Journal of Computer Science and Cybernetics, V.28, N.1 (2012), 76–87

FORMALIZING AND CHECKING WEB SERVICE DISCOVERY MODELS USING B

TRUONG NINH THUAN[1], TRINH THANH BINH[2], VU VAN HIEU[2]

[1] *VNU University of Engineering and Technology, 144 Xuan Thuy, Cau Giay, Hanoi*
[2] *Haiphong University, 171 Phan Dang Luu, Kien An, Haiphong*

Tóm tắt. Các phương pháp hình thức đã được ứng dụng thành công để kiểm chứng mô hình kiến trúc hướng dịch vụ tại giai đoạn thiết kế. Bài báo này đề xuất một phương pháp hình thức để đặc tả và kiểm chứng sự tương thích giữa các yêu cầu của dịch vụ web so với khả năng đáp ứng của chúng. Trong đó, các yêu cầu dịch vụ web bao gồm các yêu cầu chức năng và phi chức năng, khả năng đáp ứng của các dịch vụ web biểu diễn bằng OWL-S lần lượt được đặc tả bằng một máy trừu tượng và máy làm mịn B. Sự tương thích giữa yêu cầu và khả năng đáp ứng của dịch vụ web được chứng minh tự động thông qua công cụ hỗ trợ của B.

Abstract. The impact of formal methods in service oriented architecture (SOA) has been well-known because of its capability for verification and early analysis of the feasibility. This paper proposes an approach to formalize and analyze the conformance between requirements of web service requester desired and capabilities of web service provided. The requirements of service requester, including functional and non-functional properties, are modeled by a B abstract machine and capabilities of service provider, described by OWL-S, are formalized by a B refinement machine. The matching between these two sides of web service discovery models is analyzed by B support tools.

Keywords: *web service, OWL-S, B.*

1. INTRODUCTION

Web service is a program accessible that are published to the network for use by other programs. Examples of web services are stock quoters, plane reservations, weather services, etc. made over the Internet. A Web service can be regarded as a "programmatic interface" that makes application to application communication possible. An infrastructure that allows users to discover, deploy, synthesize and compose services automatically is needed in order to make Web services more practical.

Web service discovery (WSD) is the process of finding a suitable Web service for a given task. In order that a consumer can use a service, providers usually augment a Web service endpoint with an interface description using the Web Services Description Language (WSDL) [1]. A provider can explicitly register a service with a Web services registry such as UDDI or publish additional documents intended to facilitate discovery such as Web Services Inspection Language (WSIL) documents [2]. The service users or consumers need to search Web services

*This work is partly supported by the research project No. QG.11.32 granted by Vietnam National University, Hanoi.

manually or automatically. The implementation of UDDI servers and WSIL engines should provide simple search APIs or web-based GUI to help find Web services.

With the growing of Web services as a business solution to enterprise application integration, the quality of service (QoS) offered by Web services is becoming the high priority for service provider and their partners. Due to the quality of service providers and unpredictable nature of the Web environment, choosing a suitable QoS is really a challenging task of service requesters. The QoS of Web services mainly refer to the quality, both functional as well as non-functional, aspect of a Web service. This includes performance, reliability, integrity, accessibility, availability, interoperability, and security. The intergration of QoS in web service discovery has been performed at deployment phase [16].

However, service providers and requesters usually have different perspectives and different knowledges about the same service. It is unrealistic to expect advertisements and requests to be equivalent. This issue raises difficulties for finding a suitable service providers in web service applications.

In the analysis phase of software development, simulating and checking software models play an important role. This activity allows us to detect errors in the early stage of software development and consider the feasibility of the system before deploying it. Actually, formal methods [10] have been advocated as an efficient means to perform this activity thanks to their powerful analysis techniques and support tools.

