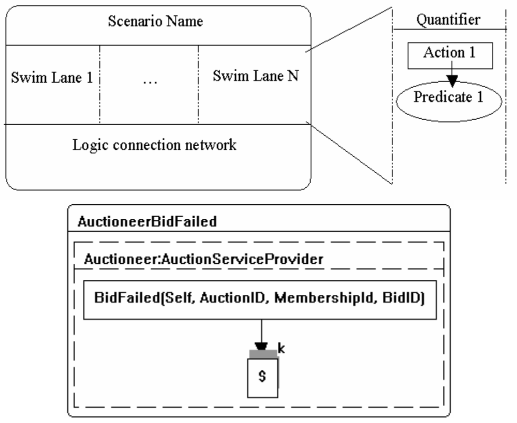


Figure 6 *Format and example of scenario diagram*

Behaviour diagrams describe agents' designed behaviour in certain scenarios. Figure 7 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.

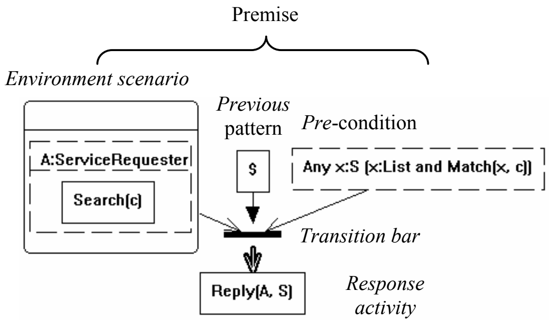
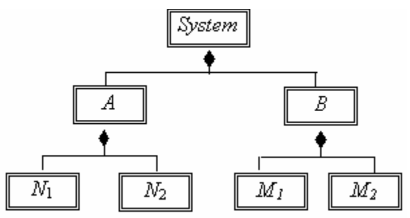
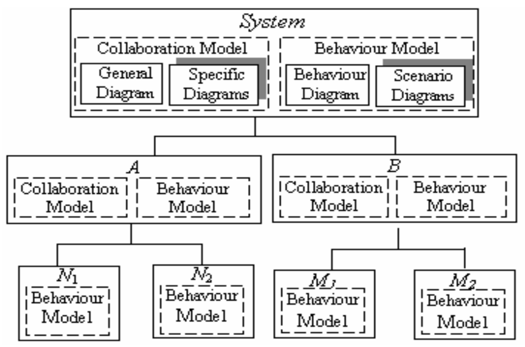


Figure 7 *Example behaviour rule of service registry*

The power of agent-based systems has been best demonstrated in dynamic environments (Jennings and Wooldridge, 1998; Huhns and Singh, 1997), 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 (Zhu, 2001a, 2001c), 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.



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



(b) Collaboration Models and Behaviour models

Figure 8 *Relationship between different kinds of CAMLE Models*

The modelling language CAMLE supports modelling complex systems at various levels of abstraction. Models of coarse granularity at a high level of abstraction can be refined into more detailed fine granularity models. At the top level, a system can be viewed as an agent that interacts with users and/or other systems in its external environment. This system can be decomposed into a number of subsystems interacting with each other. A sub-system can also be viewed as an agent and further decomposed. As analysis deepens, a hierarchical structure of the system emerges. In this way, the compound agent has its functionality decomposed through the decomposition of its structure. Such a refinement can be carried on until the problem is specified adequately in detail. Thus, a collaboration model at system level that specifies the boundaries of the application can be eventually refined into a hierarchy of collaboration models at various abstraction levels. Of course, the hierarchical structure of collaboration diagrams can also be used for bottom-up design and composition of existing components to form a system. As illustrated in Figure 8, the hierarchical relationship between the collaboration and behaviour diagrams associated to the castes are isomorphic to the whole-part relations between castes as defined in the caste diagram. Readers are referred to (Zhu, 2001a, 2003; Shan and Zhu, 2003, 2004a, 2005; Zhu and Shan, 2005a, 2005b) for more details of the CAMLE language and its meta-model.

2.2 CAMLE's modelling environment and tools

A software environment to support the process of system analysis and modelling in CAMLE has been designed and implemented (Zhu and Shan, 2005b). The main functionalities of the environment are:

(1) *Model construction*. It consists of a set of graphical editors to support the construction of models and tools for version control and configuration management.
(2) *Model consistency check*. A set of consistency constraints on CAMLE models is formally defined as an integral part of CAMLE language (Shan and Zhu, 2004b). These consistency constraints are checked by the CAMLE environment where automated tools are implemented.
(3) *Automated generation of formal specifications*. It transforms graphic models into the corresponding formal specifications in SLABS.

Figure 9 shows the architecture of the environment. The modelling environment provides a graphical user interface shown in Figure 10 to support the construction of models progressively and evolutionarily. The diagram editor supports manual editing of models through a graphic user interface. The well-formedness checker ensures that the user entered models are well-formed. The diagram generator can generate partial models (incomplete diagrams) from existing diagrams to help users in model construction. The rules to generate partial models are based on the consistency constraints so that the generated partial diagrams are consistent with existing ones according to the consistency conditions. The details of the consistency checkers and the formal specification generator are presented in (Shan and Zhu, 2004b) and (Zhu and Shan, 2005b), respectively.

