Composition of Web Services using Ontology with Monotonic Inheritance

Chang-yun Li[1,2], Bei-shui Liao[2], and Li-jun Liao[1]

[1]Department of Computer, Zhuzhou Institute of Technology
Zhuzhou 412008, P.R.China

[2]Institute of Computer Software, Zhejiang University
Hangzhou 310027, P.R.China

lcy469@163.com, baiseliao@zju.edu.cn, liaolijun@163.com

Abstract

OWL-S can't provide sufficient support for automatic composition of web services because RDF and OIL don't support overriding declaration in inheritance essentially. An ontology model capable of overriding declaration and expressing virtual class is defined by introducing into a monotonic inheritance mechanism. A method and framework of automatic composition of web services at knowledge level are proposed. The hierarchical matching from abstract to concretion in process ontology solves the task decomposing at high level and goal planning. Three kinds of comprehensive searching strategies are used to seek appropriate service from coarse granularity to thin granularity in turn.

Keywords: semantic web services, ontology, monotonic inheritance, process model.

I. Introduction

Due to their support of flexible connection, and high agility, web services are increasingly becoming a prevailing technology for the integration of distributed and heterogeneous applications [1]. However, the more challenging problem is how to dynamically integrate web services on demand, and especially how to compose them to meet some requirements automatically that are not realized by the existing services. This is the research content of semantic web services [3-10].

The Semantic Web is an extension of the current web in which information is given a well-defined meaning; better enabling computers and people to work in cooperation [11].The well-defined information means to explicitly describe the semantics of web contents, properties, and relations by markup language. Currently, the popular markup language for semantic web is DAML+OIL [12]. Its revision now has been accepted by W3C as an international standard, namely, OWL (Web Ontology Language). The sub-language of OWL, OWL-S [4,5], is used to describe the semantics of web services, i.e., the semantic web services, which support automated service discovery, invocation and composition.

OWL-S describes web services from three aspects: Profile, Model, and Grounding. The Profile gives the functionalities and parameters of web services; the Model, the process model of web services, and specifies control flows and data flows about service descriptions; and the Grounding combines the process model, communication protocols, and the message description together. Among them, the Model is the core. It tells "how the service works", gives the bases for service composing and coordinating. However, the Model of OWL-S does not provide sufficient supports for the automatic service integration. Consider the following two cases:

(1) Mortgage Loan is a special example of Loan. According to the description method of the Model in OWL-S, the process model of Mortgage Loan can inherit from that of Loan. Now, suppose that the activity in Mortgage Loan, Evaluation, needs to extend the homonymic activity of Loan (Evaluation), i.e., the activity of inherited process needs to override the definition of the homonymic activity of the super process. It is not difficult to achieve this purpose in the object-oriented programming language such as java and c++. But there is no corresponding supporting mechanism in OWL-S.

(2)Domain-specific processes, such as Registration of student recruitment in school management services, Registration of new accounts in the bank management system and Registration of new email users in email applications, have the similar decomposition and transition of activity. The Registration can be regarded as a domain-independent abstract process. However, due to big differences among the data types treated by these processes, in the Model of OWL-S these data types are referred to different domain ontology, so that these processes are specified separately and the abstract process Registration is difficult to express.

In the former case, the Model of OWL-S can't provide a flexible mechanism for the classification of the process, while in the latter case, during the service composition, the task decomposition and the goal planning are difficult to be treated with at the knowledge level and by a completely automatic manner. So in OWL-S, the gap between the notions used by human and the data interpreted by machines has not been bridged completely. The reason is that RDF and OIL, based on by OWL, adopt a monotonic inheritance mechanism without the ability of overriding declaration. In addition, every class defined in RDF and OIL can be instantiated to get individuals, while the virtual class similar to an object-oriented method does not exist, so the abstract process such as Registration can't be expressed.

Focusing on the shortcomings mentioned above, we have established a process ontology model by adopting an inheritance mechanism with overriding declaration and monotonic inheritance. This model is the extension of OWL-S. On the basis of this model, we set up a framework for the composition of semantic web services at the knowledge level to further facilitate the automatic composition and invocation of web services.

