# A Metamodel of the WS-Policy Standard Family

Balazs Simon, Balazs Goldschmidt, Peter Budai, Istvan Hartung, Karoly Kondorosi, Zoltan Laszlo, Peter Risztics
*Department of Control Engineering and Information Technology*
*Budapest University of Technology and Economics*
*Budapest, Hungary*
*Email: {sbalazs | balage | bucnak | hartungi | kondor | laszlo}@iit.bme.hu, risztics@ik.bme.hu*

***Abstract*—The basic building blocks of SOA systems are web services. The domain specific language SOAL developed by the authors has a Java and C#-like syntax for describing web service interfaces. Beside the syntax a metamodel (SoaMM) is also defined. The paper introduces an extended version of both SOAL and SoaMM that supports WS-Policy specifications. The original WS-Policy standards specify huge XML descriptions that are too complex and low level for efficient service design. The metamodel presented provides a high abstraction level that is still strong enough for generating vendor-specific service configurations for WCF and major JAX-WS implementations.**

***Keywords-SOA; web services; WS-policy; modelling; DSL.***

## I. INTRODUCTION

Complex distributed systems are best built from components with well-defined interfaces and a framework that helps connecting them. Web services and BPEL (Business Process Execution Language) processes implementing WSDL (Web Services Description Language) interfaces represent today the components that both enterprises and governments use to construct their complex distributed systems, implementing a Service Oriented Architecture (SOA) [1].

One of the advantages of building on web services technology is that one can get a vast set of standards from simple connections to middleware functionalities such as security or transaction-handling [2].

The other advantage is that SOA applies classical principles of software engineering, like model-based development, loosely coupled components, separation of interface from implementation, etc., at a higher level of abstraction. When implementing the abstract model, vendor specific issues start to dominate. In many situations, like systems in e-government or those of interoperating companies, such vendor-dependent issues undermine successfull systems integration. Without clearly defined, common configuration settings interoperability might be compromized. Therefore it is of utmost importance to have a common model of configuration descriptions that enables vendor independent specification of policies. This was the essential aim of the WS-Policy standards. Unfortunately these standards not only require huge, unreadable, and thus unmaintainable XML configuration descriptions, but in the actual products the descriptions usually still contain vendor specific details.

This paper shows how a high-level web service and BPEL metamodel, description language, and code generating framework was extended with WS-Policy standards support in order to enable developers to automatically generate platform-specific policy configurations from the general, standard-compliant policy specifications. The paper emphasizes the use of a clear and easily extensible metamodel framework that is the *sine-qua-non* of a really useable and effective solution.

The activity of our research group aims at developing principles, recommendations, components and technologies that make the application of SOA in the e-government domain easier. Considering the diversity of legacy systems of governmental organizations, all the solutions we elaborate should be platform-independent as far as it is possible.

The rest of the paper is organized as follows. In the second section, related work is examined. In the third section, the metamodel is introduced, the WS-policy representation is detailed, the code generating framework architecture is shown, and finally, an evaluation is provided. The fourth section summarizes the results.

## II. RELATED WORK

One way of describing services is using semantic web technologies. The major goal of Semantic Web Services (SWS) is to create intelligent software agents to provide automated, interoperable and meaningful coordination of web services [3]. The three main directions of SWS are SAWSDL (Semantic Annotations for WSDL), OWL-S (Semantic Markup for Web Services) and WSMO (Web Services Modeling Ontology).

SAWSDL [4] does not introduce a new language. It is a WSDL extension for referencing ontological concepts outside WSDL documents. Beyond that it does not define any execution semantics for the implementation.

The OWL-S [5] profile ontology is used to describe what a service does, and is meant to be mainly used for the purpose of service discovery. The service description contains input and output parameters, pre- and post-conditions, and also non-functional aspects. The OWL-S process model describes service composition including the communication pattern. In order to connect OWL-S to existing web service standards, OWL-S uses grounding to map service descriptions

 ISBN: 978-1-61208-116-8 

to WSDL. The OWL-S environment provides an editor to develop semantic web services and a matcher to discover services. The OWL-S Virtual Machine is a general purpose web service client for the invocation. OWL-S therefore requires a custom execution environment and cannot be used in current commercial SOA products. Its underlying description logic OWL-DL has also a limited expressiveness in practice.

