A Framework for Agent-based Service-Oriented Modelling [*]

Zhi Jin
School of Electronics Engineering and Computer Science, Peking University
Key Laboratory of High Confidence Software Technologies, Ministry of Education
Beijing 100871, China. Email: zhijin@amss.ac.cn

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

Abstract

Service-oriented computing is becoming a direction of computing technology. For realising the mission of service-oriented computing, service-oriented architecture has been proposed to facilitate the application development via discoverable services distributed on Internet. That brings out service-oriented modelling as a new technical area to provide modelling and analysis techniques of service-oriented applications. The paper proposes a framework for agent-based service-oriented modelling. It treats both the service providers and the service requesters as service agents. Domain ontology has been used to provide the sharable domain knowledge as well as terminology for allowing the agents to understand each others. A modelling process has also been illustrated with the model development of an online auction service.

1. Introduction

Service-oriented modelling (SOC) aims to develop services and use available services as basic computing entities to support distributed computing applications [4]. It assumes that service requests would be satisfied through discovering and invoking available services dynamically and automatically. One of the main challenges for supporting the mission of SOC lies in the area of service-oriented modelling (SOM) [3], which has been considered as the first phase in the lifecycle of service-oriented application [15]. Research attentions have been paid to this area recently. Levi and Arsanjani, [9] and Endrei, Ang and Arsanjani, [14] for example, used goals to guide behavioural specification for components and extend component based analysis beyond traditional OOAD to model SOA application. Zimmermann, Krohdaul and Gee considered service-oriented modelling as a hybrid approach including a set of traditional techniques that incorporate object-oriented analysis and design, business process modelling and enterprise architecture description [17]. Arsanjani advocated an iterative and incremental service-oriented architecture modelling process that consists of identification, specification and realization of services, components as well as workflows [2].

Despite of these efforts, SOM discipline is still far from being mature. Most of these efforts are trying to shift existing software modelling techniques into the field of SOM, whereas some significant features of SOC are obviously ignored. First of all, Web Services technology is fundamentally different from the traditional distributed computing technologies [16]. In [8], Zhu et al. argued that the services in service-oriented applications are autonomous, active and persistent computational entities which control their own resources and their own behaviours and even show some kind of social ability such as collaborating with each other through dynamic discovery and invocation of other services.

Second, service-oriented technology enables dynamic software integration at runtime. To realize its full power, it does not only require the interfaces between integrated entities syntactically compatible, but more importantly, the interactions must be semantically correct. It is a major problem in the development of service-oriented applications to enable dynamic search of semantically correct services and to understand required services with correct meanings. It is widely recognised that background knowledge shared by services is essential for supporting the dynamic discovery and invocation of services. Unfortunately, these have not

[*]This work was supported by the National Natural Science Fund for Distinguished Young Scholars of China under Grant No. 60625204, the Key Project of National Natural Science Foundation of China under Grant No.60796324, and the National 973 Fundamental Research and Development Program of China under Grant No.2002CB312004. This work is finished when the first author visits Oxford Brookes University funded by Oxford Brookes' International Visiting Fellowship.

been taken into consideration in existing works on SOM.

The current state of art in SOM is that there is no unified modelling language for describing the models suitable for SOM, no systematic methodology and controllable process for conducting the modelling, and no usable tools or platforms for supporting the modelling. These are necessities of a mature modelling discipline. This paper is motivated by this consideration and aims to provide a systematic methodology and a controllable process as well as a modelling language. The main essences of the approach proposed in this paper include: (1) Using agents to capture the autonomy and activeness of participants in service-oriented computing; (2) Introducing domain ontology to provide sharable knowledge and terminology for these participants coming from different parties; and (3) Providing guidelines to help conducting the process of service modelling.

This paper is organized as follows. An agent-oriented SOM framework have been presented in Section 2. Section 3 addresses the modelling of domain ontology. Section 4 is devoted to the process of the agent-based SOM. And finally, Section 5 concludes the paper.

2. An Agent-based Framework

In [7, 8], a general framework for modelling information systems as well as Web Services was proposed based on the caste-centric approach to multi-agent systems. Two main principles for the development of service oriented systems were identified and stated, i.e. the *autonomy principle* and the *explicitness principle*.

The *autonomy principle* states that in order to achieve the full power of service oriented computing, both the service providers and the service requesters should be autonomous so that the interaction between them can be established dynamically and flexibly.