II. An Ontology Model with Monotonic Inheritance

A. *Ontology Model*

In philosophy, ontology is the systematic description of existing, while in AI, ontology is regarded as an explicit specification of concepts and the relationships among them. The purpose to establish ontology involves knowledge sharing and reasoning. Currently, there are a lot of ontology languages, including Ontolingua, LOOM, OCML, FLogic and OKBC etc. In recent years, there are also some web-based ontology languages such as RDF, XOL and OIL etc. The detailed comparison of these ontology languages is referred to [14]. We define an ontology model by adopting a monotonic inheritance mechanism.

Definition 1: An ontology is defined as a 4-tuple (C, I, P, R).

C — a set of classes that consists of Cr and Cv. Cr denotes common classes that can be instantiated, while Cv denotes virtual classes that can't be instantiated. Each class has a globally unique identifier;

I — a set of individuals. Each individual has an individual identifier;

P — a set of properties. The properties are independent of classes. Each property has a property identifier;

R — a set of relations, R⊂(C∪I∪P)×(C∪I∪P). The main relations include subClassOf, sameClass, hasPropertyOf, sameIndividual, Itype, Ptype, hasValue etc., which are explained in Table 1.

Table 1. main meta-relations

Meta-relation	**Roles**	**Sematics**	**Instance**
subClassOf	C×C	C1⊆C2	Human subClassof Animal
sameClass	C×C	C1≡C2	Handset sameClass Mobile Phone
hasValue	P×I	p=i	Value hasValue RMB 13
sameIndividual	I×I	i1≡i2	Luxun sameIndividual Zhou Shuren
Itype	I×Cr	i∈C1	Newton Itype Human
Ptype	P×C	if (p=i)∧(p Ptype C1) then i∈C1	Value Ptype Currency
hasProperty	C×P	p partyof C1	Human hasProperty Name

In Table 1, hasProperty denotes that Property is a part of Class. It means that in this definition, Property of a class describes not only the general characteristics of Class, but also the components that consist of Class. For example, the class Car has not only the properties such as velocity, load, etc., but the property--- Engine. And yet in RDF, the relationship between Car and Engine is expressed by a container. In addition, the type of property (PType) can be referred to other classes. For instance, the type of property Engine is referred to the class Engine. In this way, the relationships between domain concepts are established.

There are some other meta-relations, such as differentIndividualFrom, inverseOf, transitiveProperty etc. Now, we focus on a further discussion about two forms of subClassOf.

Definition 2: Suppose c1,c2∈C，if(c1 subClassOf c2) and (∃p.((c1 hasProperty p)∧¬(c2 hasProperty p))),where p∈P, it is called that c1 incrementally inherits from c2 ,which is denoted as c1 AsubClassOf c2.

Definition 3: Suppose c1,c2,c3,c4∈C， If(c1 subClassOf c2)and((∃p.(c2 hasProperty p))∧(c1.p Ptype c3)∧(c2.p Ptype c4))，it is called that c1 extendedly inherits from c2. it is denoted as c1 EsubClassOf c2, where p∈P，c3 subClassOf c4，i.e.,c4 is the supper class of c3.

In definition 2, the inherited class has additional properties that do not exist in its super class. In this case, it is the proper subset of its super class, i.e., c1⊂c2. This type of inheritance is supported by RDF and OIL. In definition 3, the declaration of the inherited class overrides the properties inherited from its super class. Since c4 is the super class of c3, in Table 1, c1 subClassOf c2 still satisfies the semantic c1⊆c2. This inheritance overriding declaration is different from the general one, in that inherited property does not conflict with homonymic property of its super class. On the contrary, there is a compatible relationship between them. So, this is a monotonic inheritance.

B. *Process Ontology Model*

Process ontology describes the model of domain-specific processes in the form of declaration. It provides sharing knowledge about the business process for service discovery, execution and composition. Based on the ontology model described in the previous section, our process ontology model is defined as follows.

