Semantic Approach for Service Oriented Requirements Modeling

Bin Zhao, Guang-Jun Cai, Zhi Jin

Semantic Approach for Service Oriented Requirements Modeling

Bin Zhao[1,2], Guang-Jun Cai[1,2], Zhi Jin[3]

[1] The Key Laboratory of Intelligent Information Processing, Institute of Computing Technology, Chinese Academy of Sciences, Beijing, China
[2] Graduate University of the Chinese Academy of Sciences, Beijing, China
{zhaobin.ict@gmail.com, caiguangj@mails.gucas.ac.cn}
[3] Software Engineering Institute, Peking University, Beijing, China
zhijin@sei.pku.edu.cn

Abstract. Services computing is an interdisciplinary subject that devotes to bridging the gap between business services and IT services. It is recognized that Requirements Engineering is fundamental in implementing the service oriented architecture. It takes traditional RE techniques great efforts to model business requirements and search for the appropriate services. In this paper, we propose an ontological approach to facilitate the service-oriented modeling framework. The general idea is to establishing a common semantic language to describe both the business requirements and services capabilities based on their effects on the environment. After that, we used a case study to illustrate this method and showed that substantial efforts can be spared to construct a service model from business requirements.

Keywords: knowledge representation, ontology engineering, requirements engineering, SOA, services computing

1 Introduction

Services computing is an interdisciplinary subject that devotes to bridging the gap between business services and IT services. It covers the body of knowledge from business process modeling to IT infrastructure implementation. Services Computing is primarily carried out through web services enabled Service Oriented Architecture (SOA), in which, service requesters invoke local or remote black-box services that are provided by various service providers. SOA can bring flexibility to business solutions by reusing services, and it is the de facto infrastructure for cloud computing. Service orientation as a business and computational paradigm is shifting the software development methodology.

To correctly develop a business solution by utilizing SOA, it is vital to first correctly model the business requirements[1]. As many systems fail due to poorly understood, or ill-conceived, or misinterpreted business requirements[2]. While some researchers tried to adapt traditional RE techniques that are commonly used in OO and component based development to model requirements for SOA, they are labor intensive and error-prone. The first issue is that they missed the point that the SOA paradigm differs greatly from traditional software development paradigms with its emphasis on reuse and business agility[3]; and the SOA solution should be designed for change[4]. The second issue is how to establish mutual understandings between service providers and service requesters; this is more fundamental than the first issue.

There are already works attempt to add semantic sugar to the description of services(e.g. OWL-S, WSDL-S, SAWSDL, etc.); these works carry the SOA forward towards an automatic and intelligent service modeling, discovery, binding, and invocation vision. Ontology in computer science functions as a knowledge representation frame[5], it is commonly used for service description, as the reusable SOA assets are domain specific. In SOA, there are atomic services and composite services. This brings up the granularity problem to service description, and it cannot be effectively handled by existing semantic approaches.

In this paper, we propose a semantic framework for describing the functional business requirements and service functionalities. This is a major improvement to environment ontology since EC4WS was proposed: we describe service functionalities from two aspects, namely, information transformation and state transition. Besides, we also make the first attempt to enable the inference mechanism for environment ontology.

This paper is organized as follows. In section 2, related semantic approaches for services description are introduced. The environment-based service ontology is given in section 3. In section 4, we used a case study to illustrate the service modeling framework. Conclusions and future works are presented in section 5.

2 Related Works

One of the characteristic of SOA is to use heterogeneous black-box services that are published by various vendors. However, web services with the same WSDL description may have dramatically different functionalities. In the academia world, semantic web services is believed to be the remedy for the service ambiguity; techniques such as ontology is recognized as the silver bullet for precise service description.

2.1 OWL-S

OWL-S[6] is the state-of-the-art service ontology, it is widely accepted in both the academia world and industry world. OWL-S utilizes OWL as its core ontology, it describes services from three aspects: service profile, service model and service grounding. The IOPE (Input, Output, Precondition, and Results) description of service profile describes services from two aspects: IO describes information transformation and PE describes state change. OWL-S allows users to use their preferred logical language to express preconditions and effects. The detailed perspective on how to interact with a service is given in *servicemodel* by viewing it as a process. The *grounding* of a service specifies the technical details on how to access a service, e.g. protocol, message format.

There are two features that make OWL-S stand out in a crowd of semantic web service techniques. The first is that it lays it semantic foundation on OWL; it is convenient to use existing OWL inference engines, such as Pellet and Jena. The second is that it allows user to use their preferred logical expression to describe the preconditions and effects of a service process. This enables the flexibility and can be easily adopted as needed. However, this feature can also lead to confusion among users using different logical languages.