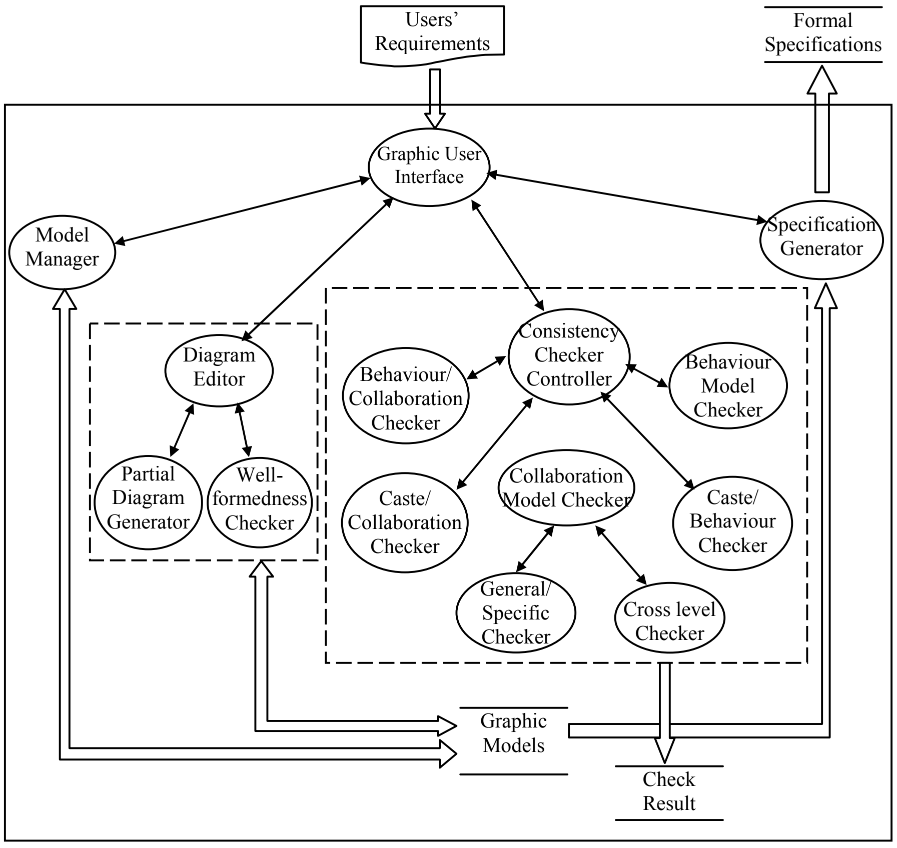


Figure 9 The Architecture of CAMLE Environment

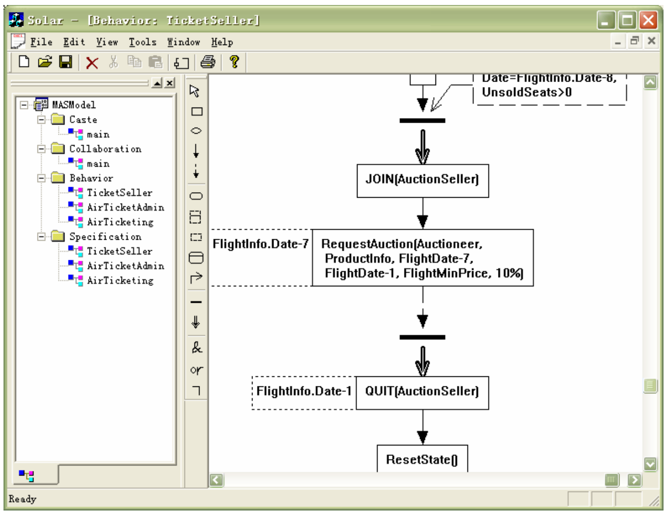


Figure 10 *CAMLE's Graphical User Interface for model construction*

CAMLE's project management functionality enables grouping diagrams into various projects so that models from different perspectives are separated. Each model can be processed and transformed independently.

In the case studies, a number of systems have been modelled in CAMLE. Their formal specifications in SLABS are generated. These systems include the following.

A. United Nations' Security Council: The organisational structure and the work procedure to pass resolutions were modelled and a formal specification of the system in SLABS was generated. Details of the case study as well as modelling in other agent-oriented modelling notations can be found on AUML's website [(1)].

B. Amalthaea: Amalthaea is an evolutionary multi-agent system developed at MIT's Media Lab to help the users to retrieve information from the Internet (Moukas, 1997). The system was modelled and a formal specification was generated.

C. University: This is a partial model of the university organisation and work procedures. The objective of the case study was not to provide a complete model; instead, it aims at providing illustrative examples to demonstrate the style of modelling in CAMLE. Details of the example can be found in (Shan and Zhu, 2004a).

In this paper, we will also demonstrate the use of the CAMLE language and modeling environment in the development of WS applications.

---

1 URL: http://www.auml.org/

3. CONSTRUCTION OF MODELS

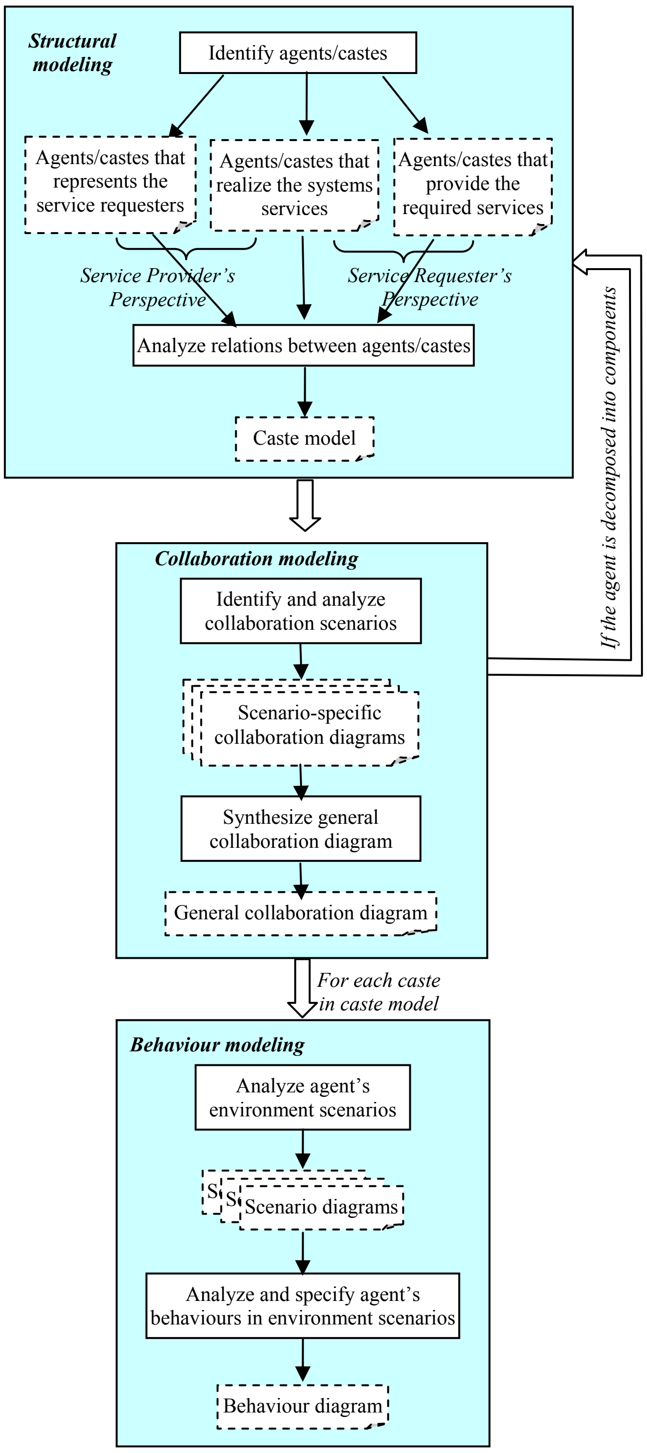
The CAMLE methodology of agent oriented software development is model-driven (Zhu and Shan, 2005b). It consists of a number of steps that involve model construction, model analysis, model transformation, and model-based validation verification and testing, etc. In this section, we investigate the construction of models in agent-oriented modelling language and environment CAMLE for the development of WS application systems.