The WSMO [6] framework provides a conceptual model and a formal language WSML for semantic markup of web services. WSMO is used for modeling of ontologies, goals, web services and mediators. Ontologies provide formal logic-based grounding of information used by other components. Goals represent user desires, i.e., the objectives that a client might have when searching for services. Web services are computational entities, their semantic description includes functional and non-functional properties, as well as their capabilities through pre- and postconditions, assumptions and effects. Mediators provide interoperability between components at data, protocol and process level. The reference implementation of WSMO is the WSMX [7] framework, a custom execution environment. It is designed to allow dynamic discovery, invocation and composition of web services. It also provides interoperability with classical web services.

The main design goals of SWS standards are discovery, invocation and composition of web services. These standards are not primarily designed for modelling purposes. They are weak in terms of security, transactional, reliability and other non-functional aspects even at the conceptual level [8]. Because of their custom execution environment, their implementations do not rely on existing SOA products of major software vendors, which can result in interoperability problems with classical web services published by these products.

Although there are directions to extend SWS standards with WS-Policy concepts [9] [10] [11], these solutions focus on service discovery and policy matching, and do not resolve the issues related to modelling and implementation.

Most BPEL workflow engines in the industry also lack support of the specification and enforcement of non-functional requirements. A. Charfi et.al. [12] proposed and implemented a container framework to include the most important WS-* standards regarding security, reliability and transaction handling in BPEL processes. However, their solution can only be used in the ActiveBPEL engine, and cannot be applied to other industry engines.

Another major drawback of the WS-Policy standard family is that the policy assertions of the WS-* standards can be very large XML structures, which makes them nearly impossible to be handwritten by humans. Luckily, most SOA products provide policy repositories (e.g., Oracle SOA Suite, IBM WebSphere) containing complete assertions that can be used for configuration. However, these assertions may differ between products and matching them can be a difficult task. There are also products, which offer GUI designers (e.g., GlassFish ESB) or transform WS-Policy assertions into their own configuration representation (e.g., Apache CXF, Microsoft Windows Communication Foundation (WCF)) making interoperability problems even harder.

Others focus their research on creating platform independent languages for describing WS-Policy assertions. These efforts come together under the umbrella of XACML [13] (eXtensible Access Control Markup Language). XACML is a declarative access control policy language implemented in XML and a processing model, describing how to interpret the policies. WSPL [14] [15] (Web Services Policy Language) is a subset of XACML and is designed for matching policy descriptions. WS-PolicyConstraints [16] is an even smaller subset of WSPL with the parts of WSPL that overlapped and conflicted with WS-Policy and WS-PolicyAttachment removed. Although these XACML-based solutions are platform independent, they are too complex and have unfriendly XML syntax. Their current tool support is also very poor and they cannot be used for modelling.

WS-Policy and XACML provide expressions to specify different configurations of policy assertions, however, most SOA products support only a single configuration option per endpoint. Different configurations are published on different endpoints. Hence, there is usually no need to dynamically choose between options; to be able to specify the exact same configuration on the client and server side between different products is a more important task in practice.

## III. META-MODEL FOR THE WS-POLICY STANDARDS

This section introduces the high level description language and the metamodel for WS-Policy standards.

### A. SOAL and SoaMM

In the SOA world, XML is used for interface and process-description, and message-formatting. The aim is interoperability, but the drawback is that handling and transforming XML documents above a certain complexity is almost impossible. This is why most development environments have graphical tools for helping developers creating interface-descriptions, connections, process-flows and message-formats. The problem with the graphical approach is that it is neither efficient, nor reliably repeatable, nor sufficiently controllable, nor easily automatable.