2.2 Web Services Capability Description based on Environment Ontology

The main idea underlying this environment-based approach to web service capability description is that services can have effects on their environments, and by portraying environment changes, services capabilities descriptions could be obtained[7][8]. This idea was originally borrowed from Jackson's Problem Frames[9].

The state changes of environment entities are formally represented using hierarchical state machines. Though this makes it straight to model the granularities of services, it cannot effectively handle web services composition problems due to the inherent incomputability of state machines composition.

The ontology of this EC4WS mainly functions as some enumerations of concepts and relationships, it only enables simple inference mechanism, such as subsumption and equivalence. Though EC4WS is still in its infancy, it has the advantage of modeling business requirements in a manner easily understandable by business stakeholders; And we believe that it will be prosperous in the future.

2.3 Service-oriented Modeling and Architecture

SOMA[10] has been used to conduct service modeling in multiple industries around the world since it was proposed in 2004. As a technical guideline that integrates SOA life-cycle management and service-oriented principles, SOMA provides a software engineering method for building end-to-end SOA solutions. The SOMA service development lifecycle includes seven phases; it is a highly structured and regular guideline for carrying out service modeling. In general, SOMA is heavyweight, top-down, model-driven iterative software development method.

SOMA had attempted to use capability patterns and solution templates to help speed up the solution specification process; however, it still heavily relies on the experience and expertise of software engineers. Hopefully, semantic techniques powered domain knowledge can be captured and reused to automate the business modeling and service identification tasks.

3. Environment-based Ontology for Service-oriented Architecture

In the paradigm of services computing, black-box IT Services are usually published and requested by different participants. There exist not only a semantic gap between service providers and service requesters but also a semantic gap between business requirements and IT service capabilities[11]. The notion of environment-base software engineering, proposed by Jackson and Zave[9], is considered to be a "silver bullet" for bridging the above two gaps.

3.1 Principle of Environment-based RE Methodology

The environment of software includes everything but the software itself. According to Michael J., requirements are located in the environment and the software system is to be used within a specific environment. One principle of environment-based RE is that the interactions between a system and its environment are the interfaces between the system and the environment. By observing the shared phenomenon between software and its environment[12], one can indirectly infer the functionalities (capabilities) of the software. In general, appropriate capabilities are required to realize a business

goal, whilst the software exposes some capabilities. Thus, we can use the capability of both ends to bridge the gap between requirements and software functionalities.

Another principle is that while the software may change frequently, its environment stays relatively unchanged. This makes it suitable for requirements engineering for the service-oriented paradigm, because the SOA paradigm is characteristic of loose coupling, reuse, flexibility, and constantly needs to react to business changes. In contrast to that, IT services are frequently subject to extensions and modifications.

3.2 Environment Ontology for Services Computing

IT services have two aspects of functionalities: information transformation or state changes to the environment (Specifically, state changes of some environment entities). For example, to finish a sales order, credit card number is required as input, the status of the transaction is the output information, as a result, the credit card is charged and the ownership of the product transfers from the seller to the buyer[6].

We use ontology to formally model the semantics of a service's functionality, and to cope with its semantic heterogeneities and interface ambiguities. The main concept of environment ontology is *environment entity*. By an environment entity we mean some independent unit of being that is identifiable from the environment[13]. It can be abstract or concrete, such as a message， event， a record in database, a book (conceptually or physically), etc.

Conceptual Model of Environment-based Ontology
This section specifies the ontology model for representing domain specific environment entities. The model is built on-top of OWL2 using the OWL2 datatypes and vocabularies. The OWL2 datatype map is a 6-tuple:

$D ::= < N_{DT}, N_{LS}, N_{FS}, \ \bullet^{DT}, \ \bullet^{LS}, \ \bullet^{FS} >$, where

- N_{DT} is a set of names of datatypes.
- N_{LS} is a function that assigns each datatype $DT \in N_{DT}$ the lexical form of strings.
- N_{FS} is the constraining facet of values of the form (F, v), where F is the constraining facet and v is a value.
- The interpretation function $\bullet^{DT}$ assigns each $DT \in N_{DT}$ a value space.
- The interpretation function $\bullet^{LS}$ assigns each $DT \in N_{DT}$ a lexical form.
- The interpretation function $\bullet^{FS}$ assigns each $DT \in N_{DT}$ a constraint $<F, v>$.

Using the above notion, a datatype *NaturalNumber* has the name "Natural Number, with the lexical form "Integer" and constraint $<minValue, 0>$.