As shown in Figure 11, the process of the construction of WS application models consists of three main iterative phases. The first phase is structural modelling aiming at developing a caste model of the structure of the system. The second phase is collaboration modelling aiming at a macro-level model of the dynamic behaviour of the system in terms of collaborations between various parts of the systems as well as the service requesters and other service providers. The final phase is behaviour modelling aiming at a clear specification of the behaviours of the individual elements of the system that implements the services.

The following will illustrate the process of model construction with an example of online auction services. We will demonstrate that CAMLE supports the requirements analysis, specification and design of WS applications for developing both service provider software and service requester systems. In particular, it supports modelling from service provider's perspective to explicitly model the service provider system without giving away internal information, to explicitly specify the service provider's assumptions on the requester without the over-restricting the uses of the services. It also supports modelling from service requester's perspective to design the requester software in the context of service provider with access to the design knowledge provided by the service provider, to combine the requester's internal business logic with the required services.

3.1 Service provider's perspective

As shown in Figure 11, the first step in the requirements analysis and model construction is to identify the types of agents that participate in the operation of the system and specify them as castes. Sub-caste, whole-part and migration relations between these castes are then identified so that a caste model can be constructed. For example, 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



Structural modeling
Identify agents/castes
Agents/castes that represents the service requesters
Agents/castes that realize the systems services
Agents/castes that provide the required services
Service Provider's Perspective
Service Requester's Perspective
Analyze relations between agents/castes
Caste model
If the agent is decomposed into components
Collaboration modeling
Identify and analyze collaboration scenarios
Scenario-specific collaboration diagrams
Synthesize general collaboration diagram
General collaboration diagram
For each caste in caste model
Behaviour modeling
Analyze agent's environment scenarios
S
S
Scenario diagrams
Analyze and specify agent's behaviours in environment scenarios
Behaviour diagram

Figure 11 *The process of model construction in the development of web services applications*

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 12. 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.

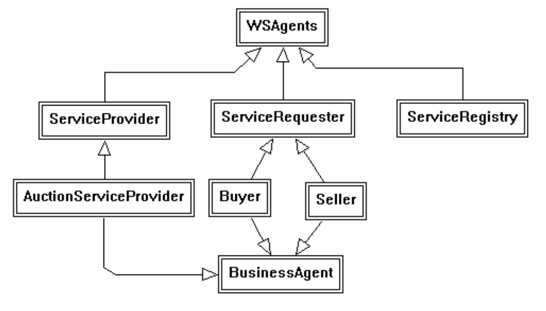


Figure 12 *Caste diagram from provider's perspective*

Scenario analysis plays an important role in the requirements analysis and the construction of detailed system behaviour models. 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. There are, therefore, 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. The outcome of the interactions between a buyer and the auctioneer can be successful or not successful. Therefore, two sub-cases of the second scenario can be identified. Each scenario is modelled by a scenario-specific collaboration diagram, as shown in Figure 13.

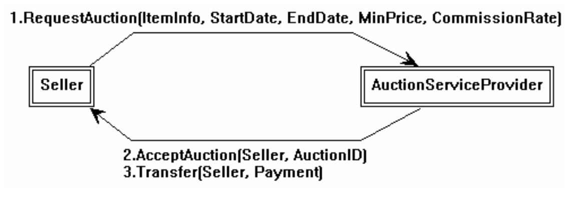
From these scenario-specific collaboration diagrams, a general collaboration diagram can be derived to summarise the communications between the agents. The communications of the auction service provider with the sellers and buyers are depicted in the generic collaboration diagram in Figure 14. Collaboration model provides a system level global model of the system.

Collaboration diagrams are expressive enough to describe workflows in the form of action sequences. However, it cannot express the semantics of business rules. For example, in the interaction with buyers, an auction service provider must follow the following rules.

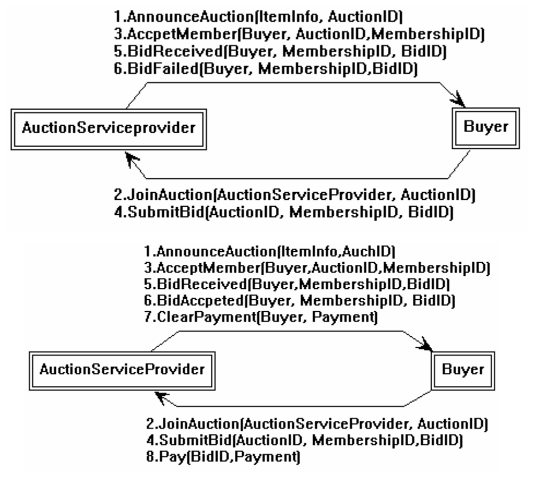
(1) When a buyer requests to join the auction, its credit must be checked and the membership issued if its credit is OK;
(2) When receives a bid from a member buyer, the auctioneer must acknowledge the receipt of the bid with a unique bid identifier;
(3) Every received bid must be compared with the current 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.

The sequence of actions in the above statements is clearly specified by the collaboration model given in Figure 13. However, the collaboration model does not define what meant by a bid is beaten by another and who is the final winner. To clarify such semantics associated to each action in a collaboration model, it is necessary to define the rules that govern the behaviour of each agent. Therefore, our next step of development is to find out the behaviour rules and specify them in the form of behaviour models.