Definition 4: Process ontology is a 4-tuple (pC, pI, pP, pR), where

pC — a set of classes of business processes;

PI — a set of process instances;

pP — a set of process properties, including purposes, tasks, categories and performances of a process. According to the definition of our ontology, the components of a class are regarded as

its properties, so the inputs and the outputs of a process, other contained processes, and the set of transitions are properties of the process. The list of inputs and outputs, and the set of preconditions, effects, other contained processes , and transitions are signed as inputs, outputs, pres, effects, subprocs and trans respectively;
pR — a set of relations, including inheritance relationships between process classes, instances of processes, and is-a relationships between process classes.
The pres, inputs, outputs, effects and trans of a process all can be referred to other domain ontology. According to the definition of virtual class (in Definition 1), if Ptype of a process class is a virtual class, then this process class is also a virtual class.
In terms of Definition 2, we get Definition 5 as follows.
Definition 5: Suppose that c1,c2 are process classes，if(c1 subClassOf c2) and (∃p.((c1 hasProperty p)∧¬(c2 hasProperty p))), where p∈subprocs∪trans，then it is called that c1 inherits from c2 in the form of process increment, which denoted as c1 pAsubClassOf c2. This definition means that c1 contains not only all the properties of c2, but the sub-processes or the transitions that do not exist in c2.
As for the extended inheritance of the process class, there are two following forms.
Definition 6: Suppose c1、c2∈pC, c3、c4∈C，If(c1 subClassOf c2)and((∃p.(c2 hasProperty p)) ∧(c1.p Ptype c3)∧(c2.p Ptype c4))，in which p∈inputs∪outs∪pres∪effects，c3 subClassOf c4, it is called that c1 inherits from c2 in the form of parameter extension，denoted as c1 pEsubClassOf c2.
Definition 7: Suppose c1、c2、c3、c4∈pC，if(c1 subClassOf c2)and((∃p.(c2 hasProperty p)) ∧(c1.p Ptype c3)∧(c2.p Ptype c4))，it is called that c1 inherits from c2 in the form of sub-process extension. It is denoted as c1 sEsubClassOf c2, where p∈subprocs，c3 is a complex process，c4 is a simple process, and c3 is extended from c4.
Definition 6 explains the action that the inherited process class embodies abstract parameters and types of variant that are referred by a super process class. On the contrary, Definition 7 explains the action that the inherited process class extends the simple processes contained in a super process class, to a complex process.
At the same time, inheritance in the form of process increment is supported by OWL-S, although these two kinds of inheritance as overriding declarations of sub process, which are not supported by OWL-S. Different from common inheritance overriding declaration, the property of derivation class will not contradict with the mark property of super class. It is a compatible relationship. And this means c1⊆c2 in subClassOf c2 still can be satisfied. As a result, it is a kind of monotonic inheritance. In the perspective of the classification mechanism, our ontology model contains OWL-S, so it can be interpreted by our ontology. Examples for the three forms of inheritances mentioned above are shown in Figure 1.
In Figure A, a new sub-class is added to the inherited process class (denoted by a grey rectangle). In Figure B, the simple sub-class Evaluation of the super process class is extended to a complex sub-process of the inherited process class (this inherited process class denotes the mortgage loan of bank possessions). In Figure C, the super process class is an abstract process, Registration. The Ptype of its inputs, registration information, About the Ptype of its inputs, registration information, it refers to a virtual class, registration information. The inherited process class is specialized to the process registration of student recruitment, while it’s Ptype of input about the Ptype of input, it refers to a concrete class, student information.

