Generic Contract Descriptions for Web Services Implementations

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 and BPEL processes. The paper introduces an extended version of the language that supports Design by Contract. From the service contract specifications software artifacts are generated that check pre- and post-conditions on the server side at runtime, applying the delegation pattern. The proposed solution provides Design by Contract for both JAX-WS and WCF technologies used in most SOA products in the industry.**

***Keywords*-SOA; web services; Design-by-Contract; 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 nowadays the components that both enterprises and governments use to construct their complex distributed systems, thus 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. Another classical principle is "Design by Contract" (DbC) introduced by Bertrand Meyer [3]. Contract is a key concept in SOA, thus it seems to be obvious to apply DbC for WebServices. Vendor support for it is usually minimal, and although there are third party solutions, the common drawback is their platform-dependency. Heterogeneous systems like those applied in e-government need a unified description of contracts that can be generally used in different SOA products. Having legacy systems the introduction of a new product or extending an existing product with native contract support is not an option. This paper shows how a high-level web service description language and code generating framework was extended with contract support in order to enable developers to automatically generate platform-specific contract-enforcing modules from the general contract specifications.

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 different governmental organizations, all the solutions we elaborate should be platform-independent as far as it is possible.

The rest of the paper introduces a platform-independent solution of DbC in SOA systems, which can work in any SOA environment even it would be a heterogenous one. In the second section, related work is examined. In the third section, the contract metamodel, its representation in SOAL syntax, the architecture and details of code generation, and an overall evaluation are presented. In the fourth section, the results are summarized.

II. Related Work

Eiffel [3], [4] is an object-oriented programming language that supports Design by Contract. The code is either compiled to native code or to .NET CLR, where it could be applied web services written in the .NET framework. The JVM is not supported, therefore it cannot be used in Java environments.

D. Florescu et.al. [5] proposed a declarative domain specific language with pre- and post-condition support for web services. Their goal is to create web services that can be executed on any platform. However, instead of generating code for the different commercial products, they created a custom execution environment, the XL virtual machine. Their solution also lacks a metamodel behind the language.

WS-CoL [6] is primarily a monitoring language, however, it provides pre- and post-conditions on the interactions between services and it can also be used for BPEL processes, although it supports only the ActiveBPEL engine. WS-CoL has an extension with time constraints described in [7].

Another 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 [8]. 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 [9] 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 [10] 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 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 [11] 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 [12] 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 modeling purposes; they are not supported by the major SOA software vendors and they require a custom execution environment.

Other efforts focus on modelling web services in UML. R. Heckel and M. Lohmann [13] introduce three levels of contract representations: implementation-level, XML-level and model-level. Their goal is to derive implementation- and model-level contracts from the model-level specification. Our aim is similar, but we think that a domain specific language and a metamodel for SOA are much more appropriate for this purpose. J. T. E. Timm [14] defines a UML profile that extends class and activity diagrams. This profile is used in transformations to automatically construct OWL-S specifications from diagrams and SWRL (Semantic Web Rule Language) expressions from OCL. The problem with this approach is the same as with SWS technologies: it cannot be applied in current major SOA products.

III. Design by Contract for SOA

This section presents the contract metamodel, its representation in SOAL syntax, the architecture and details of code generation, and finally, the overall evaluation of the proposed framework.

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 have examined a lot of products [15] 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. Unfortunately, the process designer tools usually map these constructs directly to the graphical interface instead of hiding them from the users.

