# WSOL – Web Service Offerings Language

Vladimir Tosic, Kruti Patel, and Bernard Pagurek

Department of Systems and Computer Engineering, Carleton University
Ottawa, Ontario, Canada
{vladimir,kpatel,bernie}@sce.carleton.ca

**Abstract.** WSOL (Web Service Offerings Language) is an XML (Extensible Markup Language) notation compatible with the WSDL (Web Services Description Language) standard. While WSDL is used for describing operations provided by Web Services, WSOL enables formal specification of multiple classes of service for one Web Service. A service offering is a formal representation of one class of service for a Web Service. As classes of service for Web Services are determined by combinations of various constraints, WSOL enables formal specification of functional constraints, some QoS (a.k.a., non-functional) constraints, simple access rights (for differentiation of service), price, and relationships with other service offerings of the same Web Service. Describing a Web Service in WSOL, in addition to WSDL, enables selection of a more appropriate Web Service and service offering for particular circumstances. Further, it supports dynamic adaptation and management of Web Service compositions using manipulation of service offerings.

## 1 Introduction

Many leading computing companies have recently announced their Web Service initiatives. These industrial initiatives are accompanied by the corresponding work of industrial standardization bodies, most notably the W3C (World Wide Web Consortium). Hereafter, by the Web Services framework we mean the set of W3C standards for Web Services [1]. The goal of the W3C's Web Services framework is to develop a standard platform, based on already widely used technologies like XML (Extensible Markup Language), for distributed application-to-application (A2A) and business-to-business (B2B) integration [2]. Web Services are envisioned as a mechanism particularly suitable for establishing temporary, ad hoc business relationships.

Although definitions of a Web Service in different industrial initiatives vary somewhat, the common idea is that a Web Service is a unit of business, application, or system functionality that can be accessed over a network by using XML messaging. In principle, a Web Service can provide not only software functionality and data, but also access to some hardware resources like memory, printing, network bandwidth, etc. A Web Service consumes some underlying computing resources and different use of these resources can imply a different price for the Web Service.

Ch. Bussler et al. (Eds.): WES 2002, LNCS 2512, pp. 57–67, 2002.
© Springer-Verlag Berlin Heidelberg 2002

While Web Services can be used for providing services to human end users, the true power of the W3C's Web Services framework is leveraged through compositions of Web Services. This composition can be performed dynamically (i.e., during runtime). Dynamic service composition was the topic of our past research [3], but in the research presented in this paper we assume that Web Services are already composed. Hereafter, by a consumer of a Web Service *A* we assume another Web Service that is composed with *A* and collaborates with it, not an end user (human) using *A*. One Web Service can serve many different consumers, possibly at the same time. The composed Web Services can be distributed over the network, running on different platforms, implemented in different programming languages, and provided by different vendors. The composition provides an added value, either to human end users or for further A2A integration, when a composition of Web Services can itself become a higher-level Web Service. Building complex information systems using Web Services promises to increase system agility, flexibility, and adaptability. However, there are still a number of open issues to be researched and solved and we are exploring some that are not currently addressed by Web Service industrial initiatives and standardization committees.

The W3C's Web Services framework includes several completed and in-progress standards, but for this research the most important one is WSDL (Web Services Description Language) [4]. WSDL is used for describing Web Services in an XML notation. It enables specification of data types (the *type* element), operation signatures (the *message* and *operation* elements), port types (the *portType* element), message format and transport protocol details (the *binding* element), network addresses of different ports (the *port* element), and grouping of different ports into a Web Service (the *service* element). However, WSDL does not enable specification of various constraints on operations and ports in a Web Service.