The vocabulary of the environment ontology is a 7-tuple over a datamap D:

$V ::= <V_C, V_I, V_{OP}, V_{DP}, V_{DT}, V_{AN}, V_{LF}>$, where

- V_C is a set of classes that contains at least *owl:Thing and owl:Nothing*
- V_I is a set of individuals used to represent a specific environment entity, individual can be named or anonymous.
- V_{OP} is a set of object properties; this can be used to define the merelogy of environment entity.
- V_{DP} is a set of datatype properties
- V_{DT} is a set of datatypes containing D.

- V_{AN} is a set of annotations that are used to comment purpose.
- V_{LF} is the vocabulary of a logic language that is used to construct the expression of constrants and axioms.

Given the datatype map D and vocabulary V, the environment ontology is defined as a 3-tuple as follows:

$EnvOnt ::= < Entities, Expression, Rel, Axiom >$

- $Entities \subseteq V_C \cup V_I$ is the set of identified environment entities. For a specific domain, the set of entities may vary.
- $Expression \subseteq Entities \times (V_{OP} \cup V_{DP} \cup V_{DT})$, the expression describes the properties of an environment entity.
- Rel is the set of relations between entities, $Rel \subseteq Entities \times Entities$. The details of these relations are given in section 3.2.2.
- $Axiom \subseteq V \times V \cup Entities \times Entities \cup Entities \times Expression$. Axioms are used for inference purpose, e.g., discover new relationships, automatically analyzing the control of the data, and discovering possible inconsistencies.

The environment entities identified are domain specific and have granularities. For example, in a census application, the entity "family" is composed of concrete family members. In contrast, in the domain of healthcare, human beings are view as composed of different part. This also has the characteristic of granularities.

The granularity of an environment entity is captured using the tree structure of ontology concepts. By a *composite entity*, we mean some entity that can be further decomposed into different parts. The constituent entities are called its *sub-entity*. *Atomic entity* is not decomposable. In the hierarchy of entity merelogy, atomic entity lies at the bottom layer. The property of an atomic entity is totally determined by its properties, whilst the property of a composite entity is determined by both its attributes and its merelogy (the way how it is composed).

Axioms and Relation of Environment Ontology

The semantics of the environment ontology is defined by its interpretation. In this section we give the fundamental interpretations of properties, relationships, and axioms. They are subject to extension depending on the purpose of the knowledge engineer and the characteristics of the modeling domain.

The basic *EnvOnt* building blocks are its classes and individuals. The axioms about the relationships between classes are shown in table 1.

Table 1. Class Relationship Expression

Class Axiom	Interpratation	Description
SubClassof (C_1, C_2)	$C_1 \subseteq C_2$	C_1 is the subclass of C_2
EquivalentClass(C_1, C_2)	$C_1 \subseteq C_2 \cap C_1 \supseteq C_2$	C1 *and* C2 are semantically equivalent
ClassIntersection(C_1, C_2)	$\{x \mid x \in C_1 \wedge x \in C_2\}$	It is the class that are the intersection of C_1 *and* C_2
ClassUnion(C_1, C_2)	$C_1 \cup C_2$	It is used *for* concept extension
ComplementClass(C_1, C_2)	$\{x \mid x \in C_1 \wedge x \notin C_2\}$	It is the set of concepts that belong to C_1 but not C_2

For an environment entity, its associations with other entities are specified by the object property axioms and data property axioms. A sample of the typical property axioms are shown in table 2. These are not the complete property axioms due to lack of space.

Table 2. Entity property axioms

Property Axiom	Description
HasA(I_1, I_2)	Used to describe the whole-part relation or specify attributes, se.g., *HasA*(*Person*, *Age*) means *Person* has *Age* attribute.
IsA(*I*, *C*)	Individual *I* is an instantiation of class *C*.
DataProperty(*I*, *DPE*)	*DPE* is the data property expression. Data from *I* must satisfy *DPE*.
Cardinality(*I, DPE*)	For a property specified by *DPE*, constraint its cardinality.
ObjectProperty(I, *OPE*)	Individual *I* satisfies the property *OPE*.
HasAnnotation(*I*, *An*)	Annotation is used to provide further information that is not part of the ontology.

The *state* of an environment entity is the snapshot of its properties. It is a summary of the entity description. The state changes if some of the properties of an individual changes. Following the principle of environment-based RE methodology in section 3.1, the software's functionality can be observed by the state changes of some environment entities.

The environment ontology *EnvOnt* can be used to describe service capabilities in SOA. More detail on how to describe the services will be given in section 3.2.3. It can also be utilized to represent knowledge of an application domain, or to describe resources etc.