Based on this principles, Zhu et al. proposed a core structure for agent-based approach to service oriented applications [8]. In this structure, a service oriented application consists of a number of agents classified into a number of castes as illustrated in Figure 1. Each caste represents a role of parties involved in the service. Agents are the active computational entities that play the roles. They are autonomous and collaborate with each other to provide services and to obtain services. Each agent may possess a number of 'real world' entities, i.e. resources, that it can control and change their states through a set of operations, which reflect its capability. In addition to a set of actions that the agent is capable of performing, the definition of an agent also contains a set of behaviour rules that determine when to take action and which one to take. An agent also explicitly defines a set of other agents (which can be variable) in the systems as its neighbours that collaborate with. A caste defines these structural and behavioural features of the agents as a template so that agents can be instantiated through declaration at compiler time or dynamically created at runtime. Agents can also dynamically join a caste to obtain its capability and to be bound to its behaviour rules.

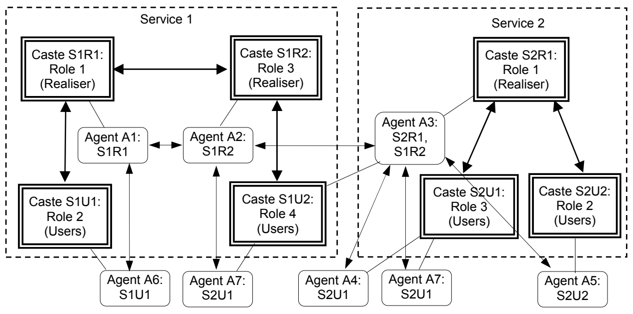
Service 1
Service 2
Caste S1R1: Role 1 (Realiser)
Caste S1R2: Role 3 (Realiser)
Caste S2R1: Role 1 (Realiser)
Agent A1: S1R1
Agent A2: S1R2
Agent A3: S2R1, S1R2
Caste S1U1: Role 2 (Users)
Caste S1U2: Role 4 (Users)
Caste S2U1: Role 3 (Users)
Caste S2U2: Role 2 (Users)
Agent A6: S1U1
Agent A7: S2U1
Agent A4: S2U1
Agent A7: S2U1
Agent A5: S2U2

Figure 1. The Core Structure

The *explicitness principle* states that all aspects of each service, including the capability, the structure of resources as well as the behaviour rules, must be explicitly specified covering both the syntax and semantic of the services. This enables the semantic correctness of service interactions to be assessable.

Applying this principle, the specification and model of a service and its castes that represent its service users and serve as the interface to the external must be published and searchable. Consequently, the core structure of caste centric agent-based approach to service orientation is altered into the architecture illustrated in Figure 2.

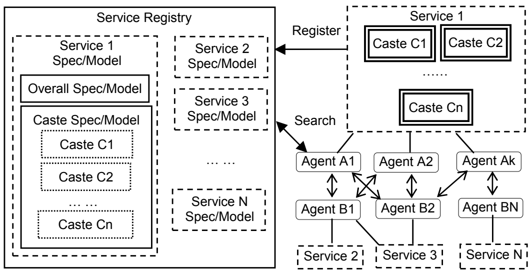

Figure 2. Structure of Service-Oriented Architecture

The architecture above provides a mechanism for registering and searching services at semantic level. It also place the formal specification and modelling of services at the centre of developing and utilization of services. In [8], it was proposed that agent-oriented formal specification language SLABS [6] and agent oriented modelling language CAMLE [12] are used for this purpose.

In addition to the above two principles, here we propose one more principle, which is called *competence principle*.

The *competence principle* states that for a service to be able to dynamically interact with other services meaningfully and correctly, it must be competent enough to understand other services' specifications, which according to the explicitness principle, is accessible.

For a service to be understandable to other services, its specification must be in a standard language, using standard terminology and encoding. A great amount of efforts have been reported in the literature on the standardisation of encoding, such as the WSDL and the representation of business process as well as ontology like OWL-S. However, how the domain knowledge are modelled at a high level of abstraction has not been addressed adequately. In [18, 19], Wang and Jin et al propose to use environment-based semantic capability specifications and to model the semantic capabilities of services as the effects imposed by services on the real world entities. This matches very well the agent approach advanced in [7, 8]. Combining these two approaches, this paper propose to add a new type of elements in the architecture, i.e. i.e. ontology of the 'real world' entities in application domains. Consequently, the architecture of SOM can be illustrated in Figure3.