On the other hand, the standards usually have a lot of redundant parts that result in poor readability and manageability. For example the `message` element in WSDL 1.x was omitted from WSDL 2.0 because of its redundancy. The development tools do not support the inclusion of special extensions in the interface specification (like pre- and post-conditions, authorization, etc.) Even some tools have special naming conventions that are to be accepted, otherwise the generated code is even less readable than necessary. We

 ISBN: 978-1-61208-116-8 

have examined a lot of products [17] and have found a lot of peculiarities, which have to be taken into account beyond the recommendations of the WS-I Basic Profile. In BPEL process-descriptions the `partnerLinkType-partnerLink` constructs for partners, or the `property-propertyAlias-correlationSet` constructs for correlations mean unnecessary redundancy in BPEL. Unfortunately, the process designer tools usually map these constructs directly to the graphical interface instead of hiding them from the users. The development tools do not support the inclusion of special extensions in the process description (like pre- and post-conditions, authorization, etc.)

To solve the above problems, an abstract model (SoaMM) has been developed that can manage both BPEL and WSDL concepts, and, in order to describe the model, an extendable language called Service Oriented Architecture Language (SOAL) was specified [18], [19] that can be used for describing webservice interfaces and BPEL processes, and is more easily readable and manageable by humans. This model and language also enables automatization, vendor-specific WSDL and BPEL generation, and compile time type checking.

The following example illustrates a simple stack web service description in SOAL. The service has the URL http://localhost/Calculator and can be accessed through SOAP 1.1 over HTTP:

```
namespace CalculatorSample {
  interface ICalculator {
    double Add(double left, double right);
    double Subtract(double left, double right);
    double Multiply(double left, double right);
    double Divide(double left, double right);
  }
  binding Soap11HttpBinding {
    transport HTTP;
    encoding SOAP { Version = SoapVersion.Soap11 }
  }
  endpoint Calculator : ICalculator {
    binding Soap11HttpBinding;
    location "http://localhost/Calculator";
  }
}
```

### B. WS-Policy in SoaMM and SOAL

The most widely supported WS-* protocols by the industry are WS-Addressing, WS-ReliableMessaging, WS-SecureConversation and WS-AtomicTransaction. Each of them has a respective WS-Policy standard, which defines assertions that can be used to describe a set of parameters in a platform independent way to configure these protocols.

SOA products implement these protocols as web services stacks. These stacks consist of various layers, and the products offer different kinds of configuration mechanisms to set the parameters of these layers. We have reviewed all the WS-Policy standards corresponding to the protocols enumerated at the beginning of this section, and we have also examined the configuration mechanisms of the most popular SOA products. Oracle SOA Suite and IBM WebSphere directly use WS-Policy assertions, which can be selected from a repository. GlassFish ESB, using the Metro web services stack, also consumes WS-Policy assertions, however, it offers a graphical user interface built into Netbeans in order to make the settings easier. Apache CXF and Microsoft WCF transform WS-Policy assertions into their own configuration representation. These differencies between products make it very hard to achieve interoperability when the number of protocols in the stack increases, since finding the exact same options between the various representations can be error prone. It is not a coincidence, that GlassFish ESB being a Java implementation offers configurations labeled as .NET interoperable in its GUI settings.

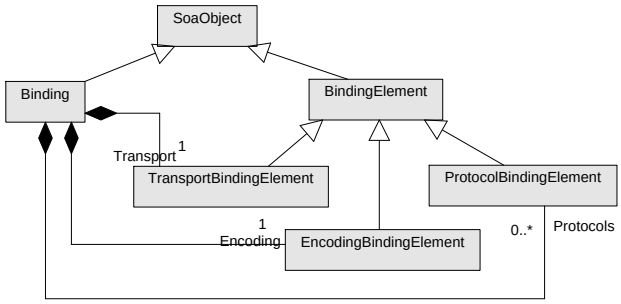


Figure 1. Bindings and binding elements in SoaMM

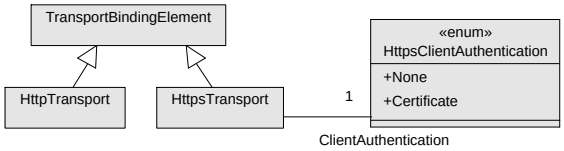


Figure 2. Transport binding elements in SoaMM

After reviewing all of these configuration representations we have created a platform independent metamodel as an extension of SoaMM in order to be able to describe all the parameters of the various WS-* protocols and to be able to generate directly interoperable configurations for the individual SOA products.