Using environment ontology to model service capability

The SOA services are black-box units of functionality that are published by different service providers. The traditional syntactical interface description cannot handle the inherent heterogeneity of service semantics. *EnvOnt* is purposed to handle this heterogeneity by providing the semantic terms that can be used by both ends.

A service accepts some messages, and then performs some functionality accordingly or sends out some messages. This can be categorized to information transformation and state changes of its environment. Formally, we define the capability of services as a 3-tuple:

SerCap ::=< (*Input*, *Output*)?, *Conds*, *StateTrans**>, where

- *Input* is the message that service requires.
- *Output* is the message that service generates as response to *Input*. A service can have zero or exact one IO pairs, this is denoted by the question mark"?". When errors happen, an exception is generated as *Output*.
- *Conds* is the logical expressions that specify conditions under which a service can perform its functionality.
- *StateTrans* describes the semantics of services by means of its effect on the environment entities. *StateTrans* ::={(Δ*Entity.property*)$^+$ }, this means that a service can effect on more than one entities, and causes their properties to change.

The message interchanged is a 4-tuple, as defined below:

Message ::=< *Sender*, *Receiver*, *Content*, *Type* >, where

- *Sender*∈*Entities* and *Receiver*∈*Entities* are the participants in a message exchange. The sender and receiver can be services, users, or other applications.
- *Content* includes the parameters to invoke a service, or the results generated by services.
- *Type* = *one-way* | *broadcast* | *request-response*, this specifies the patterns of message interchange.

Based on the above definition of *message* and *service capability*, a services is described as a 4-tuple,

Serv ::={*Des*, *Process*, *SerCap*, *Groundings*}, where

- *Des* uses natural language and annotations form V_{AN} to describe service profiles, this is used for service discovery.
- *Process* specifies the collaborations between service partners. For a composite service, it specifies how its component services choreograph.
- *SerCap* as defined above is for the service functionality description.
- *Groundings* provides the detail on integrating with existing IT standards.

In reality, there are various web services, e.g., VoIP, VOD (information transformation), online docs (state changes), e-business, etc. Quite often there are services that have the same functionality, e.g., their effects on environment are indistinguishable. We call these services functionally **isomorphism.**

4. Case study: Online Shopping

We use online shopping to demonstrate the environment-based service oriented requirements modeling technique. The modeling process starts by identifying the environment entities. However, one may, with different purposes in mind, look at a system quite differently. So, this is the place where domain knowledge can function as vocabularies that help different users unify their terminology.

In this case, the entities are Customer, PurchaseOrder, Invoice, Manifest, Schedule, Shipping, InvoiceMessage, and POMessage. PurchaseOrder has one Invoice, one or more Manifests, and one or more Schedule as its sub-entity. PruchaseOrder has the property ‘id’ of type string, ‘totalPrice’ of type Integer, and property ‘priority’ of type Integer. As illustrated in Figure 1.

The capability requirements for Purchasing Service is that it can receive the POMessage, calculate the price of Invoice, and schedule the shipping according to PurchaseOrder.Schedule. This changes the PurchaseOrder’s *totalPrice* property to some amount with constrains greater than 0. When the order is processed, the ownership of the manifests is transferred to the customer, and the *status* property of this PurchaseOrder instance is changed to “Processed and Done”.

The example illustrates how to capture the business requirements, and the remaining work is to look up the service repository to discovery available services that expose the required capabilities. This work can be facilitated by the inference mechanism of the environment ontology for service description.