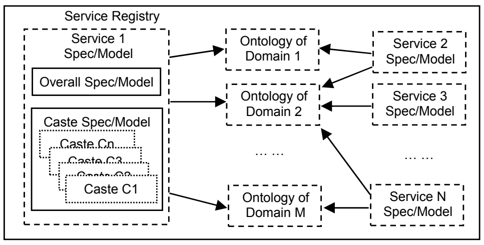

Figure 3. Structure of Service Registry

3. Domain Ontology Specification

The idea of environment-based semantic capability of services is motivated by the observation that the semantics of a software system concerns the environment on which the system will operate. In this sense, only considering the 'inner world' of a service is not enough for modelling its capability. It is necessary to distinguish the capability specification from the internal structure of the service and to allow the capability specification to refer to its 'outer world' that corresponds to the environment of the service.

The outer world which consists of 'real world' entities is outside the service. These entities are domain relevant and independent of the existence of any particular service and thus they can be shared by services from different parties. For modelling the 'real world' entities, a practical approach is to use ontology that is standardised and shared in the application domain. This technique has been widely used in agent-oriented architecture [10] for providing machine readable vocabularies and sharable domain knowledge.

This ontology needs to specify the domain entities as well as the attributes of these entities. Here, we include a portion of Auction Domain Ontology for illustrating the structure of the ontology as shown in Figure 4. This portion of ontology contains domain stateful entities, such as *Bid*, *Auction*, *Item*, etc, and domain stateless attributes such as *Date* and *Price*, *Identifier* etc. The ontology also contains associations among these entities. Obviously, this portion of ontology can be easily extended by including more types of entities and associations.

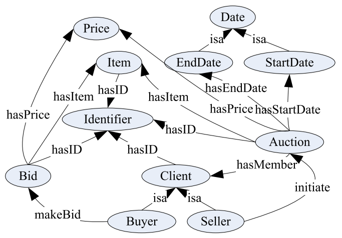

Figure 4. A Portion of Auction Domain Concept Structure

For those stateful entities, further specification needs to be given for describing their states and state transitions. Figure 5 shows the state diagrams of some stateful domain entities in *Auction* domain. For example, each *auction* could be in the states: *null* (i.e. it is not yet proposed), *created* (i.e. it has been proposed), *started* (i.e. after the start date), *end-in-Success* (i.e. there exists a successful bid) or *end-by-Overdue* (i.e. no bid succeed after the end date).

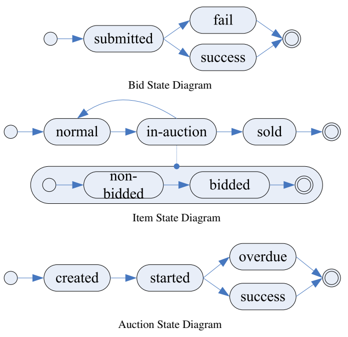

Figure 5. Domain Entity State Diagram

We use an effect-based view for specifying the service capability. In terms of the ontology, there are several kinds of elementary effects on the real world entities which can be imposed by services. They are: (1) an instance of a real world entity can be created and assigned to a unique identifier; (2) a stateless attribute of an entity/instance can be assigned a value; (3) the state of a stateful entity instance can be changed; and (4) an entity instance can be eliminated.

The effect of a service can be the combination of the elementary effects which may happen on the entity instances. Sometimes the effect can be represented as a compressed combination of effects so that it only contains necessary milestones of the changes of the instances. The feasibility constraints can be given on the effect representation. They are: (1) an instance has to be created before other effects happen on it; (2) when a stateful instance is created, it is in its starting state by default; (3) when a stateful instance enters its ending state, it is eliminated by default; and (4) any stateless attribute can only be assigned one value.

The effect can be formalized as $< Entity, Effect >$ in which *Entity* is a set of entity instances and the *Effect* is a set of effects which include creating/eliminating entity instances, making the stateful entity instances changing their states, and assigning values to stateless attributes.

For example, a possible effect in *Auction* domain is '*JoinAuction*'. It concerns two entity instances, i.e. *Auction[auctionID]* which is already there and *Client[clientID]* which will be created for *Auction[auctionID]*, and makes *Client[clientID]* becoming a *member* (from being a *non-member*) in *Auction[auctionID]*.

4. Process of Service-Oriented Modelling

Our process of SOM consists of four main stages: (a) domain identification and capability modelling; (b) service model construction; (c) model verification, validation and checking; (d) specification generation. The following discuss each stage.