The refinement of the B formal method is a special mechanism which is used to transform an abstract specification step by step into more concrete ones. It is thus appropriate and reasonable to use the B notation to formalize web service requester desired and capabilities of web service provider. In addition, the proof obligations for B specifications are generated and proved automatically (or manually) by support tools like B-Toolkit [12] and B4free [7]. These ones allow us to analyse the matching between two formalized sides of web service models. This paper proposes an approach to formalize and check WSD models using B refinement, including functional and non-functional aspects. It can be summarised as follows:

- A web service requester gives requirements desired to invoke a web service
- The requirements desired are then formalized by a B abstract machine
- Capabilities of web service provider are expressed by a B refinement machine
- The conformance between requirements desired of web service requester and capabilities of web service provider will be analysed by B support tools.

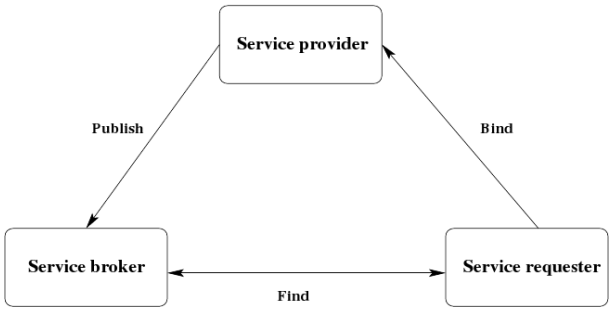
The rest of this paper is organized as follows. Section 2 gives an overview about web services and their QoS. Section 4 prsents an approach to formalize and check WSD models using B notation. Section 5 illustrates the approach by a case study, Tax calculation web service. Section 6 discusses some related works. We conclude the paper and give some future works in Section 7.

2. WEB SERVICE AND THEIR QOS

In this section, we give an overview of web services: the notion and the architecture. We then present their QoS, an important property in WSD models.

2.1. Web services

Web services are self-contained, self-describing, modular applications that can be published, located, and invoked across the Web. Web services perform functions, which can be anything from simple requests to complicated business processes. Once a Web service is deployed, other applications (and other Web services) can discover and invoke the deployed service [3]. The architecture of Web service is the logical evolution of object-oriented analysis and design, and the logical evolution of components geared towards the architecture, design, implementation, and deployment of e-business solutions. Both approaches have been proven in dealing with the complexity of large systems. As in object-oriented systems, some of the fundamental concepts in Web services are encapsulation, message passing, dynamic binding, and service description and querying.



Hình 2.1. Publish, find, and bind in web services

A Web services architecture requires three fundamental operations: publish, find, and bind. Service providers publish services to a service broker. Service requesters find provided services using a service broker and bind to them (Figure 2.1). A transaction in a web services involves three parties: the service requesters, the service provider, and infrastructure components. The service requester, which may broadly identify with the buyer, seeks a service to complete its work; the service provider, which can be broadly identified with the seller, provides a service sought by the requester. A description of a desired service is called a capability request. In web service development, it is the process by which a client applies a declarative description of a service to request something of the service and interpret the response.

2.2. QoS of web services

In order to be viable for e-business, Web services should possess the same QoS that business applications in an enterprise possess such that performance, reliability, availability, security, interoperability, etc.

*P*erformance. Performance is the quality aspect of Web service, which is measured in terms of throughput and latency. Higher throughput and lower latency values represent good performance of a Web service. *T*hroughput represents the number of Web service requests served at a given time period. *L*atency is the round-trip time between sending a request and receiving the response.

*R*eliability. Reliability is the overall measure of a Web service to maintain its service quality. The number of failures per day, week, month, or year represents an overall measure

of reliability for a Web Service. Reliability also refers to the assured and ordered delivery for messages being sent and received by service requesters and service providers.

*A*vailability. Availability is the quality aspect of whether the Web service is present or ready for immediate use. Availability represents the probability that a service is available. Larger values represent that the service is always ready to use while smaller values indicate unpredictability of whether the service will be available at a particular time.