Figure 1 shows the basic building blocks of the policy metamodel. *SoaObject* is the root of the class hierarchy in SoaMM. *Binding* contains a set of protocols represented by *BindingElement*s. *TransportBindingElement*, *EncodingBindingElement* and *ProtocolBindingElement* denote transport protocols (e.g., HTTP, UDP, JMS, etc.), encoding protocols (e.g., SOAP, binary, etc.) and higher level protocols (e.g., WS-SecureConversation, etc.) respectively.

Figure 2 contains the two transport protocols currently supported by our metamodel: *HttpTransport* for HTTP and *HttpsTransport* for HTTPS. Through the *ClientAuthentication* property of type *HttpsClientAuthentication* the latter

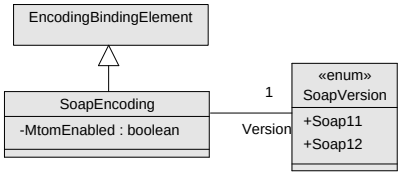


Figure 3. Encoding binding elements in SoaMM

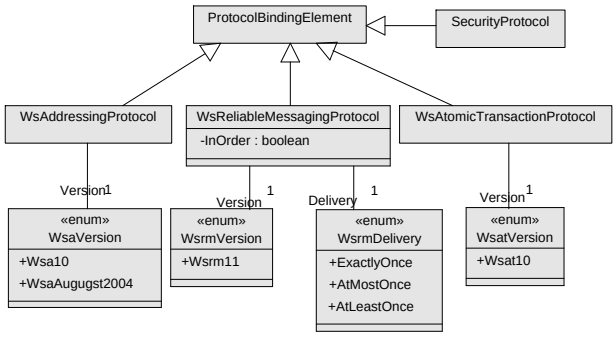


Figure 4. Protocol binding elements in SoaMM

can prescribe that the client must authenticate itself with an X.509. certificate.

Web services use the SOAP encoding protocol for communication. Our metamodel offers *SoapEncoding* (see Figure 3) to represent this protocol and also allows the MTOM option to be enabled for increased efficiency in the communication. The SOAP version can be specified through the *Version* property.

Figure 4 introduces the higher level protocols. *WsAddressingProtocol*, *WsReliableMessagingProtocol* and *WsAtomicTransactionProtocol* denote the settings of WS-Addressing, WS-ReliableMessaging and WS-AtomicTransaction respectively. Each of these has a *Version* property for the version number. The *Delivery* property of *WsReliableMessagingProtocol* defines what kind of delivery should be used in the WS-ReliableMessaging protocol while the *InOrder* property indicates whether the order of the messages should be preserved. The security protocols are omitted from this paper because of space limitations, however, in the figure they are marked by the *SecurityProtocol* class.

We have also extended the SOAL language so that the various protocol settings can be represented in the language. The main extension point is the binding declaration. For example the following binding can be used to configure WS-Addressing 1.0 and WS-ReliableMessaging 1.1 with messages delivered exactly once over HTTPS:

```
binding ReliableBinding {
  transport HTTPS {
   ClientAuthentication =
     HttpsClientAuthentication.None
  }
  encoding SOAP {
    Version = SoapVersion.Soap11
  }
  protocol WsAddressing {
    Version = WsaVersion.Wsa10
  }
  protocol WsReliableMessaging {
    Version = WsrmVersion.Wsrm11,
    Delivery = WsrmDelivery.ExactlyOnce,
    InOrder = true
  }
}
```