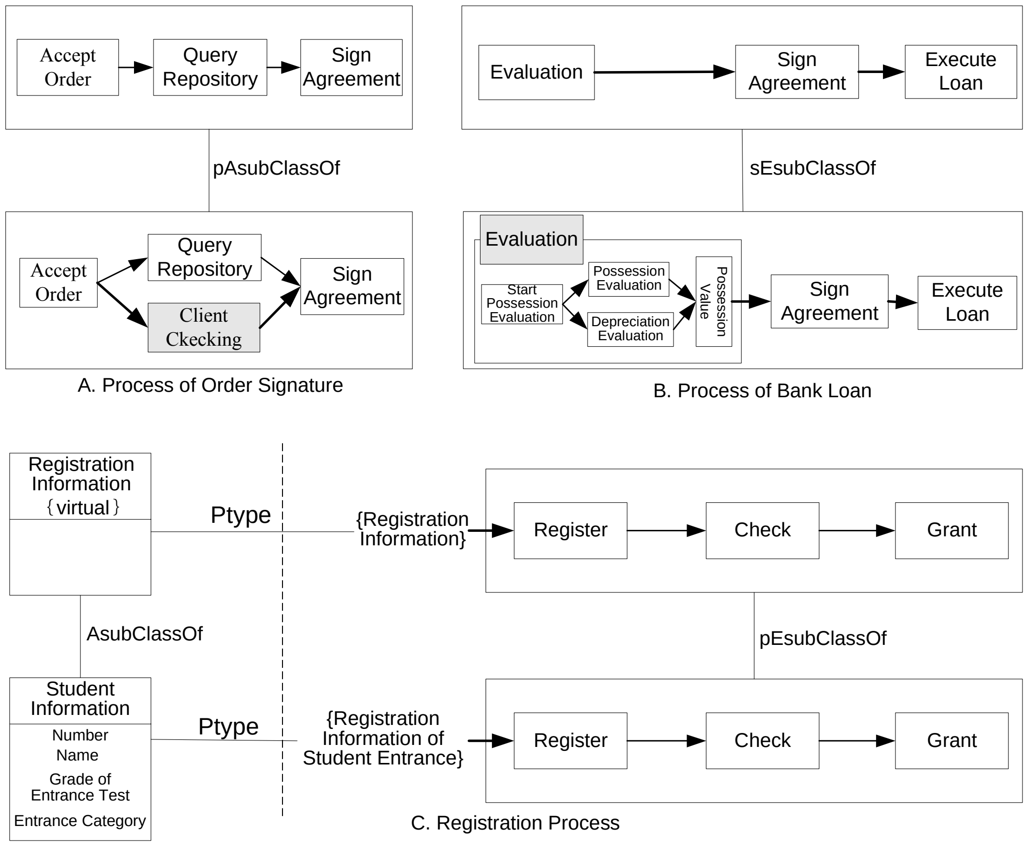

Fig. 1. Three forms of inheritances of process ontology

III. Ontology-based Integration of Web Services

A. *Integration framework*

To realize the automatic, on-demand integration of web services, it is necessary to resolve such issues as the high-level task decomposition and the goal planning at the concept level. We propose an integration framework for web services as shown in Figure 2.

In this figure, user's applications (Application) are problems and tasks the user needs to resolve. The Process ontology describes how the user's task is decomposed into a series of activities, and how a process forms through the transitions among activities. It is oriented to users and problems for high-level goal planning and task decomposing. The solution of the user's task may be cross-domain, so process ontology should be able to express the abstract, application-independent solving process of problems. In order to reduce the complexity of task decomposing, a layered structure is adopted.

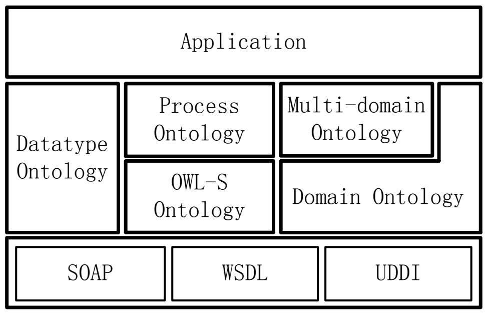

Fig. 2. Integration framework for web services

Meanwhile, the description of a process is independent of web services. The process ontology model mentioned in the section 2.2 can meet this need. In addition, the expression of process ontology needs to be referred to data type ontology, domain ontology and multi-domain ontology.