In this work-in-progress paper, we present our ongoing work on the concept of classes of service provided by a Web Service [5] and formal representation of such classes of service. After a brief introduction to Web Services and WSDL in this section, we explain why classes of service are beneficial for Web Services. Then, we argue for formal specification of classes of service, using formal specification of several types of constraints – functional, QoS (Quality of Service; a.k.a., non-functional), simple access rights, price, and others. We use the term 'service offering' to denote a formal specification of one such class of service. Further, we present our work on WSOL (Web Service Offerings Language), a language we are developing for specification of service offerings for Web Services described in WSDL. Our discussion of the status of the work on WSOL is illustrated with examples. Next, we briefly discuss possible applications of WSOL. At the end, we summarize conclusions and challenges for our future research.

## 2 Service Offerings for Web Services

In certain circumstances, it can be useful to enable a Web Service to offer several different classes of service to consumers. Note that in this paper we discuss classes of

service at the level of Web Services, not at the level of constraints (e.g., reponse time) that are part of the overall service and QoS of the Web Service. Consequently, we define a class of service as a discrete variation of the complete service and QoS provided by one Web Service. Classes of service can differ in usage privileges, service priorities, response times guaranteed to consumers, verbosity of response information, etc. The concept of classes of service also supports different capabilities, rights, and needs of potential consumers of the Web Service, including power and type of devices they execute on. Further, different classes of service may imply different utilization of the underlying hardware and software resources and, consequently, have different prices. Additionally, different classes of service can be used for different payment models, like pay-per-use or subscription-based.

The issues of QoS and balancing of limited underlying resources are particularly motivating for having multiple classes of service for Web Services. If the underlying resources were unlimited, all consumers would always get the highest possible QoS. Unfortunately, this is not the case, so it is suitable to provide different QoS to different classes of consumer. Providers of Web Services want to achieve maximal monetary gain with optimal utilization of resources. Providing different classes of service and their balancing helps in achieving this goal because of the flexibility to accommodate several classes of consumer. On the other hand, consumers of such Web Services can better select service and QoS they need and are willing to pay for, while minimizing their price/performance ratio.

Which classes of service a given Web Service will support is specific to the Web Service. In most cases, developers and providers of a Web Service determine classes of service supported in particular circumstances. In some limited situations, new classes of service for the Web Service can be created dynamically without direct human involvement. Later in this paper, we briefly mention our work on this issue, while more details can be found in [5].

As classes of service of a Web Service can differ in many various aspects, a class of service is determined by a combination of various constraints. We define a service offering as a formal representation of one class of service of one Web Service or one port. Consequently, a service offering is a combination of formal representations of various constraints that determine the corresponding class of service. Service offerings of one Web Service relate to the same characteristics described in the corresponding WSDL file, but differ in constraints that define classes of service. These service offerings are specified separately from the WSDL description of the Web Service. A port-level service offering specifies only constraints upon the constructs in the referred port. A component-level service offering of a Web Service with multiple ports describes an allowed combination of port-level service offerings. If a Web Service has only one port, the component-level service offering is identical to the corresponding port-level service offering.

Let us now illustrate the previous discussion and potential benefits of service offerings with an e-business example. A financial market analysis Web Service consumes one or several stock market notification Web Services and supplies results of its analyses (e.g., recommendations) to different consumers. It can provide its results on request from a consumer (the pay-per-use business model), but a consumer can

also subscribe for periodic reports from the component (the subscription model). Service offerings could accommodate different classes of consumer, for example different classes of consumers that require a slightly different emphasis or depth of the financial analysis. Also, the service offerings could differ in verbosity of results, the rate of unsolicited notification to consumers, in priority of notification of significant market disturbances, in the guaranteed response time, etc. Service offerings would differ in price and play an important role in balancing of the resources used by the Web Service when it processes requests from a large number of different consumers. Examples of these resources are processing power, threads, consumed memory, and used stock market notification Web Services. The resources are limited and their use incurs some costs. For example, the used stock market notification components from other Web Service vendors have to be paid for, probably according to the received level of service and QoS. In fact, the stock market notification components can offer multiple service offerings (differing in the rate of notification, verbosity of provided information, etc.) with different prices. From the point of view of the financial analysis component, choosing which stock market notification Web Services and their service offerings to use is tightly related with the service offerings its own consumers request and with the goal of maximizing the monetary gain for its vendor.