4.1 Domain identification and Capability Modelling

As in the development of all software systems, the first step in the development of a service is the elicitation, clarification and definition of its requirements. For service oriented applications, one of the most important requirements is to determine the capability of the service to be provided, which as argued in the previous section, must be specified in the context of the application domain using a domain ontology. Thus, our first step of SOM is to identify the application domain and to determine the domain ontology to be used, if more than one such ontology exists. In case of such a domain ontology is not available, develop a domain ontology is necessary. Once such a domain ontology is determined, the capability of the service can be specified in the vocabulary of the ontology without ambiguity.

For example, the capability of *Auction Membership Manager* can be represented as follows using the domain ontology of auction.

```
Capability Profile Auction Membership Manager
  Participants: {Client,Auction}
  Effects:
    Effect JoinAuction
      InstanceChange: {+Client[clientID]}
      StateChange:
        {<beInState(Client[clientID],non-member|Auction[auctionID]),
        beInState(Client[clientID], member|Auction[auctionID])>}
    Effect WithdrawAuction
      InstanceChange: {-Client[clientID]}
      StateChange:
        {<beInState(Client[clientID],member|Auction[auctionID]),
        beInState(Client[clientID],non-member|Auction[auctionID])>}
```

Capability modelling is equivalent to the initial functional requirements specification. Non-functional requirements of services such as the quality of services should also be modelled and specified at this stage. However, details on this issue is beyond the scope of this paper.

4.2 Model Construction

In the model construction stage, the functional requirements of a service is further clarified and specified through a set of models on various aspects of the services. In particular, a structural model provides the information about how to service fits into its environment by specifying its intended types of users and the types of other services it relies on. A collaboration model specifies the protocol that the service interacts with its users and other services that it depends on in various interaction scenarios. The behaviour model specifies its internal decision processes, etc.

4.2.1 Structural Modelling

After capability modelling, we get a set of entities and/or their instances which will be operated by the service. However, such operations may be performed through collaboration with other services, which could either be a service requester, or another service provider. Then, structural modelling is to identify relationships among these services.

For example, for the online auction service, we can identify the following participants in the service application system, *Auction service manager* who provide auction services, *Seller* who sells items at the auction and *Buyer* who submit bids and buys the items. The *Seller* and *Buyer* are service requesters and *Auction service provider* obviously are the service provider. The domain objects involved in the service are *bid* and it auction. This leads to the a caste diagram in CAMLE language [8] shown in Figure 6.

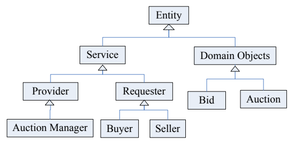

Figure 6. Structure Hierarchy Diagram

4.2.2 Collaboration Modelling

Collaboration model is specifying the interactions between the participants of the service. We use scenario analysis as the basis for developing the collaboration model.

A scenario is a typical situation in the interaction between certain participants. It represents a typical service usage. A collaboration model refers to the structural model for the participants and domain objects involved in the interaction as well as the domain ontology.

A scenario can be specified as a sequence of interactions among participants. A scenario can be formulated as follows where *Service Name* gives the service in which the scenario happens (indicated by *Scenario Name*). The participants in the interaction are named in *Interactee*. For this scenario to happen, the *Pre-conditions* must be satisfied. The main body of the scenario is a sequence of events, which consists of the action taken and the agent name who takes the action. As a result, interaction must finish with the *Post-conditions* satisfied.

```
Scenario Profile Service Name : Scenario Name
  Interactee: ℙ Autonomous Entity
  Reference: ℙ Domain Object
  Pre-Condition: ℙ Assertions
  Event sequence: ℙ Interactee: Message
  Post-Condition: ℙ Assertions
```

The following is a simplified scenario of *Auction Service* for accepting an auction request in which it only accepts the request from a member of the *Auction*:

```
Scenario Profile Auction Service : Acceptance of sell request
  Interactee: {Auction Manger, Seller[SellerID], Bank[BankID]}
  Reference: {Item[ItemID], Auction[AuctionID]}
  Pre-Condition:
   Seller[SellerID]:beInState(member,-),
     belongTo(Item[ItemID], Seller[SellerID]),
   Item[ItemID]:not(beInState(in-auction,-),
     belongTo(CreditCard[CreditCardNo], Seller[SellerID]),
   CreditCard[CreditCardNo]: beInState(valid(credit),Seller[SellerID]),
   CreditCard[CreditCardNo]: beInState(valid(credit-Payment),Seller[SellerID]
  Event Sequence:
   Seller[SellerID]: ↓RequestAuction(Seller[SellerID],
     Item[ItemID], StartDate, EndDate, MinPrice),
   Bank[BankID]: ↑Transfer(CreditCard[CreditCardNo], Payment),
   Seller[SellerID]: ↑AcceptAuction(Seller[SellerID],
     Item[ItemID], Auction[AuctionID])
  Post-Condition:
    Auction[AuctionID]: beInState(created, -),
    Item[ItemID]: beInState(in-auction, Auction[AuctionID])
```