In the example of online auction services, the above set of behaviour rules specifies how the auction service provider should behaviour in the interaction with buyers. It can be


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 13 *Scenario-specific collaboration diagrams*

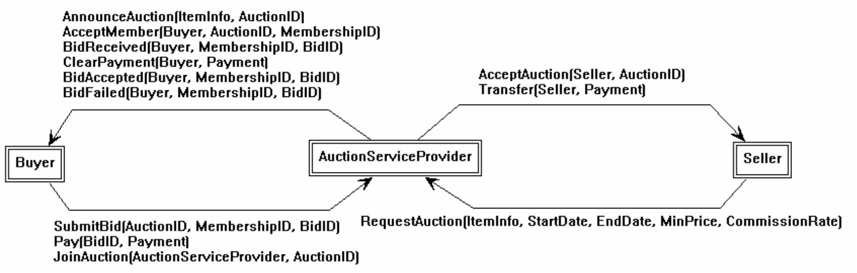
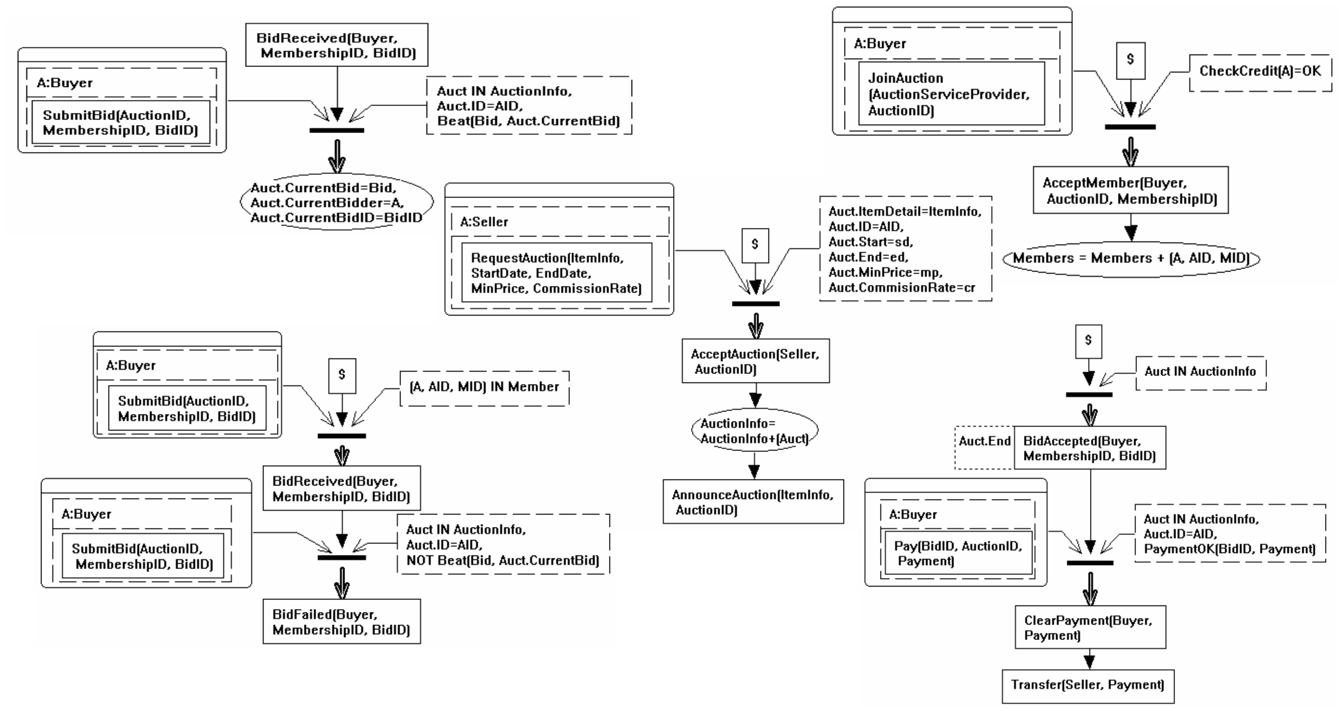

AnnounceAuction(ItemInfo, AuctionID)
AcceptMember(Buyer, AuctionID, MembershipID)
BidReceived(Buyer, MembershipID, BidID)
ClearPayment(Buyer, Payment)
BidAccepted(Buyer, MembershipID, BidID)
BidFailed(Buyer, MembershipID, BidID)
AcceptAuction(Seller, AuctionID)
Transfer(Seller, Payment)
Buyer
AuctionServiceProvider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

Figure 14 *Generic collaboration diagram from auction service provider's perspective*

specified in CAMLE by the behaviour diagram for the Auction Service Providers caste as shown in Figure 15. Similarly, there are also behaviour rules for the service provider to interact with Sellers, also shown in the behaviour diagram.

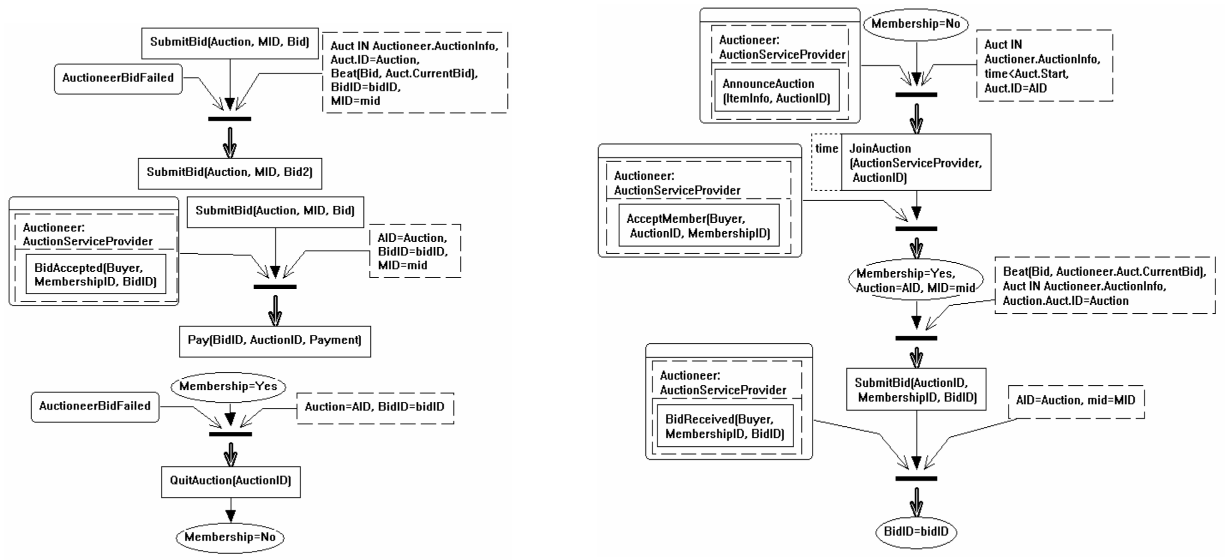
In the development of a service, certain assumptions on the service requesters' behaviour must be made. For example, in the interactions with the auction service provider, the buyers must follow an interaction protocol that consists of the following rules.

(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 failed.