We have conducted a thorough analysis to compare service offerings with relevant alternatives, including parameterization, multiple ports, multiple Web Services, personalization techniques like user profiling, etc. The main advantages of having a relatively limited number of classes of service over other types of service customization are limited complexity of required management and relatively low overhead incurred. For example, we find that personalization techniques aimed at human users can be too complex for customization of simpler Web Services composed with other Web Services. We want to limit the complexity and overhead in order to assure solutions are scalable to large compositions of Web Services. In addition, classes of service are supported by many underlying technologies, e.g., in telecommunications. Our approach is an additional and complementary mechanism for discrete differentiation of service and QoS, not a complete replacement for alternatives. It does not exclude applying in addition other methods for customization of service and QoS, but in the latter case management can be more complex. We are also aware that our approach might not be appropriate for all circumstances, e.g., due to its own overhead.

Note that providing differentiated services and multiple classes of service are well-known concepts in other areas, like telecommunications. Discussion of service differentiation and classes of service in other areas is out of scope of this paper. Our work was particularly influenced by the TINA (Telecommunications Information Networking Architecture) standard [6]. However, such concepts have not been researched and applied in the area of software-based components like Web Services. While we have extrapolated and adapted some existing concepts to address issues relevant for Web Services and their compositions, we are also researching a number of additional issues, like formal specification of service offerings, representation of relationships between service offerings, and dynamic adaptation using manipulation of service offerings.

## 3 Formal Specification of Different Constraints in WSOL

We specify service offerings for Web Services in a comprehensive XML-based notation called WSOL (Web Service Offerings Language). The syntax of WSOL is defined using XML Schema. WSOL is a fully compatible extension of WSDL. While WSDL can (and has to) be extended in several different areas, WSOL extends WSDL only with capabilities directly relevant to the concept of service offerings. WSOL currently enables formal specification of functional constraints (pre- and post-conditions, and invariants), QoS (a.k.a., non-functional) constraints, simple access rights, price (i.e., cost), entities (the Web Service, the consumer, or some trusted third party) responsible for monitoring particular constraints in the service offering, and relationships between service offerings. Constraints in WSOL can be Boolean or arithmetic expressions. Note that access rights in WSOL describe what subset of Web Service's operations a service offering allows to use, i.e., they serve for differentiation of service. Conditions under which particular consumers or classes of consumer may use a service offering are specified and stored outside the WSOL description of a Web Service. QoS constraints describe properties like performance, reliability, availability, etc.

Specifications of different constraints are separated into multiple distinct dimensions to achieve greater flexibility and reusability of specifications. This is a separation-of-concerns issue. However, to support easier choice by consumers, these constraint dimensions are integrated into a service offering. The crucial issue here is that while many constraints are mutually independent and orthogonal, it is not always the case. For example, while 'availability' and 'response time' are orthogonal, 'response time' and 'throughput' are not orthogonal for a given number of operation invocations. Therefore, separation and integration of constraint dimensions is non-trivial. At a minimum, it must capture dependencies between non-orthogonal constraint dimensions. We have developed a solution for this issue in WSOL. This solution will be presented it in a later publication. Another related issue, discussed in [7], is definition of appropriate ontologies for different QoS constraint dimensions. We are still studying some other issues related to separation and integration of constraint dimensions. As these issues are out of scope of this work-in-progress paper, we will discuss them and report our solutions elsewhere.