The Data type ontology contains the basic data types of the concept properties of other ontologies, such as integer, numerical value, and string etc. The XSD defined in XML Schema can be used to describe the data type ontology. The domain ontology defines entities of a specific application domain, and the relationships among them. It has a universe of discourse, i.e., a set of objects to be expressed and treated with. The vocabulary of the knowledge of an application domain is composed of the objects (concepts and individuals) and the describable relations among them. The multi-domain ontology defines a classification for the value types of parameters in an abstract process. It is established according to the ontology model defined in section II.A. It depends on the existence of the process ontology, and is on the basis of various domain ontologies. From the perspective of process, the multi-domain ontology links several domain ontologies together. So ontologies of different application domains can be comparable.

OWL-S ontology describes the semantics of web services by using OWL-S, and establishes classification architecture of web services. During of the process of solving a task, the simple process can be mapped to the port operation of web services. A series of web services perform this task according to the process control flow and the data flow.

In the lowest layer, there are industry standards of web services, including SOAP, WSDL, and UDDI.

B. *Automatic Integration Process*

Based on the integration framework, the integration of web services begins with the task description. Through a parsing performed by a machine, the task description is transformed into a normative task specification. Then, the task specification is mapped to executable processes by a classified matching of the process ontology. And then, those web services that can fulfill the simple process are selected according to the semantic descriptions of web services. Finally, web services are executed. From abstract to concretion, and from concepts to realization, this integration process has an explicit hierarchy, as shown in figure 3.

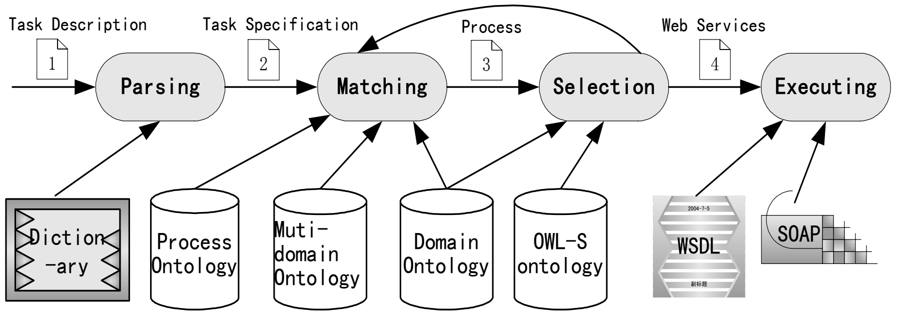

Fig. 3. Automatic integration process of web services

In this figure, the task description is described by using a noun with an attribute, and the parsing of the task description is achieved through automatic word segmentation. There are many methods for automatic word segmentation, including mechanical matching, characteristic word repository, constraints matrix, syntax parsing, and comprehending segmentation, etc. Among them, the mechanical matching method, with the characteristics of simplicity, feasibility, and fast rate, has no requirement of any specific Chinese knowledge. On account of the task description using normative syntax (a noun with an attribute), we adopt a dictionarybased, and improved mechanical matching method for word segmentation. About its algorithm, which the paper[15] is referred to. After matching, the results are expressed in the form of a 3-tuple (<Domain>, <Restriction>, <Basic-Task>), then we get task specifications. For example, “loan of possession mortgage of bank” is parsed as <Bank, Possession Mortgage, Loan>.

According to the task specification (<Domain>, <Restriction>, <Basic-Task>), the process matching begins. Firstly, a high-level matching within the process ontology is conducted in terms of <Basic-Task>. Secondly, in the inheritance tree of the process classification, the hierarchical matching from the high-level to the low-level is done by using <Domain> and <Restriction>. Finally, if the process that meets the requirements can’t be located. Then two strategies, tight matching and loose matching , are used. As for the tight matching, a searching in terms of <domain> and <Restriction> is conducted in the domain ontology and the cross-domain ontology at first. Then the classification hierarchy matching of the process is performed by using <inherited domain> and <inherited restriction>. As we know from the definition of an ontology model, <inherited domain> and <inherited restriction> are the specializations of <Domain> and <Restriction>, so they comply with all constraints of <Domain> and <Restriction>, and their semantics are tighter. On the other hand, as to the loose matching, a searching in terms of <domain> and <Restriction> is conducted in the domain ontology and the cross-domain ontology firstly, to obtain their higher-level <super domain> and <super restriction>. And then the classification hierarchy matching of the process is performed by using <super domain> and <super restriction>. The <super domain> and <super restriction> are the generalizations of <Domain> and <Restriction>, hence, their semantics are looser. So, generally speaking, the tight matching is used first. For example, if “loan of possession mortgage of bank” can’t be located, then “loan of house property mortgage of bank” or “loan of possession mortgage of finance industry” may be located instead. The former is tight matching, and the latter is loose matching. In respect that the virtual class can’t be instantiated, when it is matched, the tight matching strategy will be applied to do searching until an ordinary process class is found.