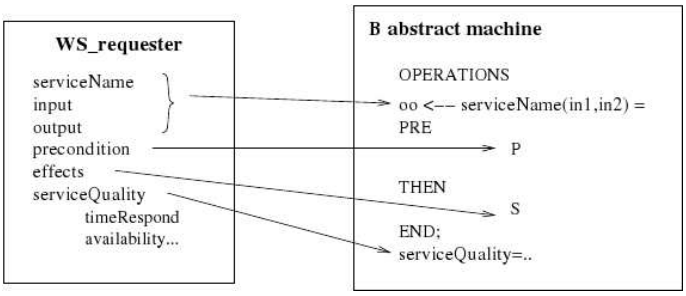
*S*ecurity. Security is the quality of providing confidentiality and non-repudiation by authenticating the parties involved, encrypting messages, and providing access control. Security has added importance because Web service invocation occurs over the Internet. The service provider can have different approaches and levels of providing security depending on the service requester.

3. FORMALIZING AND CHECKING WEB SERVICE DISCOVERY MODELS

From the similarity in web service discovery models and the B refinement mechanism. We propose in this section an approach to formalize web service discovery models using B notation. Particularly, this formalization allows us to prove automatically the model by using B provers.

3.1. Formalizing web service discovery models

The web service discovery process requires a language such that OWL-S [4], which can be used to encode Web service capabilities for advertisement and for requests. Furthermore, discovery requires a matching process that compares the advertisements with the requests to verify whether they describe matching capabilities. An advertisement matches a request when all the inputs, outputs of the request are matched by the inputs, outputs of the advertisement, respectively. Additionally, precondition, postcondition and other QoS of service advertisement need to match the ones of service requester. Note that, OWL-S supports both views, requester and provider, of the capabilities of Web services. Additionally, OWL-S is based on formal semantics, it can fully represents the inputs, outputs and QoS of the service. As a result, we use OWL-S to express web service semantics in our work.



Hình 3.2. Formalizing requester requirements by a B abstract machine

Figure 3.2 illustrates the transformation of web service requester requirements to a B abstract machine. The name of service, input and output desired are transformed to the name

of operation, input parameters and output parameters, respectively. The precondition and effects of web service are formalized by precondition and body of the operation in the B abstract machine. The others QoS of web service desired are formalized in an operation called *s*erviceQuality.

An OWL-S Profile (has also been called service capability advertisement) describes a service as a function of three basic types of information: what organization provides the service, what function the service computes, and a host of features that specify characteristics of the service. The three pieces of information are reviewed in order below.

Firstly, the provider information consists of contact information that refers to the entity that provides the service. For instance, contact information may refer to the maintenance operator that is responsible for running the service, or to a customer representative that may provide additional information about the service.

Secondly, the functional description of the service is expressed in terms of the transformation produced by the service. Specifically, it specifies the inputs required by the service, the outputs generated, precondition required by the service and the expected effects that result from the execution of the service.

Finally, the profile allows the description of a host of properties that are used to describe features of the service. The first type of information specifies the category of a given service. The second type of information is quality rating of the service: some services may be very good, reliable, and quick to respond; others may be unreliable, sluggish, or even malevolent. Before using a service, a requester may want to check what kind of service it is dealing with; therefore, a service may want to publish its rating within a specified rating system.

From the description of OWL-S Profile, we propose to formalize them using a B refinement machine as shown in Figure 3.3. An input specifies the information that the service requires to proceed with the computation. For example, a book-selling service could require the credit card number and bibliographical information of the book to sell. The input is described in OWL-S as follows.

```
<owl:ObjectProperty rdf:ID="hasInput">
  <rdfs:subPropertyOf rdf:resource="#hasParameter"/>
  <rdfs:range rdf:resource="#Input"/>
</owl:ObjectProperty>
```

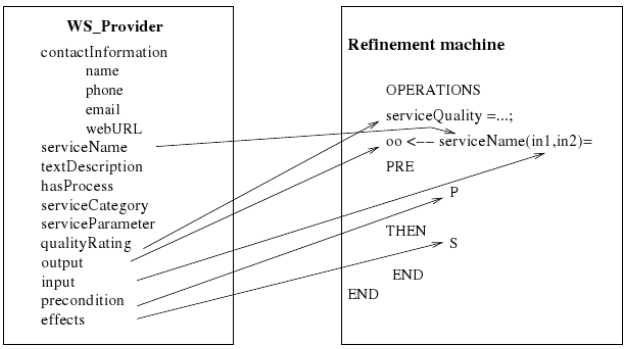
An output is a type of parameter and is a property of a profile. An output specifies what is the result of the operation of the service, it is expressed:

```
<owl:ObjectProperty rdf:ID="hasOutput">
  <rdfs:subPropertyOf rdf:resource="#hasParameter"/>
  <rdfs:range rdf:resource="#Output"/>
</owl:ObjectProperty>
```