In Fig. 1, we illustrate WSOL with an example service offering. This service offering *buyStockSO1* is defined for the Web Service *buyStockService* in the *buyStock* namespace. Due to space limits, we do not give definition of this Web Service in WSDL here. It is important to note that this Web Service contains one port type *buyStockPortType*, which contains one operation *buyStockOperation*. This operation is defined with the input message *buyStockRequest* and the output message *buyStockResponse*. The first message contains the following parts: *symbol* (string) that represents a company's stock symbol, *number* (nonNegativeInteger) that represents the number of stocks of the company a consumer wants to buy, and *maxPrice* (float) that represents the maximum price at which a consumer wants to buy a stock. The second message contains one part: *buyStockReply* (float), which represents the amount of money spent in buying the number of stocks that the consumer had requested. The

```
<wsol:offeringType name="buyStockSO1"
 service="buyStock:buyStockService"
 portType="buyStock:buyStockPortType">
  <wsol:postcondition
   operation= "buyStock:buyStockOperation">
    <wsol:comparisonExpression>
      <wsol:arithmeticExpression>
        <wsol:variableName
        vName="buyStock:buyStockReply"/>
      </wsol:arithmeticExpression>
      <wsol:comparator type="&lt;="/>
      <wsol:arithmeticExpression>
        <wsol:variableName vName="buyStock:number"/>
        <wsol:arithmeticOperator type="*"/>
        <wsol:variableName vName="buyStock:maxPrice"/>
      </wsol:arithmeticExpression>
    </wsol:comparisonExpression>
  </wsol:postcondition>
  <wsol:price operation= "buyStock:buyStockOperation">
    <wsol:serviceType
     typeName="costns:regularService"/>
    <wsol:value> 0.005 </wsol:value>
    <wsol:unit unitName="costns:CanadianDollar"/>
  </wsol:price>
  <wsol:accessRights
   operation= "buyStock:buyStockOperation">
    <logicExpression>true</logicExpression>
  </wsol:accessRights>
  <wsol:QoSconstraintList
   operation= "buyStock:buyStockOperation">
    <wsol:QoSconstraint name="MaxResponseTime">
      <wsol:QoSname qName="QoSns:responsetime"/>
      <wsol:QoStype typeName="QoSns:max"/>
      <wsol:qValue> 50 </wsol:qValue>
      <wsol:qUnit unitName="QoSns:ms"/>
    </wsol:QoSconstraint>
  </wsol:QoSconstraintList>
  <wsol:managementResposibility>
    <supplierResponsibility scope="tns:buyStockSO1"/>
    <independentResponsibility
     scope="tns:MaxResponseTime" entity="http://
     www.myManagementCompany.com/myManagerService"/>
  </wsol:managementResponsibility>
  <wsol:relatedSOsList>
    <wsol:relatedSO name="tns:buyStockSO2"
     dimension="tns:MaxResponseTime"/>
  </wsol:relatedSOsList>
</wsol:offeringType>
```

**Fig. 1.** An Example Service Offering in WSOL

presented service offering first defines a postcondition that *buyStockReply* must be less than or equal to the product of *number* and *maxPrice*. Next, the service offering defines the price of using *buyStockOperation* in this service offering. In the given example, if the invocation is successful, the consumer has to pay 0.005 CAN$. Further, the service offering defines a simple access right specifying that consumers using this service offering can use the *buyStockOperation*. After that, the service offering defines a QoS constraint defining *MaxResponseTime*. Note that the meaning of response time, maximum, and ms (milliseconds) is defined in an external ontology (namespace *QoSns*). *The managementResponsibility* tag specifies that an independent external entity (with the given URL) is responsible for monitoring response times for *buyStockOperation* and informing the Web Service and its consumers if it is greater than *MaxResponseTime*. The last constraint in the service offering is specification that this service offering is related to another service offering, *buyStockSO2*, through the *MaxResponseTime* constraint dimension. This means that if *MaxResponseTime* cannot be achieved, the Web Service and a consumer using *buyStockSO1* should negotiate switching to *buyStockSO2* (in some cases, this switch can be performed automatically by the Web Service).

One of the crucial issues in WSOL is how to represent relationships between service offerings. These relationships have to be specified for at least three purposes. The first one is to provide a more straightforward and more flexible specification of new service offerings. This is needed to specify relatively similar service offerings of one Web Service, as well as relatively similar service offerings of similar Web Services. The second purpose is to enable easier selection and negotiation of service offerings. The third purpose is to support dynamic adaptation of Web Service compositions based on the manipulation of service offerings, which we will briefly discuss in the next section. We want to find a mechanism or a coherent combination of mechanisms that best supports all three purposes. We have explored several possible alternatives for representing these relationships. Our current solution is based on constraint dimensions, but a more powerful solution is under development.