The corresponding WS-Policy assertions used by most SOA products would be:

```
<wsp:Policy wsu:Id="ReliableBinding_Policy">
  <sp:TransportBinding>
    <wsp:Policy>
      <sp:TransportToken>
        <wsp:Policy>
          <sp:HttpsToken
             RequireClientCertificate="false"/>
        </wsp:Policy>
      </sp:TransportToken>
      <sp:AlgorithmSuite>
        <wsp:Policy>
          <Basic256/>
        </wsp:Policy>
      </sp:AlgorithmSuite>
      <sp:Layout>
        <wsp:Policy>
          <sp:Strict/>
        </wsp:Policy>
      </sp:Layout>
      <sp:IncludeTimestamp/>
    </wsp:Policy>
  </sp:TransportBinding>
  <wsam:Addressing/>
  <wsrmp:RMAssertion>
    <wsp:Policy>
      <wsrmp:DeliveryAssurance>
        <wsp:Policy>
          <wsrmp:ExactlyOnce/>
          <wsrmp:InOrder/>
        </wsp:Policy>
      </wsrmp:DeliveryAssurance>
    </wsp:Policy>
  </wsrmp:RMAssertion>
</wsp:Policy>
```

The WCF configuration would be the following:

```
<binding name="ReliableBinding">
  <reliableSession reliableMessagingVersion=
                     "WSReliableMessaging11"
     ordered="true" />
  <textMessageEncoding
     messageVersion="Soap11WSAddressing10" />
  <httpsTransport
     requireClientCertificate="false" />
</binding>
```

As it can be seen, the SOAL description is much cleaner than the bloated XML-based WS-Policy assertions and is as compact as the custom configuration representation of WCF.

### C. Architecture

The grammar for the WS-Policy extension in SOAL was implemented in the M language designed by Microsoft. The M language is a declarative language for working with data and building domain models. It is part of the SQL Server

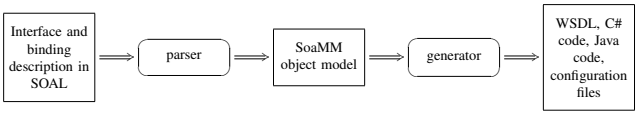


Figure 5. The architecture of the platform independent SOAL framework

Modeling Services [20] (formerly Oslo) framework, which also includes an editor called IntelliPad for domain specific languages and a repository for storing data models.

Based on the grammar the M language parser can process binding descriptions in the SOAL form and can transform the input into an object model described by the SoaMM metamodel. The object model is then easily processable from any .NET language. From this object model the framework generates directly importable projects for different SOA products. The projects include web services with directly interoperable configurations even between different products; the programmer only has to specify the implementation of the services. The code generator is written in the Text Template Transformation Toolkit for Visual Studio.

The architecture of the platform independent SOAL for WSDL framework can be seen in Figure 5.

### D. Generated code

Our framework generates code for two popular web service technologies: Windows Communication Foundation (WCF [21]) and Java API for XML-Based Web Services 2.0 (JAX-WS [22]). The generated C# and Java classes contain only standard elements, hence they can be used in any SOA product implementing these standards (e.g., Microsoft .NET, Oracle SOA Suite, IBM WebSphere, Apache CXF).

However, JAX-WS (unlike WCF) does not cover WS-Policy standards for non-functional requirements, therefore, configuration files for SOA products may differ between JAX-WS implementations. Our framework currently supports WCF and the following JAX-WS implementation stacks: Metro (used in GlassFish ESB), IBM WebSphere and Oracle SOA Suite. The open-source Apache CXF (also used in the JBossWS stack) is planned to be supported.

Table I shows the number of lines required in configuration files of the various web services stacks regarding the different protocols. It can be seen that SOAL is as compact as WCF, while the other configuration representations are more verbose. Therefore, generating these configuration files, so that they are even directly interoperable, results in great increase of productivity.

### E. Evaluation

SOAL is a human readable domain specific language for SOA with a metamodel called SoaMM behind it. It provides much cleaner syntax than an XML based WSDL document. From a SOAL description a SoaMM object model can be constructed and from this object model WSDL files, program code and configuration files can be automatically generated.