BidReceived(Buyer, MembershipID, BidID)
A:Buyer
SubmitBid(AuctionID, MembershipID, BidID)
Auct IN AuctionInfo, Auct.ID=AID, Beat(Bid, Auct.CurrentBid)
Auct.CurrentBid=Bid, Auct.CurrentBidder=A, Auct.CurrentBidID=BidID
A:Buyer
JoinAuction (AuctionServiceProvider, AuctionID)
$
CheckCredit(A)=OK
AcceptMember(Buyer, AuctionID, MembershipID)
Members = Members + (A, AID, MID)
A:Seller
RequestAuction(ItemInfo, StartDate, EndDate, MinPrice, CommissionRate)
$
Auct.ItemDetail=ItemInfo, Auct.ID=AID, Auct.Start=sd, Auct.End=ed, Auct.MinPrice=mp, Auct.CommisionRate=cr
AcceptAuction(Seller, AuctionID)
AuctionInfo= AuctionInfo+(Auct)
AnnounceAuction(ItemInfo, AuctionID)
A:Buyer
SubmitBid(AuctionID, MembershipID, BidID)
$
(A, AID, MID) IN Member
BidReceived(Buyer, MembershipID, BidID)
A:Buyer
SubmitBid(AuctionID, MembershipID, BidID)
Auct IN AuctionInfo, Auct.ID=AID, NOT Beat(Bid, Auct.CurrentBid)
BidFailed(Buyer, MembershipID, BidID)
$
Auct IN AuctionInfo
Auct.End
BidAccepted(Buyer, MembershipID, BidID)
A:Buyer
Pay(BidID, AuctionID, Payment)
Auct IN AuctionInfo, Auct.ID=AID, PaymentOK(BidID, Payment)
ClearPayment(Buyer, Payment)
Transfer(Seller, Payment)

Figure 15 *A behaviour rule of service provider*



SubmitBid(Auction, MID, Bid)
AuctioneerBidFailed
Auct IN Auctioneer.AuctionInfo, Auct.ID=Auction, Beat(Bid, Auct.CurrentBid), BidID=bidID, MID=mid
SubmitBid(Auction, MID, Bid2)
Auctioneer: AuctionServiceProvider
BidAccepted(Buyer, MembershipID, BidID)
SubmitBid(Auction, MID, Bid)
AID=Auction, BidID=bidID, MID=mid
Pay(BidID, AuctionID, Payment)
AuctioneerBidFailed
Membership=Yes
Auction=AID, BidID=bidID
QuitAuction(AuctionID)
Membership=No
Auctioneer: AuctionServiceProvider
AnnounceAuction (ItemInfo, AuctionID)
Membership=No
Auct IN Auctioneer.AuctionInfo, time<Auct.Start, Auct.ID=AID
time
JoinAuction (AuctionServiceProvider, AuctionID)
Auctioneer: AuctionServiceProvider
AcceptMember(Buyer, AuctionID, MembershipID)
Membership=Yes, Auction=AID, MID=mid
Beat(Bid, Auctioneer.Auct.CurrentBid), Auct IN Auctioneer.AuctionInfo, Auction.Auct.ID=Auction
Auctioneer: AuctionServiceProvider
BidReceived(Buyer, MembershipID, BidID)
SubmitBid(AuctionID, MembershipID, BidID)
AID=Auction, mid=MID
BidID=bidID

Figure 16 *Behaviour diagram for the Buyer caste*

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, which is shown in Figure 16.

Similarly, there is a set of behaviour rules for the Seller caste as well, shown in Figure 17.

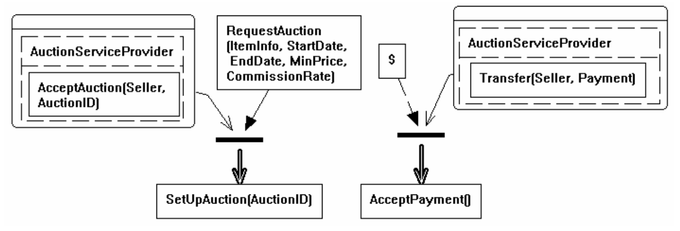

AuctionServiceProvider
AcceptAuction(Seller, AuctionID)
RequestAuction (ItemInfo, StartDate, EndDate, MinPrice, CommissionRate)
$
AuctionServiceProvider
Transfer(Seller, Payment)
SetUpAuction(AuctionID)
AcceptPayment()

Figure 17 *Behaviour diagram for Seller caste*

3.2 Service requester's perspective

We now discuss how CAMLE can be used to develop models from the requester's perspective. Generally speaking, the development of service requester software follows the same process of developing service providers. In many cases, a service requester is also a service provider and vice versa. For example, the auction service provider may well be a service requester of banking service and advertisement service, credit checking service, etc. Therefore, in the following discussion we omit the details that repeat what we have discussed in section 3.1.

Consider an online flight ticketing service that sells air tickets via an e-commerce website. It operates according to a set of business rules to determine how to set the prices for each ticket and when the change the price. For example, 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 with a discount price. Suppose that 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 in the sequel, must satisfy all the structure and behaviour requirements specified in the castes Ticket Sellers and Sellers. They can be specified as a sub-caste of Ticket Sellers and Seller to model their behaviours as shown in Figure 18. In the development of the caste Sell By Auction, software developers need to have access to the specifications of the castes of Auction Service Provider and Seller. It is not necessary to have access to the specification of the caste Buyer.

As shown in Figure 19, 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. However, how to evaluate different designs is beyond the scope of this paper.

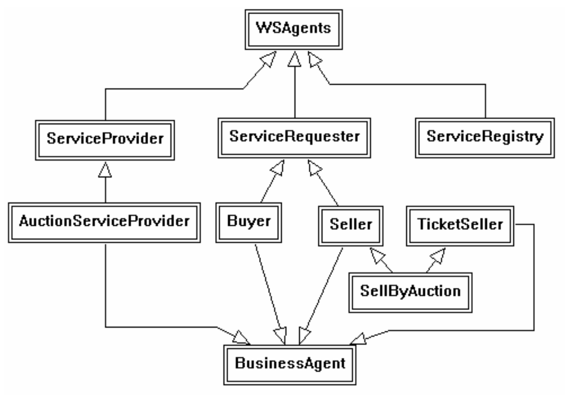


Figure 18 *A design of auction service user*

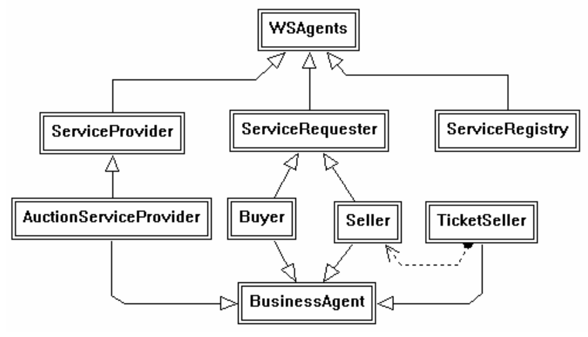


Figure 19 *Caste model from requester's perspective*

For the castes shown in Figure 19, 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.