In the future, we might extend WSOL with formal specification of dependencies on other Web Services and infrastructure, which consumers should know about for successful use and service composition. Other constraints left for future work include: known relationships that can be formed with other Web Services; potential incompatibilities with other Web Services; alternative or similar Web Services from the same vendor or its business partners; roles that can be played in different patterns and coordination protocols; synchronization/concurrency and sequencing constraints, etc. We are also exploring some possible improvements of the WSOL syntax.

We are working intensively on proof-of-concept prototypes for WSOL tools. Most importantly, development of a prototype WSOL parser with syntax checks and some semantic checks accompanies development of the WSOL grammar rules. Consequently, the previously given examples can be parsed with the current version of this parser. Its implementation is based on the Apache Xerces XML Java parser. Second, we are looking at automatic generation of some constraint-checking code (in Java) from WSDL and WSOL files. This is a complex issue. As constraint dimensions can be viewed as aspects of service offerings, we are exploring use of composition filters

[8] and similar aspect-oriented approaches. We are plan to develop a Java API (Application Programming Interface) for generation of WSOL files. The goals of the prototypes are to check feasibility of the suggested solutions, uncover hidden issues, demonstrate contributions, check whether the adopted decisions are better than possible alternatives, and provide new insights and ideas.

The work on WSOL is strongly influenced by the ideas from [9] and a number of other works on formal specification of particular types of constraints. Further, specification of different types of constraints in XML makes our work also related to [10] and [11], but these papers are not in the area of Web Services and they are not compatible with WSDL. On the other hand, IBM has been working [12] on WSEL (Web Services Endpoint Language). One of the goals of WSEL is to enable specification of some constraints, including QoS, for Web Services described with WSDL. There is no detailed publication on WSEL to date. If WSEL is fully developed before WSOL, we will explore making WSOL compatible with WSEL. Further, the DAML (DARPA Agent Markup Language) initiative includes work on the DAML-S language [13] for semantic description of Web Services, including specification of functional and some QoS constraints. However, DAML-S is not compatible with WSDL. None of these related works enables specification of multiple classes of service for one Web Service and manipulation of these classes of service, which is the main goal of WSOL.

## 4 Applications of WSOL

We believe that as the number of Web Services on the market that offer similar functionality increases, the offered QoS and price/performance ratio, as well as adaptability, will become the main competitive advantages. The comprehensive specification of Web Services and service offerings in WSOL supports selecting appropriate Web Services and service offerings, e.g., in the process of dynamic service composition. Consumers get additional flexibility to better choose service and QoS that they will receive and pay for and minimize thus the price/performance ratio and/or the total cost of received services. Such comprehensive formal specification also helps reduce unexpected interactions between the composed Web Services. The need for such comprehensive formal specification was one of the conclusions of our past project on dynamic service composition [3]. As already noted, dynamic service composition is outside the scope of this work-in-progress paper, as are various issues related to Web Service and service offering discovery, selection, and negotiation.

Composing complex information systems from Web Services, especially during run-time, can significantly increase system agility, flexibility, and adaptability. However, to further increase these qualities, such compositions have to be managed and adapted to various changes, particularly to those changes that cannot be accommodated on lower system levels like communication software, operating system, etc. This management and adaptation should occur while the information system is running, with minimal disruption to its operation and with minimal human involvement. In other words, it should be dynamic and autonomous. We want to achieve management by dynamic adaptation of compositions of Web Services without breaking an