The inputs and outputs in OWL-S Profile are reprensented by input and output parameters of the operation in the B refinement machine, respectively (Figure 3.3).

Precondition represents a condition that is expected for the service to work appropriately. For example, a precondition is that some money is available to pay for some goods, or that the goods are available and so on. The precondition is described in the form as follows.

```
<owl:ObjectProperty rdf:ID="hasPrecondition">
  <rdfs:domain rdf:resource="#Process"/>
  <rdfs:range rdf:resource="&expr;#Condition"/>
</owl:ObjectProperty>
```



Hình 3.3. Formalizing provider capabilities by a B refinement machine

Effect represents conditions that are expected to result after the execution of the service. For example, a credit card is charged, goods are delivered as so on. The description of effect is as the following.

```
<owl:ObjectProperty rdf:ID="hasEffect">
  <rdfs:label>hasEffect</rdfs:label>
  <rdfs:domain rdf:resource="#Result"/>
  <rdfs:range rdf:resource="&expr;#Expression"/>
</owl:ObjectProperty>
```

The precondition and effects of the web service are formalized by precondition and body of the operation in the refinement machine (Figure 3.3). The quality rating of a service will be a very important issue in the selection of services. The quality rating has two fields: ratingName and rating: ratingName that specifies the name of the rating service; rating is the rating assigned to the service. An example of rating service is presented as below.

```
<profile:qualityRating>
  <profile:QualityRating rdf:ID="Provider-Rating">
     <profile:ratingName>
          SomeRating
     </profile:ratingName>
     <profile:rating rdf:resource="#GoodsRating"/>
  </profile:QualityRating>
</profile:qualityRating>
```

QoS of web service provider, including the estimation of $maxResponseTime$ and $averageResponseTime$, are formalized in an operation called $s$erviceQuality presented in the refinement machine (Figure 3.3).

3.2. Checking capability matching in WSD model through the B refinement mechanism

In a WSD model, capability matching compares the capabilities provided by any of the advertised services with the capabilities needed by the requester. The goal is to find the advertiser that produces the results required for the requester.

In order to simplify the development of the system, the method in B machines uses the same notation for all the stages of the development, that goes on for successive refinement steps. B refinement provides a way to construct stronger invariants and also to add details in a model. It is also used to transform an abstract model in a more concrete version by modifying the

state description. Increasingly, proof obligations defined in B refinement relation allow us to check that preconditions and postconditions of lower level operations conform to preconditions and postconditions of higher level operations. Recall that, the abstract state variables, xx, and the concrete ones, xx', are linked together by means of a gluing invariant $Inv_{M'}$ defined in the refinement machine.

Formalizing WSD model using B refinement, as a result, enables us to check the identity of the name, input and ouput of web service requester required and service provider. Additionally, as presented in the paper [15], the degree of matching can be differentiated by four levels: exact, plug in, subsumes, fail. Depending on the degree of matching, we can also check the correlativeness of preconditions, effects and QoS between requirements of web service requester and capabilities of the one provider.

4. CASE STUDY

We illustrate our approach by considering a simple system of Tax calculation web service. This one build a web service that can calculate sales tax amounts for goods. With the development of applications processing data based Internet resources, such sales tax services are going to be critical for calculating tax amount of imported goods.