It is worth noting that to implement this design, there are two posible approaches. First, the service provider of the auction service could implemented a caste of Seller so that when Ticket Seller joins the caste it will be able to access the auction service. Second, the developers of the service requester, i.e. the Ticket Seller, could also implement the caste Seller as a part of their system and integrate the caste with the service provider software. It is an open problem that which approach is better.

4. CONSISTENCY CHECK

Assume that three services *A*, *B* and *C* were developed by three different vendors with interdependency relationships illustrated in Figure 20. Here, service *A* is used by services *B* and *C*. Service *A* also uses service *B*; while service *B* uses both services *A* and *C*.

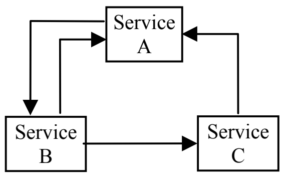


Figure 20 *An example of interdependence between services*

In the development of each service, the developer will construct a model that consists of three parts: (a) modelling the agents that provide the service, (b) modelling the agents that request its service, and (c) modelling the agents that implement its internal business logic of the service. The first two parts should be public to the users of the service. The last part should be hidden from the public as the internal information. These three parts are not adequate for the development of the service if it also uses other services. In that case, the public parts of models of the requested services are used. This leads to a 'jigsaw puzzle' like structure of the system as illustrated in Figure 21.

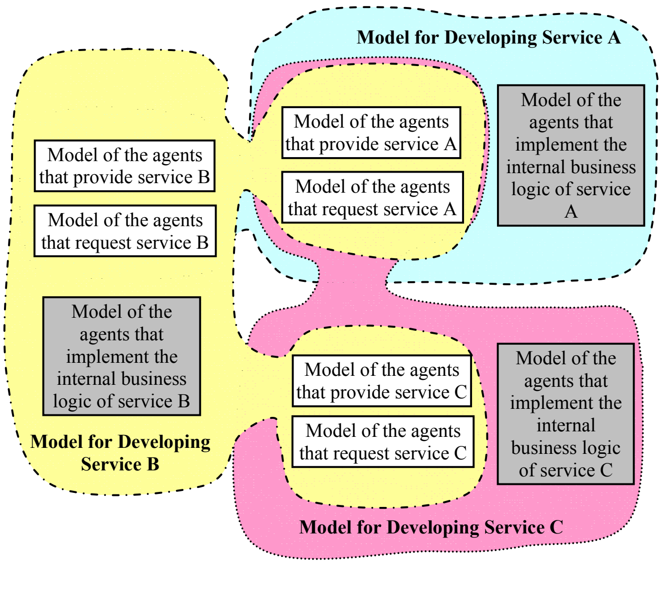

Model for Developing Service A
Model of the agents that provide service A
Model of the agents that request service A
Model of the agents that implement the internal business logic of service A
Model of the agents that provide service B
Model of the agents that request service B
Model of the agents that implement the internal business logic of service B
Model for Developing Service B
Model of the agents that provide service C
Model of the agents that request service C
Model of the agents that implement the internal business logic of service C
Model for Developing Service C

Figure 21 *Illustration of the 'jigsaw' structures of the models*

In section 3, we have discussed and demonstrated by a non-trivial example how the 'jigsaw puzzle' can be built in the agent-oriented modelling language CAMLE and its automated modelling environment. The question is now how to ensure the system will work, especially the consistency between the models from different perspectives. Because the whole 'jigsaw puzzle' is developed by different vendors, it is impossible to check the consistency of the whole system via analysis of the whole structure (e.g. all the elements in Figure 21), because each service provider or requester has only limited access to the information. Fortunately, the caste language facility enables us to achieve the whole systems' consistency through maintaining the consistency of each piece of the jigsaws.

CAMLE's consistency check facility provides a strong support to ensure the consistency between models.

First, a set of consistency constraints are defined on CAMLE models. These explicitly defined consistency constraints and the automated consistency checking tools help to ensure the quality of the model from one perspective, i.e. the consistency of each jigsaw piece. A very important feature of CAMLE language and its modelling environment is that it supports the separation of perspectives in the way that the model from each perspective is self-contained and satisfies all consistency conditions. In our case study, it is noticed that the consistency constraints are effective to identify the missing assumptions of the requesters in the development of a model from the provider's perspective. For example, in the development of the models for auction services, 2 projects were developed separately. The first project was the model from auction service provider's perspective. The model was constructed, consistency checked and formal specification generated. The project of ticket seller as the auction service requester was developed after the auction service provider project is finished. A part of models developed in the auction service provider project was imported. The caste model was revised to represent the perspective of the WS application from the auction service provider's point of view. Consistency of the model is then checked and when it passed the consistency checking, the formal specification is generated.

Second, and most importantly, inconsistency between models of different perspectives can be effective identified so that the consistency of all jigsaw pieces implies the consistency of the whole system. That is, the design of the consistency constraints has the feature that if every jigsaw piece of a system passes the consistency check, the system obtained by putting all pieces together can also pass the consistency check. This set of constraints includes the following types.

- *Intra-model consistency constraints*: the consistency conditions impose on each type of caste, collaboration and behaviour models. This kind of constraints can be further divided into the following two sub-types.
  - *Intra-diagram consistency constraints*: the consistency conditions imposed on each diagram.
  - *Inter-diagram consistency constraints*: the consistency condition the imposed on the relationships among a set

of diagrams that form a specific type of model.

- *Inter-model consistency constraints*: the constraints imposed on the relationships between models. This kind of consistency constraints can be further divided into the following two types.
  - *Horizontal consistency*: the consistency between models of different types but on the same level of abstraction.
  - *Vertical consistency*: the consistency between models of the same type, but on the different level of abstraction. Vertical consistency constraints can be a *local* constraint, which is imposed on the relationships between models on two adjacent levels of abstraction. It can also be or a *global* constraint, which is imposed on the overall structure of the models as they are related to each other.

The following table summarises the number of consistency conditions defined on CAMLE models and checked by automated consistency checking tools in CAMLE modelling environment.

Table 1 Summary of CAMLE's consistency constraints

		Horizontal Consistency	*Vertical Consistency*	
			Local	*Global*
Intra-model	*Intra-diagram*	*10*	–	–
	Inter-diagram	*8*	*8*	–
Inter-model		*4*	*1*	*4*

Readers are referred to (Shan and Zhu, 2004a) for details of the consistency constraints and the automated consistency checking tools.