existing relationship between a Web Service and its consumer. This goal differentiates our work from the past work on adaptable software, like the architecture-based approaches based on finding alternative components and rebinding [14]. To achieve this goal we are researching dynamic adaptation capabilities based on manipulation of service offerings. Our dynamic adaptation capabilities include switching between service offerings, deactivation/reactivation of existing service offerings, and creation of new appropriate service offerings [5]. Compared to finding alternative Web Services and rebinding, these dynamic adaptation capabilities enable faster and simpler adaptation and enhance robustness of the relationship between a Web Service and its consumer. They enable Web Services to retain existing consumers and do not require establishment of new trust relationships between Web Services. This is important in many e- and m-business systems. In some cases, finding and selecting an appropriate alternative Web Service can turn out to be too slow and its success cannot always be guaranteed. Further, these capabilities are simple and incur relatively low overhead, while providing additional flexibility. We find our approach particularly advantageous when dynamic adaptation is required relatively frequently and can be achieved with a variation, not a drastic modification, of provided services and QoS. Such circumstances occur in many non-trivial situations, ranging from small temporary disturbances of service and QoS caused by mobility to dynamic evolution of Web Services. However, compared to finding alternative Web Services, our dynamic adaptation capabilities have limitations. Service offerings of one Web Service differ only in constraints, which might not be enough for adaptation. Further, appropriate alternative service offerings cannot always be found or created. Manipulation of service offerings is an additional and complementary approach to dynamic adaptation, not a complete replacement to finding alternative Web Services. Therefore, we suggest a two-level, integrated, approach to dynamic adaptation. The first step is to try to adapt using manipulation of service offerings of the same Web Service. The second step is to try to find an alternative Web Service, if the first step was unsuccessful. In fact, a Web Service can provide a temporary replacement service offering while the consumer searches for another, more appropriate, Web Service.

We are also developing a corresponding management infrastructure, called DAMSC (Dynamically Adaptable and Manageable Service Compositions). Among other issues, DAMSC will enable various manipulations of WSOL descriptions of Web Services. In our work on WSOL we pay particular attention to issues relevant for this intended use of WSOL. One example is specification of relationships between service offerings to support dynamic and automatic switching of service offerings. The detailed discussion of our dynamic adaptation capabilities and the DAMSC infrastructure is outside the scope of this work-in-progress paper. More information can be found in [5].

## 5 Conclusions and Future Work

The issues of classes of service and formal specification of various constraints have not been previously addressed for Web Services. Solving these issues has practical importance for building information systems that use Web Services. Web Services

are envisioned as a means for more agile, flexible, and adaptable development of complex B2B and A2A information systems. Our work on the concept of multiple classes of service for Web Services and WSOL provides some additional flexibility. Consumers get additional flexibility in selecting appropriate Web Services and their levels of service and QoS, while their price/performance ratio is reduced. On the other hand, providers of Web Services have more flexibility in balancing underlying resources, as well as in covering the Web Service market by addressing the needs of diverse consumers. The support that service offerings provide for dynamic adaptation further increases flexibility, adaptability, and agility. Our dynamic adaptation capabilities based on manipulation of classes of service can be a useful complement to finding alternative Web Services, due to their speed, simplicity, low overheads, and enhanced robustness of relationships between Web Services (and thus e-business partners).

Using service offerings has its limitations and it is an additional and complementary mechanism for discrete differentiation of service and QoS, not a complete replacement for alternatives. Its main advantages are limited complexity of required management and relatively low overhead. Consequently, we find that our approach is appropriate in many non-trivial situations.

WSOL currently enables formal specification of functional constraints, some QoS constraints, simple access rights, price, entities responsible for constraint monitoring, and relationships with other service offerings of the same Web Service. In the future, we will make specification of these constraints more powerful and we plan to enable specification of some other constraints. We emphasize here two important open issues. First, we have to work more on separation and integration of constraint dimensions, without conflicts and with straightforward implementation of constraint-checking code. This work includes a study of applicability of aspect-oriented approaches like composition filters. Second, we have to improve specification of relationships between service offerings to support both easier and more flexible specification and dynamic adaptation. In parallel, we are continuing our work on prototypes for WSOL tools. We are also working on using WSOL for dynamic adaptation capabilities in the DAMSC infrastructure, as well as selection and negotiation of service offerings specified in WSOL.