This web service is described as follows. The calling application must provide a valid type as well as quantity of goods. The returned result is the amount of money calculated basing on tax regulations. Below table shows a capability description for the Tax Calculation web service provider.

```
Context:          Tax Calculation;
Types:            Goods = {Tobacco,Wine,Laptop},
                  Money = Real;
Input:            typeOfGoods:Goods;
                  quantity:Integer;
Output:           TaxAmount:Money;
Precondition:     quantity > 0;
Effect:           TaxAmount =
                  taxOf(typeOfGoods)*quantity;
TextDescription:  calculating tax amount for imported goods.
```

On the other hand, the rating system declares that the max response time of the web service provider is 8 ms, the availability is at least 99% and reliability is at least 98% in the 100-degree rating scale system.

Supposing that, in the side of the requester, one requires a web service with the context, types, inputs, outputs, precondition are similar as information described above. The requester does not know the effect of the web service provider, they only require the result of web service is a value of Money type. The also require that, reliability of the web sevice is at least 98%, availability is at least 98% and the max response time is below 10 ms.

4.1. Describing in OWL-S

As introduced in Subsection 3.1, Semantic web services are described using OWL-S (formerly DAML-S) which is an OWL ontology with three interrelated subontologies, known as the profile, process model, and grounding. In brief, the profile is used to express "what a service does", for purposes of advertising, constructing service requests, and matchmaking; the

process model describes "how it works", to enable invocation, enactment, composition, monitoring and recovery; and the grounding maps the constructs of the process model onto detailed specifications of message formats, protocols, and so forth (normally expressed in WSDL).

Because of the similarity of description in OWL-S between web service provider and web service requester, we consider only the description of the Tax Calculation web service provider. We representes inputs, outputs, precondition and effect of the web service in OWL-S Profile and these are then refered to OWL-S Process as the following:

```
<process:AtomicProcess rdf:ID="TaxCalculation">
   <process:hasInput>
      <process:Input rdf:ID="TypeOfGoods"/>
   </process:hasInput>
   <process:hasInput>
      <process:Input rdf:ID="Quantity"/>
   </process:hasInput>
   <process:hasOutput>
      <process:Output rdf:ID="TaxAmount"/>
   </process:hasOutput>
   <process:hasPrecondition>
     <expr:KIF-Condition>
       <expr:expressionBody>
         (and (member ?TypeOfGoods ?Goods)
              (and (member ?Quantity ?Integer)
                   (> (?Quantity 0))))
       </expr:expressionBody>
     </expr:KIF-Condition>
   </process:hasPrecondition>
   <process:hasEffect>
     <expr:KIF-Condition>
        <expr:expressionBody>
           (= ?TaxAmount
              (* (value ?taxOf ?TypeOfGoods)
                 ?Quantity))
        </expr:expressionBody>
     </expr:KIF-Condition>
   </process:hasEffect>
 ...
</process:AtomicProcess>
```

Note that, in this description, the precondition and effect of web service are expressed by Knowledge Interchange Format (KIF) [5]. *Goods* is a set containing all product names in the tax table, *taxOf* is a function, from the *Goods* type to the *Money* type, applied to calculate the tax of goods ruled by government. It is described in OWL-S as follows.

```
  <owl:Class rdf:ID="Goods">
   ...
   <Goods rdf:ID="Tobacco" />
   <Goods rdf:ID="Wine" />
   <Goods rdf:ID="Laptop" />
  </owl:Class>
 <owl:ObjectProperty rdf:ID="taxOf">
  <rdfs:domain rdf:resource="#Goods" />
  <rdfs:range rdf:resource="#Money" />
 </owl:ObjectProperty>
```

4.2. Formalizing and checking in B notation

Applying the transformation rules presented in Section 3, we formalize the description of the Tax Calculation web service provider and requester presented in Figure 4.5. The requester is described by WS_Requester abstract machine and the provider is introduced in WS_Provider refinement machine which refines WS_Requester machine. These machines see (SEES clause) the Types machine which declares sets used in the model (Figure 4.4).