5. GENERATION OF FORMAL SPECIFICATIONS

Using CAMLE's specification generator, a model that passed consistency check can be automatically transformed into a formal specification in SLABS.

In SLABS, each caste is specified in the following syntax, which can also be in the equivalent graphic format shown in Figure 22.

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

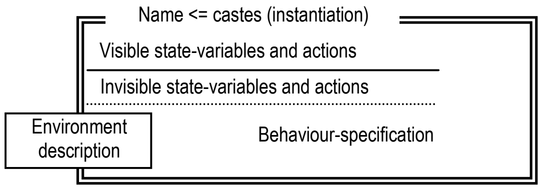

Name <= castes (instantiation)
Visible state-variables and actions
Invisible state-variables and actions
Environment description
Behaviour-specification

Figure 22 *Format of caste description in SLABS*

The SLABS language uses transition rules as a facility to explicitly specify how an agent's observations on the environment are related to its 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 its previous behaviour.

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

The formal specification of caste Buyer given in Figure 23 is generated from the CAMLE models. Similarly, the specifications of other castes can be generated. Details are omitted for the sake of space.

Using 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 is the winner, which 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 WS description methods.

Readers are referred to (Zhu, 2005) for the details of a formal system called Scenario Calculus that enables formal reasoning about the properties of multi-agent systems.

```
              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 ]   |→ time: Join_Auction(Auctioneer, AID);
                    if Auctioneer:[Announce_Auction(d, AID)];
                    where Auct∈ Auctioner.AuctionInfo & time < Auct.Start & Auct.ID=AID
Auctioneer: Auction
Service Provider

<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 23 *Specifications generated from models*

6. CONCLUSION

In this paper, we proposed a model-driven approach to the development of WS applications. We investigated the uses of the graphic agent-oriented modelling language CAMLE and its automated modelling environment 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 as well as the context in which the services are used. The models from different perspectives are separated so that internal information is hidden, yet the models are self-contained. This enables consistency checking, specification generation as well as other processes and analysis of the models. The explicit specification of the service provider's assumptions on the service requester's behaviours also enables proofs of the properties of the system without full knowledge of the requester's system, which is usually unavailable to the service providers. 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.

This paper has focused on the structure of the models, the model construction process, model consistency check and the generation of formal specifications. We have designed and implemented an experimental agent-oriented programming language called SLABSp based on the meta-model underlying the formal specification language SLABS (Wang, Shen and Zhu, 2004, 2005a, 2005b). The language extends Java with the caste facility that allows agents to dynamically join into and quit from castes. It also implements the scenario description and environment description facilities so that programming distributed and concurrent systems can be at a very high level of abstraction. The language is implemented with a runtime support system based on Java Virtual Machine. The source code of a SLABSp program is compiled into Java code with components that implement the runtime support system. As shown in (Wang, Shen and Zhu, 2005b), formal specifications in SLABS can be translated into SLABSp executable programs straightforwardly. Therefore, once a model of WS application is constructed, a formal specification of the system can be generated as demonstrated in this paper using the CAMLE automated tools. The formal specification can then be implemented in SLABSp fairly straightforwardly. The research on this topic will be reported separately.

Another use of formal specifications is to reasoning about the properties of the systems. A formal logic called Scenario

Calculus has been developed (Zhu, 2005). It can be used to reasoning about how a multi-agent system will behave in certain scenarios. In particular, it can be used to reason about whether a scenario will occur, whether a state of a system expressed in the form of a scenario will be stable once it reaches the state, etc. More details of the formal logic system can be found in (Zhu, 2005).

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 based on the Scenario Calculus to enable 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.

ACKNOWLEDGEMENT

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

REFERENCES

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

BPML.org (2004) *The BPML specification version 1.0*, URL: http://www.bpmi.org, accessed on 2nd August 2004.

Bresciani, P., Giorgini, P., Giunchiglia, F., Mylopoulos, J. and Perini, A. (2002) *TROPOS: An Agent-Oriented Software Development Methodology*. Technical Report DIT-02-015, Informatica Telecomunicazioni, Università degli Studi di Trento, 2002.

Choren, R., Garcia, A., Lucena, C. and Romanovsky, A. (Eds.) (2005) *Software Engineering for Multi-Agent Systems III*, Lecture Notes in Computer Science, Vol. 3390, Springer.

Ciancarini, P. and Wooldridge, M. J. (eds.) (2001) *Agent-Oriented Software Engineering*, Lecture Notes in Computer Science, Vol. 1957, Springer-Verlag, April.

Dam, K. H. and Winikoff, M. (2003) ‘Comparing agent-oriented methodologies’, in *Proc. of 5th International Bi-Conference Workshop on Agent-Oriented Information Systems*, Melbourne, Australia, July.

DAML.org (2004) *OWL-S 1.1 Release*, URL: http://www.daml.org/services/owl-s/1.1/, accessed on 25th November 2004.

DAML.org (2001) *The DAML Services Coalition. DAML-S: A Semantic Markup For Web Services*, URL: http://www.daml.org/services/daml-s/2001/10/daml-s.pdf.

Garcia, A., Lucena, C., Zambonelli, F., Omicini, A. and Castro, J. (Eds) (2003) *Software Engineering for Large-Scale Multi-Agent Systems*, Lecture Notes in Computer Science, Vol. 2603, Springer; November.

Giorgini, P., Muller, J. P. and Odell, J., (Eds.) (2004) *Agent-Oriented Software Engineering IV*, Lecture Notes in Computer Science, Vol. 2935, Springer-Verlag, March.