Table I
NUMBER OF LINES OF THE VARIOUS CONFIGURATIONS IN SOA PRODUCTS

Legend: WS-A=WS-Addressing, WS-RM=WS-ReliableMessaging, WS-AT=WS-AtomicTransaction, WS-SC=WS-SecureConversation

| Protocol | SOAL | WCF | Metro/IBM/Oracle | CXF/JBoss |
|---|---|---|---|---|
| HTTP | 1 | 1 | 1 | 1 |
| HTTPS | 2 | 1 | 20 | 29 |
| SOAP | 2 | 1 | 1 | 1 |
| MTOM | 2 | 1 | 4 | 14 |
| WS-A | 2 | 1 | 3 | 5 |
| WS-RM | 4 | 1 | 10 | 11 |
| WS-AT | 2 | 1 | 10 | - |
| WS-SC | 4 | 5 | 40 | 40 |

This is a powerful tool in the top-down development process where the task is to create interoperable web services based on WSDLs, while it is as simple as the less interoperable bottom-up development process primarily supported by SOA products. The C#-Java-like textual syntax of SOAL is easier to maintain than XML or graphical WSDL representations provided by the products.

Furthermore, the authors are not aware of any other framework or metamodel that has such a high abstraction level policy representation and strong descriptive power at the same time as SoaMM and SOAL. The metamodel and the language also has advantage over the web services standard family: the descriptions are unified in a single language on all three (transport, encoding, protocol) configuration layers.

The flexibility and extensibility of the metamodel are demonstrated by the integration of standards specifying the most important non-functional aspects, like WS-Addressing, WS-ReliableMessaging, WS-Security, WS-SecureConversation, WS-AtomicTransaction. The metamodel can be easily extended at any time by further solutions for transport, encoding, or protocol.

The most important aspect of the metamodel extension is that configuration profiles can be specified on a high conceptual level. The actual implementation-specific configuration files are automatically generated from these high level descriptions. Interoperability, which is of utmost importance in systems integration, is thus guaranteed without further vendor-specific, low-level product configuration.

There is one weak point of the framework: current BPEL engines usually do not support advanced non-functional requirements (e.g., security, transactions), hence, configuration cannot be generated for them. However, a proxy web service can be produced that delegates calls towards the BPEL process. This solution introduces a major overhead though, since the SOAP XML messages have to be serialized and deserialized twice, instead of once. Nevertheless, this proxy web service concept is useful for other purposes as well, e.g., testing or adapting other web services.

 ISBN: 978-1-61208-116-8

## IV. Conclusion

The original WS-Policy standards specify huge XML descriptions that are too complex and too low level for efficient service design. The paper presented an extension to the metamodel SoaMM and description language SOAL that was originally designed as a simple, human readable domain specific language specifically for webservices and BPEL. This extension provides a high abstraction level that is still strong enough for generating vendor-specific policy configurations. Furthermore, this description guarantees interoperability of SOA products of different vendors. For BPEL engines that do not support advanced non-functional requirements, a proxy web service can be produced that delegates calls towards the BPEL process.

In our future work, we will extend the framework with other concepts, e.g., versioning and SAML authentication.

## Acknowledgment

This work is connected to the scientific program of the "Development of quality-oriented and harmonized R+D+I strategy and functional model at BME" project. This project is supported by the New Hungary Development Plan (Project ID: TMOP-4.2.1/B-09/1/KMR-2010-0002).

## References

[1] OASIS, *SOA Reference Model*, http://www.oasis-open.org/committees/tc_home.php?wg_abbrev=soa-rm, accessed: 8.12.2010.

[2] ——, *WS-* Standards*, http://www.oasis-open.org/specs/, accessed: 8.12.2010.

[3] M. Klusch, *CASCOM - Intelligent Service Coordination in the Semantic Web*. Birkhuser Verlag, Springer, 2008, ch. 3.

[4] W3C, *Semantic Annotations for WSDL and XML Schema (SAWSDL)*, http://www.w3.org/TR/sawsdl/, accessed: 8.12.2010.

[5] ——, *OWL-S: Semantic Markup for Web Services*, http://www.w3.org/Submission/OWL-S/, accessed: 8.12.2010.

[6] E. W. working group, *Web Service Modeling Ontology (WSMO)*, http://www.wsmo.org/, accessed: 8.12.2010.