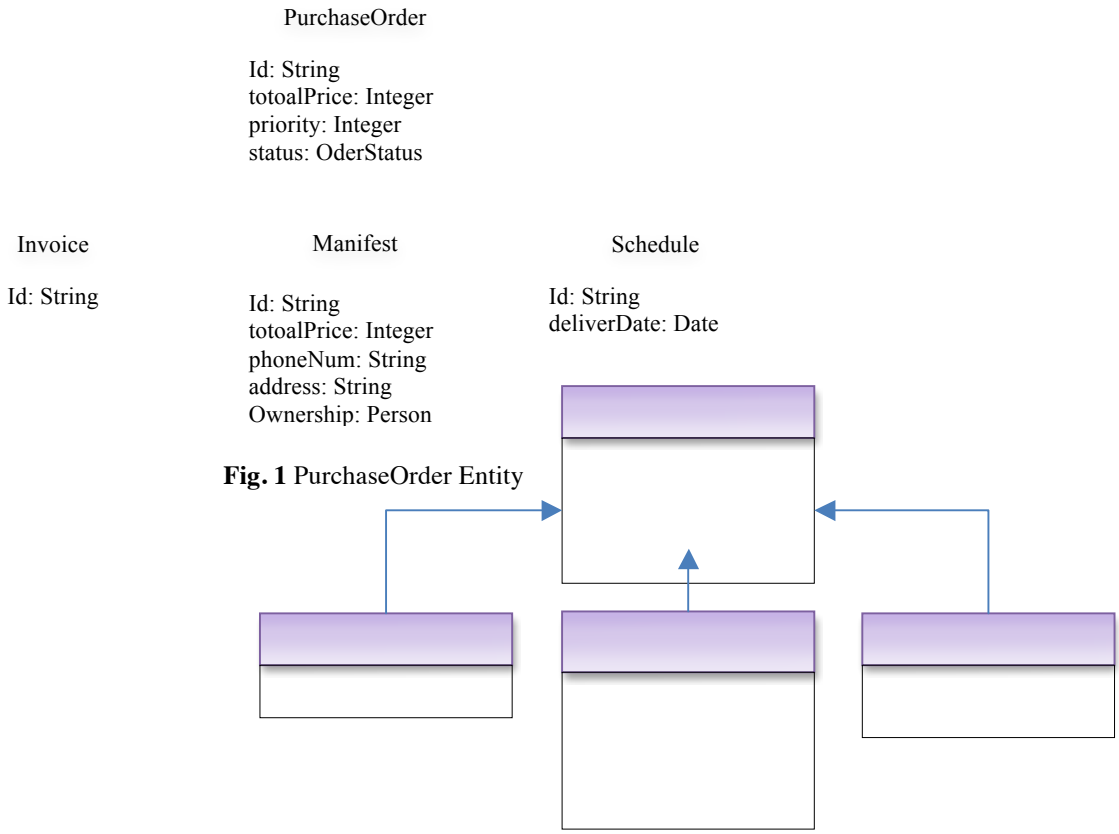

Fig. 1 PurchaseOrder Entity

5. Conclusions and Future Work

Requirements engineering for services computing have different features compared to traditional OO or component-based RE. It is an interdisciplinary subject that still in its infancy. In this paper we proposed an environment-based semantic model for service capability modeling.

This work builds a semantic model based on environment ontology to establishing a common semantic description for both the business service capability requirements and services capabilities. We use an online shopping case study to illustrate how to capture business requirements using the environment-based method. It can be seen that, now we can model the service functionalities using the semantics provided by environment ontology. As a result, substantial efforts can be spared to construct a service model from business requirements.

In the future, we will extend this modeling framework to model the non-functional requirements. The quality of a business solution depends on both its functional requirements and quality requirements.

Acknowledgement

This work was supported by the National Natural Science Foundation for Distinguished Young Scholars of China (Grant No. 60625204), the National Basic Research Program of China (Grant No. 2009CB320701), the Key Projects of National Natural Science Foundation of China (Grant Nos. 90818026, 60736015).

Appreciation also goes to our research team; this work would not have been possible without their generous contribution of our research team.

References

1. Michael B.: Service-oriented modeling: Service Analysis, Design and Architecture. John Wiley & Sons, Inc. (2008)
2. Gary C., Eric N.: The Value of Modeling, A technical discussion of software modeling, http://www.ibm.com/developerworks/rational/library/6007.html
3. Erl T. SOA Principles of Service Design, Prentice Hall (2007)
4. Ian G.: Requirements Modeling and Specifications for Service Oriented Architecture, Wiley (2008)
5. Nicola G.: Formal Ontology, Conceptual Analysis and Knowledge Representation. International Journal of Human-Computer Studies, Vol.43: 625-640, (2005)
6. OWL-S: Semantic Markup for Web Services, www.w3.org/Submission/OWL-S/
7. Tsai W.T., Zhi J., Xiaoying B.: Internetware Computing: Issues and Perspective. Internetware'09, Oct 17-18, Beijing, China (2009)
8. Puwei W., Zhi J, Hongyan L.: Capability Description and Discovery of Internetware Entity. Science China, Vol.53 No.4:685-703 (2010)
9. Pamela Z., Michael J.: Four dark corners of requirements engineering, ACM Transactions on Software Engineering and Methodology, vol.6(1), pp.1-30 (1997)
10. Ali A., Shuvanker G., Abdul A., Tina A., Sella G., Kerrie H.: SOMA: A Method for Developing Service-oriented Solutions, IBM System Journal, vol.47(3), 377--396 (2008)
11. Liang-Jie Z., Jia Z., Hong C.: Services Computing, Springer-Verlag GmbH & Tsinghua University Press (2007)
12. Dines B.: Software Engineering: Abstraction and Modeling, Springer, Heidelberg (2006)
13. Jerome H. S., M.Frans K.: Principles of Computer System Design--An Introduction, Elsevier (2009)