MACHINE Types
SETS $GOODS = \{Tobacco, Alcohol, ...\}$
CONCRETE_CONSTANTS
$MONEY$,
$AVAILABILITY$,
$RELIABILITY, ...$
PROPERTIES
$MONEY \subseteq INT \wedge$
$AVAILABILITY \subseteq NAT \wedge AVAILABILITY = 1..100 \wedge$
$RELIABILITY \subseteq NAT \wedge RELIABILITY = 1..100 \wedge$
...
END

Hình 4.4. Types machine

In this model, the rating system uses 100 levels (as supposed above). Each QoS element of the web service thus gets the value from 1 (lowest) to 100 (highest) (see Figure 4.4). The effect in the method used to calculate the tax amount of requester (*calcul_Tax*) requires merely the return value is belong to the *Money* type [1]. The one in service provider is a formula calculating the tax amount.

The B provers will check if the return value of the formulation in the refinement machine is belong to *Money* type, corresponding with the output value of the method *calcul_Tax* in the abstract machine. This is done based on proof obligations of B refinement. By this way, we can check the correlation of the value of QoS between web service requester and provider, defined in gluing invariant of the refinement machine.

We proved our B model with B4free tool [7], 10 proof obligations are generated for the WS_Provider refinement machine, 80% (8) of them are proved automatically and the rest are proved interactively. Other proof obligations in the model are proved automatically.

5. RELATED WORK

The impact of formal methods in SOA has been confirmed [6]. Actually, however, many of them using model checking for formalizing web service compositions [11, 13]. These approaches focus on formalizing web service communications by state transition system in order to check deadlock or LTL properties.

The paper [14] is aimed at enabling markup and automated reasoning technology to describe, simulate, compose and verify compositions of Web services. It defines the semantics for a relevant subset of DAML-S in terms of a first-order logical language. With the semantics,

[1]The B notation does not support the *Real* type, we thus use *Integer* as a *Money* type

MACHINE WS_Requester
SEES *Types*
VARIABLES
$availabilityR$,
$maxResTimeR, reliabilityR$
...
INVARIANT
$availabilityR \in AVAILABILITY \wedge$
$maxResTimeR \in INT \wedge$
$reliabilityR \in RELIABILITY$
...
INITIALISATION
...
OPERATIONS
$taxAmount \longleftarrow calcul\_Tax($
$typeOfGoods, quantity) =$
PRE
$typeOfGoods \in GOODS \wedge$
$quantity \in INT \ \wedge \ quantity > 0$
THEN
$taxAmount :\in MONEY$
END;

$serviceQuality =$
BEGIN
$availabilityR := 98 \| $
$maxResTimeR := 10 \| $
$reliabilityR := 98 \ldots$
END
END

(a) Web service requester

REFINEMENT WS_Provider
SEES *Types*
REFINES *WS_Requester*
VARIABLES
$taxOf, availabilityP,$
$maxResTimeP, reliabilityP$
...
INVARIANT
$taxOf \in GOODS \longrightarrow MONEY \wedge$
$availabilityP \in AVAILABILITY \wedge$
$maxResTimeP \in INT \wedge$
$reliabilityP \in RELIABILITY \wedge$
$availabilityR \leq availabilityP \wedge$
$maxResTimeR \geq maxResTimeP \wedge$
$reliabilityR \leq reliabilityP$
...
INITIALISATION
...
OPERATIONS
$taxAmount \longleftarrow calcul\_Tax($
$typeOfGoods, quantity) =$
PRE
$typeOfGoods \in GOODS \wedge$
$quantity \in INT \wedge quantity > 0$
THEN
$taxAmount :=$
$taxOf(typeOfGoods) * quantity$
END;

$serviceQuality =$
BEGIN
$availabilityP := 99 \| $
$maxResTimeP := 8 \| $
$reliabilityP := 98...$
END
END

(b) Web service provider

Hình 4.5. Formalizing web service discovery model of Tax system by B

they encode service descriptions in a Petri Net formalism and provide decision procedures for Web service simulation, verification and composition.

Wang *e*t al [17] present an idea about describing, verifying and developing Web Service using the B-method. This work transforms the BPEL4WS documents based on WSDL into the B machine models, and then uses the more rigorous and well-developed verification theory, concept and supported tools to guarantee the validity of Web services and their composition.