Giunchiglia, F., Odell, J. and Weiß, G. (Eds.) (2003) *Agent-Oriented Software Engineering III*, Lecture Notes in Computer Science, Vol. 2585 , Springer-Verlag, December.

Gottschalk, K. *et al.* (2002) ‘Introduction to web services architecture’, *IBM Systems Journal*, Vol.41, No.2, pp.170-177.

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

Jennings, N. R. (1999) ‘Agent-oriented software engineering’, *Multi-Agent System Engineering, Proceedings of 9th European Workshop on Modelling Autonomous Agents in a Multi-Agent World*, Valencia, Spain, Springer, pp.1-7.

Jennings, N. R. (2000) ‘On agent-based software engineering’, *Artificial Intelligence*, Vol.117, pp.277-296.

Jennings, N. R. and Wooldridge, M. J. (1998) *Agent Technology: Foundations, Applications, And Markets*, Springer.

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

Lange, D.B. (1998) ‘Mobile objects and mobile agents: the future of distributed computing?’ *Proceedings of The European Conference on Object-Oriented Programming*.

Lau, C. and Ryman, A. (2002) ‘Developing XML Web services with WebSphere studio application developer’, *IBM Systems Journal*, Vol. 41, No.2, pp.178-197.

Leymann, F. (2001) *Web Services Flow Language*, IBM Corporation.

Leymann, F., Roller, D. and Schmidt, M.-T. (2002) ‘Web services and business process management’, *IBM Systems Journal*, Vol. 41, No.2, pp.198-211.

Lucena, C., Garcia, A., Romanovsky, A., Castro, J. and Alencar, P. S. C. (Eds.) (2004) *Software Engineering for Multi-Agent Systems II*, Lecture Notes in Computer Science, Vol. 2940, Springer, March.

Moukas, A. (1997) ‘Amalthaea: information discovery and filtering using a multi-agent evolving ecosystem’, *Journal of Applied Artificial Intelligence,* Vol.11, No.5, pp.437–457.

Odell, J., Giorgini, P., and Muller, J. P. (Eds.) (2005) *Agent-Oriented Software Engineering V*, Lecture Notes in Computer Science, Vol. 3382, Springer; January.

Odell, J., Van Dyke Parunak, H. and Bauer, B. (2001) ‘Representing agent interaction protocols in UML’, Wooldridge, M. (Eds): *Agent-Oriented Software Engineering*, Springer, pp.121-140.

Rao, A.S. and Georgreff, M.P. (1991) ‘Modeling rational agents within a BDI-architecture’, *Proc. of the International Conference on Principles of Knowledge Representation and Reasoning*, pp.473-484.

Shan, L. and Zhu, H. (2003) ‘Modelling and specification of scenarios and agent behaviour’, *IEEE/WIC conference on Intelligent Agent Technology (IAT’03)*, Halifax, Canada, IEEE CS, pp32-38.

Shan, L. and Zhu, H. (2004a) ‘CAMLE: a caste-centric agent-oriented modelling language and environment’, *Proc. of SELMAS'04 at ICSE'04*, Edinburgh, UK, IEE, pp.66-73.

Shan, L. and Zhu, H. (2004b) ‘Consistency check in modeling multi-agent systems’, *Proc. of COMPSAC’04*, IEEE CS, Hong Kong, September, pp.114-121.

Shan, L. and Zhu, H. (2005) ‘CAMLE: A caste-centric agent-oriented modelling language and environment’, in Choren, R., Garcia, A., Lucena, C. and Romanovsky, A. (Eds.): *Software Engineering for Multi-Agent Systems III*: *Research Issues and Practical Applications*, Lecture Notes in Computer Science, Vol. 3390, Springer, pp.144-161.

Stal, M. (2002) 'Web Services: beyond component-based computing', *Communications of ACM*, Vol.45, No.10, pp.71-76.

Thatte, S. (2001) *XLANG-Web Services for Business Process Design*, Microsoft Corporation.

Tsai, W. T., Paul, R., Yu, L., Saimi, A., Cao, Z. (2003) 'Scenario-based Web Service testing with distributed agents', *IEICE Transactions on Information and Systems*, Vol. E86-D, No. 10, pp2130-2144.

Tsai, W. T., Song, W., Paul, R., Cao, Z. and Huang, H. (2004) 'Service-oriented dynamic reconfiguration framework for dependable distributed computing, *Proc. of 28th Annual International Computer Software and Applications Conference* (*COMPSAC*'04).

Tsai, W. T., Wei, X., Chen, Y., Xiao, B., Paul, R., and Huang, H. (2005a) 'Developing and assuring trustworthy Web Services', *Proc. of 7th International Symposium on Autonomous Decentralized Systems* (*ISADS*'05).

Tsai, W. T., Liu, X., Chen, Y. and Paul, R. (2005b) 'Simulation verification and validation by dynamic policy enforcement', *Proc. of 38th Annual Simulation Symposium* (*ANSS*'05), IEEE Computer Society.

Wang, J., Shen, R. and Zhu, H. (2004) 'Scenario Mechanism in Agent-Oriented Programming', Proc. *of APSEC'04*.

Wang, J., Shen, R. and Zhu, H. (2005a) 'Agent oriented programming based on SLABS', *Proc. of COMPSAC'05*, Edinburgh, UK, pp127-132.

Wang, J., Shen, R. and Zhu, H. (2005b) 'Towards an agent-oriented programming language with caste and scenario mechanisms', *Proc. of AAMAS'05*, Utrecht, Netherlands. (*to appear*)

Weiß, G. and Ciancarini, P. (eds.) (2002) *Agent-Oriented Software Engineering II*, Lecture Notes in Computer Science, Vol. 2222, Springer-Verlag, March.

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

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

Zambonelli, F. and Omicini, A. (2004) 'Challenges and research directions in agent-oriented software engineering', *Autonomous Agents and Multi-Agent Systems*, Vol. 9, pp.253–283.

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

Zhu, H. (2001b) 'The role of caste in formal specification of MAS', *Proc. of PRIMA'2001*, Springer, pp.1-15.

Zhu, H. (2001c) 'Formal specification of agent behaviour through environment scenarios', in Rash, J. *et al.* (Eds): *Formal Aspects of Agent-Based Systems*, Lecture Notes in Computer Science, Springer, pp.263-277.

Zhu, H. (2003) *A Formal Specification Language for Agent-Oriented Software Engineering*, Technical Report DoC-TR-03-01, Department of Computing, Oxford Brookes University, Oxford, UK.

Zhu, H. (2005) 'Towards formal reasoning of emergent behaviour in multi-agent systems', *Proc. of SEKE'05*, Taipei, pp280-285.

Zhu, H. and Shan, L. (2005a) 'Agent-oriented modelling and specification of web services', *Proc. of Tenth IEEE International Workshop on Object-oriented Real-time Dependable Systems* (*WORDS'05*), IEEE CS Press, February, Sedona, Arizona, USA, pp152-159.

Zhu, H. and Shan, L. (2005b) 'Caste-centric modelling of multi-agent systems: the CAMLE modelling language and automated tools', in Beydeda, S. and Gruhn, V. (Eds.): *Model-driven Software Development*, *Research and Practice in Software Engineering*, Vol. II, Springer, pp57-89.

Zhu, H., Zhou, B., Mao, X., Shan, L., and Duce, D. (2004) 'Agent-oriented formal specification of Web Services', *Proc. of the AAC-GEVO'04 Workshop at GCC'04*, Springer, October.