The simple process should be mapped to the operation of web services. OWL-S describes semantics of web services from three aspects: general information, input and outputs, preconditions and effects, which provide a base for developing an insistent searching mechanism for structural services. The insistent searching depends on the ontology. In terminology classification architecture, if the node corresponding to the terminology A is an ancestor of the node corresponding to the terminology B, B is called semantically consistent with A. The tight matching mentioned above is a kind of consistent searching. Corresponding to these three aspects, we propose three kinds of consistent filtering strategies: (1) classification filtering — is to select consistent and available services (called candidate services) from the service ontology, according to the domain classification of requested services in general information and properties of the simple process. (2) parameter filtering — is to select parameter consistent services from the candidate services (called loosely selected services) in accordance with the inputs and outputs described by the simple process; (3) constraint filtering — is to select constraint consistent services (called accurately selected services) from loosely selected services, in terms of preconditions, effects and resource constraints of the simple process. The selections from classification filtering to parameter filtering, and the accurate selection by constraint filtering are conducted from coarse-granularity to fine-granularity .So, the selection efficiency is higher.

If the requested services of the simple process can’t be found in the Selection stage, it will return to the Matching stage, in which a certain inherited process class in the form of process extension may be located, so that this simple process is extended to a composite process. The composite process contains finer-granularity and lower-level simple processes. With these simple processes as objective activities, a searching is conducted to obtain related services described in the OWL-S ontology.

Finally, web services are invoked via SOAP and WSDL. The Grounding of OWL-S binds the semantic web services with WSDL descriptions. When web services are executed, the parameters are transferred in the form of WSDL message. Meanwhile, the communication between service provider and consumer is via SOAP. The details of SOAP and WSDL are referred to [1].

IV. Related Work and Conclusions

Literatures [7] and [16] propose a non-monotonic inheritance mechanism as the basis of the process classification. Based on the Process Handbook developed by the MIT Center for Coordination Science (CCS), [7] focuses on the flexible process reuse, but it has no strict definitions of the ontology model. And [16] aims at facilitating process designing and dynamic modification, but it does not take the integration of web services into considerations. Both inheritance mechanisms in these two papers are unrestricted non-monotonic inheritance. They support the inherited class to do overriding declaration of properties over its super class, and allows the semantics conflicts between the inherited class and its super-class. On the contrary, our inheritance mechanism is monotonic, and is defined formally. In this mechanism, the overriding declaration of the inherited class should maintain the semantics of the properties of its super-class, so the containing relationship between the super-class and the inherited class are guaranteed. This also lays a theoretical foundation for tight matching, loose matching and consistent searching of web services during the integration process of web services.

In addition, Narayanan and McIlraith [8] have defined he semantics of a subset of the OWL-S specification with the first order logic language. Based on this work, they use Petri Net to formally describe the process model of services and the analysis of service integration, simulation and composition. However, this method is difficult to use and realize.

And besides, there are many researches on the semantic web services-based integration by directly using the process model and the reasoning ability of OWL-S or DAML-S [9]. These methods have the advantages of describing and integrating web services from one perspective and adopting a unified language. However, as mentioned above, the disadvantages are obvious, i.e., due to the direct connections between service operations and specific parameters, and the bindings from process structures to specific domains. Besides, the level of process abstraction is so low that it is difficult to perform the high-level task decomposition and integrate. In order to cope with these problems, many methods have been proposed, such as the integration framework of web services based on Problem Solving Method (PSM) [6], the automatic web service composition using AI planning techniques [10], etc. Our method presented in this paper enables the service integration at the knowledge level, with a formal theoretical basis and an explicit hierarchy, and is understandable and highly automatic. Currently, we have developed a prototype system to test the integration framework and the methods proposed in this paper.