The paper [15] proposes an approach based on DAML-S for semantic matching of Web Service capabilities. They show how service capabilities are presented in the Profile section of a DAML-S description and how a semantic match between advertisements and requests is performed. However, this paper consider only the matching between inputs and ouputs of web services but do not consider the conformance of precondition, effect as well as non-functional properties of web services. Another paper concerns the connection between semantic webs and formal methods presented in [9]. This work focuses on building a Semantic Web (RDF/DAML) environment for supporting, extending and integrating many different formalisms.

6. CONCLUSION

Transformation techniques between different languages are useful and important not only for checking Web ontology through software modeling languages and tools, but also for checking and analysing complex software design models [8].

In this paper, we make a link between formal methods and SOA by presenting an approach to formalize and check web service discovery models using B. Based on charateristics of B specification which can be used to model a software system at different levels of abstraction, we formalize requirements of web service requester desired in a high abstraction level and capability of service provided in a lower abstraction level. The different abstraction levels in the B method are developped in a consistent and incremental way and are guaranteed by proof obligations between levels. The matching between two sides of web service discovery models thus can be analyzed by using B provers which prove automatically proof obligations generated.

The contribution of this paper can be used in early analysis of SOA applications by building a B formal model. Alternatively, it can be also applied to check the matching of capabilities between service requesters and service providers by extracting their capabilities. In the future, we will consider to formalize the sending and receiving security aspects into WSD models in order to apply the approach in a high security web service system.

REFERENCES

[1] http://www.w3.org/TR/wsdl.

[2] http://www.ibm.com/developerworks/library/specification/ws-wsilspec/.

[3] http://webservices.xml.com/pub/a/ws/2001/04/04/webservices/index.html.

[4] OWL-S: Semantic Markup for Web Services. http://www.w3.org/Submission/OWL-S/.

[5] http://logic.stanford.edu/kif/kif.html.

[6] L. Bocchi and P. Ciancarini, On the impact of formal methods in the SOA, *Electr. Notes Theor. Comput. Sci.* 160 (2006) 113–126.

[7] Clearsy, B4free, Available at *http://www.b4free.com*, 2004.

[8] J. S. Dong, From semantic web to expressive software specifications: a modeling languages spectrum, *ICSE'06: Proceedings of the 28th International Conference on Software engineering*, ACM (2006) 1063–1064.

[9] J. S. Dong, J. Sun, and H. Wang, Semantic web for extending and linking formalisms, *FME '02: Proceedings of the International Symposium of Formal Methods Europe*, Springer-Verlag, (2002) 587–606.

[10] M. G. Hinchey and J. P. Bowen, *Applications of Formal Methods*, Prentice Hall, 1995.

[11] R. Kazhamiakin, M. Pistore, and L. Santuari, Analysis of communication models in web service compositions, *WWW'06: Proceedings of the 15th international conference on World Wide Web*, ACM (2006) 267–276.

[12] B.-C. Ltd, *B-Toolkit User's Manual*, Oxford (UK), 1996, Release 3.2.

[13] S. Nakajima, Model-checking of safety and security aspects in web service flows, *ICWE'04:International conference on Web Engineering* (2004) 488–501.

[14] S. Narayanan and S. A. McIlraith, Simulation, verification and automated composition of web services, *WWW'02: Proceedings of the 11th International Conference on World Wide Web*, ACM (2002) 77–88.

[15] M. Paolucci, T. Kawamura, T. R. Payne, and K. P. Sycara, Semantic matching of web services capabilities, *ISWC'02: Proceedings of the First International Semantic Web Conference*, Springer-Verlag (2002) 333–347.

[16] S. Ran, A model for web services discovery with QoS, *ACM SIGecom Exchange* **4** 1 (2003) 1–10.

[17] S. Wang, J. Wan, and X. Yang, Describing, verifying and developing web service using the B-method, *NWESP'06: Proceedings of the International Conference on Next Generation Web Services Practices*, IEEE Computer Society (2006) 11–16.

Received on February 2 - 2012