In the same way, we can specify other scenarios taken part by the auction service provider. For each scenario, a scenario specific collaboration model can be derived. Once a complete set of collaboration diagrams for a service is obtained, a general collaboration diagram depicts all possible interactions between service participants can be produced [8]. Figure 7 shows the general collaboration model of the *Online Auction Service*.

4.2.3 Behaviour Modelling

The next step is to develop a behaviour model for each caste.

A behaviour model defines a set of behaviour rules that the agents of the caste must follow. Each rule specifies the condition in which an action should be taken. These conditions give additional information that cannot be specified in scenario and collaboration diagrams, but they must be consistent with the information contained there. The purpose of behaviour model is making the dynamic behaviour of each agent more accurate.

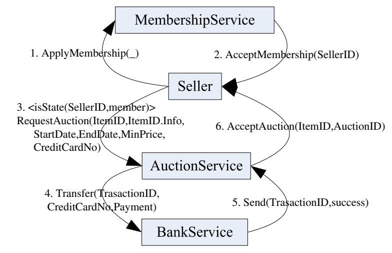
MembershipService
1. ApplyMembership(_)
2. AcceptMembership(SellerID)
Seller
3. <isState(SellerID,member)> RequestAuction(ItemID,ItemID.Info, StartDate,EndDate,MinPrice, CreditCardNo)
6. AcceptAuction(ItemID,AuctionID)
AuctionService
4. Transfer(TrasactionID, CreditCardNo,Payment)
5. Send(TrasactionID,success)
BankService

Figure 7. General Collaboration Model of Auction Service

For example, the behaviour model for *Auction Acceptance* could be figured out as follows.

```
Behaviour Profile Auction Acceptance
  Interactee: {Seller[SellerID], Bank[BankID]}
  Reference: {Item[ItemID], Auction[AuctionID]}
  Behaviours:
    Pre-Condition:
      Seller[SellerID]:inState(member,-),
      belongTo(Item[ItemID], Seller[SellerID]),
      Item[ItemID]:not(inState(in-auction,-)
    Interaction:
      Seller[SellerID]: ↓RequestAuction(Seller[SellerID],
          Item[ItemID], StartDate, EndDate, MinPrice)
    Post-Condition:
    Pre-Condition:
      belongTo(CreditCard[CreditCardNo], Seller[SellerID]),
      CreditCard[CreditCardNo]:inState(valid(credit), Seller[SellerID]),
    Interaction:
      Bank[BankID]: ↑Transfer(CreditCard[CreditCardNo],Payment)
    Post-Condition:
      CreditCard[CreditCardNo]:inState(valid(credit-Payment), Seller[SellerID]),
    Pre-Condition:
    Interaction:
      Seller[SellerID]: ↑AcceptAuction(Seller[SellerID],
          Item[ItemID], Auction[AuctionID])
    Post-Condition:
      Auction[AuctionID]:inState(created, -),
      Item[ItemID]:inState(in-auction, Auction[AuctionID])
```

4.3 Checking Model Consistency and Generation of Formal Specification

Verification, validation and testing play a significant role in SOM. It is difficult because services are normally developed by different vendors. It is impossible to check the consistency via analysis of the whole system structure as in traditional development of software system, because each service provider has only limited access to the information.

However, as demonstrated in [8], when all the models are represented in CAMLE, the consistency of between services can be automatically checked by the CAMLE modelling tool. This consistency checking can be extended to include domain ontology models by explicitly defining a set of consistency constraints on the models in terms of the domain ontology.

Once models in CAMLE passes the consistency check, using CAMLE's specification generator, a service model can be automatically transformed into a formal specification in SLABS. Readers are referred to [12] for details.