Our future researches will be done on continually improving the ontology model described in this paper, especially on its reasoning mechanism, the theory of semantic inconsistency checking, issues of integrating with OWL, and the improvement of process matching and the services searching strategies, etc. We believe that the service integration system realized with our integration mechanism will further facilitate the service integration to be more automatic and intelligent.

References

[1] A. Tsalgatidou, T. Pilioura, “An Overview of Standards and Related Technology in Web services”. Distributed and Parallel Databases, Kluwer Academic Publishers, 2002, 12(3), pp. 135–162
[2] F. Curbera, Y. Goland, J. Klein, etc, “Business Process Execution Language for Web Services”. http://dev2dev.bea.com/techtrack/BPEL4WS.jsp, 2003.
[3] A.S.McIlraith, T.CaoSon, Z.Honglei, “Semantic Web Services”. IEEE intelligent systems,2001,16(2), pp. 46–53.
[4] A.Ankolenkar, M.Burstein, etc, “DAML-S: Web Service Description for the Semantic Web”. In Proceedings of the First International Semantic Web Conference, Sardinia, Italy.2002, pp. 348–363.
[5] M.Dean, etc. “OWL-S: Semantic Markup for Web Services”. http://www.daml.org/services/owl-s/1.0/owl-s.pdf, 2004.
[6] A.Gomez-Perez ,R.G.Cabero, “A Framework for Design and Composition of Semantic Web Services”. 2004 AAAI Spring Symposium Series. Palo Alto, California, 2004 .
[7] A.Bernstein, B.Grosof, “Beyond Monotonic Inheritance: Towards Semantic Web Process Ontologies”. Working Paper, University of Zurich, Department of Information Technology, 2003.
[8] S. Narayanan, S. McIlraith. Simulation, “Verification and Automated Composition of Web Services”. In Proceedings of the Eleventh International World Wide Web Conference (WWW-2002), Hawaii, USA. 2002, pp. 77–88.
[9] E.Sirin, J.Hendler, B. Parsia, “Semi Automatic Composition of Web Services using Semantic Descriptions”. In Proceedings of the ICEIS-2003 Workshop on Web Services:Modeling, Architecture and Infrastructure. Angers, France, 2003.
[10] J.Peer, “Towards Automatic Web Service Composition using AI Planning Techniques”. http://sws.mcm.unisg.ch/docs/wsplanning.pdf, 2003.
[11] T. Berners-Lee, J. Hendler, O. Lassila, “The Semantic Web”. Scientific American, 2001,284(5), pp. 34–43.
[12] I.Horrocks, F.van Harmelen, etc, “Reference Description of the DAML+OIL Ontology Markup Language”. http://www.daml.org/2001/03/reference.html, 2001.
[13] M. Dean, G. Schreiber, etc, “OWL Web Ontology Language Reference”. W3C Candidate Recommendation, http://www.w3c.org/TR/owl-ref/, 2003.
[14] O. Corcho, A. Gomez-Perez, “A Roadmap to Ontology Specification Languages”. 12th International Conference on Knowledge Engineering and Knowledge Management, Juan-les-Pins, French Riviera, 2000.
[15] LuoZhengqing, ChenZengwu, WangZebing, HuShangxu, “A review of the study of chinese automatic segmentation”. Journal of Zhejiang University (NaturalScience), 1997, 31(3), pp. 306–312.
[16] G.Greco, A.Guzzo, etc, “An Ontology-Driven Process Modelling Framework”. http://wwwinfo.deis.unical.it/~sacca/Papers/dexa04-submitted.pdf, 2004.

Chang-Yun Li, born in 1971, associate professor. His research interests include software architecture and multi-agent system.

Bei-Shui Liao, born in 1971, Ph.D. candidate. His Current research interests include intelligent agent and multi-agent system.

Li-Jun Liao, born in 1973, Ph.D. candidate. Her research interests include software automation, formal methods and environments.