## References

1. International Business Machines Corporation (IBM), Microsoft Corporation: Web Services Framework. In Proc. of the W3C Workshop on Web Services – WSWS'01 (San Jose, USA, Apr. 2001) W3C. On-line at: http://www.w3.org/2001/03/WSWS-popa/paper51
2. Curbera, F., Mukhi, N., Weerawarana, S.: On the Emergence of a Web Services Component Model. In Proc. of the WCOP 2001 workshop at ECOOP 2001 (Budapest, Hungary, June 2001) On-line at: http://www.research.microsoft.com/~cszypers/events/WCOP2001/Curbera.pdf
3. Mennie, D., Pagurek, B.: A Runtime Composite Service Creation and Deployment and Its Applications in Internet Security, E-commerce, and Software Provisioning. In Proc. of the 25th Annual International Computer Software and Applications Conference - COMPSAC 2001 (Chicago, USA, Oct. 2001) IEEE Computer Society Press. 371-376

4. World Wide Web Consortium (W3C): Web Services Description Language (WSDL) 1.1. W3C note. (March 15, 2001) On-line at: http://www.w3.org/TR/wsdl
5. Tosic, V., Pagurek, B., Esfandiari, B., Patel, K.: On the Management of Compositions of Web Services. In Proc. of the OOWS'01 (Object-Oriented Web Services 2001) workshop at OOPSLA 2001 (Tampa, Florida, USA, Oct. 2001) On-line at: http://www.research.ibm.com/people/b/bth/OOWS2001/tosic.pdf
6. Kristiansen L.: (ed.) Service Architecture, Version 5.0. TINA-C (Telecommunications Information Networking Architecture Consortium) specification. (June 16, 1997) On-line: http://www.tinac.com/specifications/documents/sa50-main.pdf
7. Tosic, V., Esfandiari, B., Pagurek, B., Patel, K.: On Requirements for Ontologies in Management of Web Services. In Proc. of the Workshop on Web Services, e-Business, and the Semantic Web: Foundations, Models, Architecture, Engineering and Applications, (Toronto, Canada, May 2002)
8. Bergmans, L., Aksit, M.: Composing Crosscutting Concerns Using Composition Filters. Comm. of the ACM, Vol. 44, No. 10. ACM. (Oct. 2001) 51-57
9. Beugnard, A., Jezequel, J.-M., Plouzeau, N., Watkins, D.: Making Components Contract Aware. Computer, Vol. 32, No. 7. IEEE. (July 1999) 38-45
10. Mckee, P., Marshall, I.: Behavioural Specification using XML. In Proc. of the 7th IEEE Workshop on Future Trends of Distributed Computing Systems - FTDCS'99, (Cape Town, South Africa, Dec. 1999) IEEE Computer Society Press. 53-59
11. Jacobsen, H.-A., Karamer, B. J.: Modeling Interface Definition Language Extensions. In Proc. Technology of Object-Oriented Languages and Systems - TOOLS Pacific 2000 (Sydney, Australia, November 2000) IEEE Computer Society Press. 241-252
12. Ferguson, D. F.: Web Services Architecture: Direction and Position Paper. In Proc. of the W3C Workshop on Web Services – WSWS'01 (San Jose, USA, Apr. 2001) W3C. On-line at: http://www.w3c.org/2001/03/WSWS-popa/paper44
13. The DAML Services Coalition: DAML-S: Semantic Markup for Web Services. WWW page. (December 12, 2001) On-line at: http://www.daml.org/services/daml-s/2001/10/daml-s.html
14. Oreizy, P., Medvidovic, N., Taylor, R. N.: Architecture-Based Software Runtime Evolution. In Proc. of the International Conference on Software Engineering 1998 - ICSE'98 (Kyoto, Japan, Apr. 1998) ACM Press. 177-186