5. Conclusions

In this paper, we take a holistic view to service-oriented modelling, which consists of requirements analysis, domain knowledge analysis, model construction with structural, collaborational and behavioural modelling of services, model consistency checking and specification generation. It fits into the holistic view of developing of service oriented application, which include analysis and design, architecture and deployment [2]. The service-oriented modelling framework [13] has been devised for illustrating the service-oriented development life cycle methodology. There have been some efforts to modelling service-oriented applications. Proposals to the description of semantic aspects of service have been advanced in the literatures that rely on ontology for taxonomic descriptions of the functionality of services and workflow descriptions, e.g. WSFL [5] based on Petri Net theory and the Pi-Calculus model with its XLANG [20]. These two approaches are unified in BPML [1]. But most of them are designed for manually modelling not for dynamic and autonomic application modelling which has been recognized as a must-be-addressed challenge [11].

Compared with the existing works, the key features of our approach include: (1) Extending the agent-oriented flavour of our previous work by explicitly capturing the capability, strategy and knowledge of service agents; (2) Domain ontology is used as the background knowledge shared by service agents; and (3) A process has been proposed for conducting the service-oriented modelling, which is obviously urgently needed for developing service-oriented applications.

There are some issues worthy further research. We are investigating the model analysis and the verification and validation of modelled services. More efforts should also be made on the development of domain ontology for capturing the precise meanings of the service capability and other models.

References

[1] A.Arkin. Business process modeling language. http://www.w3.org/TR/2004/WD-ws-cdl-10-20041217/BPML, 2004.

[2] A.Arsanjani. Service-oriented modeling and architecture. http://www-128.ibm.com/developerworks/webservices/library/ws-soa-design1/, 2004.

[3] B.Michael. *Introduction to Service-Oriented Modeling: Service Analysis, Design, and Architecture*. John Wiley and Sons, 2008.

[4] C.M.MacKenzie, K.Laskey, F.McCabe, and et al. OASIS reference model for service oriented architecture 1.0. http://www.oasis-open.org/committees/tc_home.php?wg_abbrev=soa-rm.

[5] F.Leymann. Web services flow language. http://www-306.ibm.com/software/solutions/webservices/pdf/WSFL.pdf, 2001.

[6] H.Zhu. SLABS: A formal specification language for agent-based systems. *International Journal of Software Engineering and Knowledge Engineering*, 11(5):529–558, 2001.

[7] H.Zhu. *Towards an Agent-oriented paradigm of Information systems*, chapter in Handbook of Research on Nature Inspired Computing for Economy and Management, pages 679–691. Idea Group Inc., 2006.

[8] H.Zhu and L.Shan. Agent-oriented modeling and specification of web services. *International Journal of Simulation and Process Modeling*, 3(1&2):26–77, 2007.

[9] K.Levi and A.Arsanjani. A goal-driven approach to enterprise component identification and specification. *Communications of The ACM*, 45(10):45–52, 2002.

[10] K.Sycara and M.Paolucci. *Ontologies in agent architectures*, chapter in Handbook on Ontologies in Information Systems, pages 343–364. Springer-Verlag, 2004.

[11] K.Sycara, M.Paolucci, A.Ankolekar, and et al. Automated discovery, interaction and composition of semantic web services. *Journal of Web Semantics*, 1(1):27–46, 2003.

[12] L.Shan and H.Zhu. CAMLE: a caste-centric agent-oriented modelling language and environment. In *Proc. SELMAS'04 at ICSE'04*, pages 66–73, Edinburgh, UK, 2004. IEE.

[13] M.Bell. *Service-oriented Modeling: Service Analysis, Design and Architecture*. Wiley & Sons, 2008.

[14] M.Endrei, J.Ang, and A.Arsanjani. Patterns: Service oriented architecture and web services. Technical report, IBM, 2004.

[15] M.P.Papazoglou and W. van den Heuvel. Business process development lifecycle methodology: Bridging together the world of business processes and web services. Technical report, University of Tilburg, 2006.

[16] M.Stal. Web services: beyond component-based computing. *Communications of ACM*, 45(10):71–76, 2002.

[17] O.Zimmermann, P.Krogdahl, and C.Gee. Elements of service-oriented analysis and design. Technical report, IBM, 2004.

[18] P.Wang, Z.Jin, and L.Liu. An approach for specifying capability of web services based on environment ontology. In *Proceedings of IEEE International Conference on Web Services, ICWS'06*, pages 365–372, 2006.

[19] P.Wang, Z.Jin, L.Liu, and G.Cai. Building towards capability specifications of web services based on an environment ontology. *IEEE Transactions on Knowledge and Data Engineering*, 20(4):547–561, 2008.

[20] S.Thatte. XLANG-Web services for for business process design. http://www.gotdotnet.com/teams/xml_wsspecs/slang-c/default.htm, 2001.