[7] A. Haller, E. Cimpian, A. Mocan, E. Oren, and C. Bussler, "Wsmx - a semantic service-oriented architecture," in *In Proceedings of the International Conference on Web Service (ICWS 2005*, 2005, pp. 321–328.

[8] O. Shafiq, M. Moran, E. Cimpian, A. Mocan, M. Zaremba, and D. Fensel, "Investigating semantic web service execution environments: A comparison between wsmx and owl-s tools," *Internet and Web Applications and Services, International Conference on*, vol. 0, p. 31, 2007.

[9] V. Kolovski, B. Parsia, Y. Katz, and J. Hendler, "Representing web service policies in owl-dl," in *In International Semantic Web Conference (ISWC*, 2005, pp. 6–10.

[10] K. Verma, R. Akkiraju, and R. Goodwin, "R.: Semantic matching of web service policies," in *Proceedings of the Second Workshop on SDWP, 2005*, 2005, pp. 79–90.

[11] N. Sriharee, T. Senivongse, K. Verma, and A. Sheth, "On using ws-policy, ontology, and rule reasoning to discover web services," in *Intelligence in Communication Systems*, ser. Lecture Notes in Computer Science, F. A. Aagesen, C. Anutariya, and V. Wuwongse, Eds. Springer Berlin / Heidelberg, 2004, vol. 3283, pp. 246–255, 10.1007/978-3-540-30179-0_22.

[12] A. Charfi, B. Schmeling, A. Heizenreder, and M. Mezini, "Reliable, secure, and transacted web service compositions with ao4bpel," in *ECOWS '06: Proceedings of the European Conference on Web Services*. Washington, DC, USA: IEEE Computer Society, 2006, pp. 23–34.

[13] T. Moses, *eXtensible Access Control Markup Language (XACML)*, http://www.oasis-open.org/committees/tc_home.php?wg_abbrev=xacml, accessed: 8.12.2010.

[14] A. Anderson, *An Introduction to the Web Services Policy Language, Fifth IEEE International Workshop on Policies for Distributed Systems and Networks (POLICY'04)*, http://labs.oracle.com/projects/xacml/Policy2004.pdf, accessed: 8.12.2010.

[15] T. Nadalin, *Web Services Security Policy Language (WS-SecurityPolicy)*, http://www-128.ibm.com/developerworks/library/specification/ws-secpol/, accessed: 8.12.2010.

[16] A. Anderson, *XACML-based Web Services Policy Constraint Language (WSPolicyConstraints)*, http://labs.oracle.com/projects/xacml/ws-policy-constraints-current.pdf, accessed: 8.12.2010.

[17] B. Simon, Z. Laszlo, B. Goldschmidt, K. Kondorosi, and P. Risztics, "Evaluation of ws-* standards based interoperability of soa products for the hungarian e-government infrastructure," in *International Conference on the Digital Society*. Los Alamitos, CA, USA: IEEE Computer Society, 2010, pp. 118–123.

[18] B. Simon and B. Goldschmidt, "A human readable platform independent domain specific language for wsdl," in *Networked Digital Technologies*, ser. Communications in Computer and Information Science. Springer Berlin Heidelberg, 2010, vol. 87, pp. 529–536.

[19] B. Simon, B. Goldschmidt, and K. Kondorosi, "A human readable platform independent domain specific language for bpel," in *Networked Digital Technologies*, ser. Communications in Computer and Information Science. Springer Berlin Heidelberg, 2010, vol. 87, pp. 537–544.

[20] Microsoft, *SQL Server Modeling Services*, http://msdn.microsoft.com/en-us/data/ff394760.aspx, accessed: 8.12.2010.

[21] ——, *Windows Communication Foundation*, http://msdn.microsoft.com/en-us/netframework/aa663324.aspx, accessed: 8.12.2010.

[22] Oracle, *JSR 224: Java API for XML-Based Web Services (JAX-WS) 2.0*, http://jcp.org/en/jsr/detail?id=224, accessed: 8.12.2010.

 ISBN: 978-1-61208-116-8 