To solve the above problems, an abstract SOA metamodel called SoaMM has been developed that can manage both BPEL and WSDL concepts, and, in order to describe the model, an extensible language called Service Oriented Architecture Language (SOAL) was specified [16], [17] 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/Stack and can be accessed through SOAP 1.1 over HTTP:

```
namespace StackSample {
  interface IStack {
    void Push(int number);
    int Pop();
    int Top();
    bool IsEmpty();
  }
  binding Soap11HttpBinding {
    transport HTTP;
    encoding SOAP { Version = "1.1" }
  }
  endpoint Stack : IStack {
    binding Soap11HttpBinding;
    location "http://localhost/Stack";
  }
}
```

B. Design by Contract in SOAL

We extended SOAL with contract descriptions. A contract can be specified using the `contract` keyword and must implement exactly one interface. The implementation of an operation has to specify the pre-conditions with the `requires` keyword and the post-conditions with the `ensures` keyword. After these keywords a textual description has to be included about the condition being checked. This description is included in the error messages on the violation of the conditions. Invariants are not yet supported but they are subject to further investigation. The current instance of the service can be accessed through the `this` keyword and the `result` keyword represents the return value of the current operation. In the endpoint declaration the name of the contract must be specified with the `contract` keyword. Here is an example of the contract extension for SOAL applied to the `IStack` interface defined in the previous section:

```
contract StackContract : IStack {
  void Push(int number) {
    ensures "stack is not empty"
    { !this.IsEmpty(); }
    ensures "top equals to number"
    { this.Top() == number; }
  }
  int Pop() {
    requires "stack is not empty"
    { !this.IsEmpty(); }
    ensures "result is the old top element"
    { result == old(this).Top(); }
  }
  int Top() {
    requires "stack is not empty"
    { !this.IsEmpty(); }
  }
  bool IsEmpty() { }
}
endpoint GuardedStack : IStack {
  binding Soap11HttpBinding;
  contract StackContract;
  location "http://localhost/GuardedStack";
}
```

The operations may change the state of a stateful web service. In this case the method calls on `this` may return different values after the execution of the current operation than before. The `old` expression can be used to access the state prior to the execution of current operation. The input parameters of the operations are read-only, therefore there is no need to use `old` on them. The expressions in the `requires` and `ensures` clauses have the same syntax as the expressions in C#. In .NET 3.0 the API introduced the `System.Linq.Expressions` namespace with classes that can be used to construct expression trees in memory. Our extension to SOAL and the metamodel behind it is based on these expression tree nodes and supports the following node types (including lambda expressions) [18]: `Add`, `And`, `AndAlso`, `ArrayLength`, `ArrayIndex`, `Call`, `Coalesce`, `Conditional`, `Constant`, `Convert`, `Default`, `Divide`, `Equal`, `ExclusiveOr`, `GreaterThan`, `GreaterThanOrEqual`, `Lambda`, `LeftShift`, `LessThan`, `LessThanOrEqual`, `MemberAccess`, `MemberInit`, `Modulo`, `Multiply`, `Negate`, `UnaryPlus`, `New`, `NewArrayInit`, `NewArrayBounds`, `Not`, `NotEqual`, `Or`, `OrElse`, `OnesComplement`, `Parameter`, `RightShift`, `Subtract`, `TypeAs`, `TypeIs`, `Variable`.

The semantics of these expressions are defined by the C# language specification [19].

SOAL also supports array types. The .NET 3.0 framework allows arrays to be queried through the LINQ API, which provides a lot of useful query functions. Our extension to SOAL supports all of these [20]: `Aggregate`, `All<TSource>`, `Any`, `AsEnumerable<TSource>`, `Average`, `Cast<TResult>`, `Concat<TSource>`, `Contains`, `Count`, `DefaultIfEmpty`, `Distinct`, `ElementAt<TSource>`, `ElementAtOrDefault<TSource>`, `Empty<TResult>`, `Except`, `First`, `FirstOrDefault`, `GroupBy`, `GroupJoin`, `Intersect`, `Join`, `Last`, `LastOrDefault`, `LongCount`, `Max`, `Min`, `OfType<TResult>`, `OrderBy`, `OrderByDescending`, `Range`, `Repeat<TResult>`, `Reverse<TSource>`, `Select`, `SelectMany`, `SequenceEqual`, `Single`, `SingleOrDefault`, `Skip<TSource>`, `SkipWhile`, `Sum`, `Take<TSource>`, `TakeWhile`, `ThenBy`, `ThenByDescending`, `ToArray<TSource>`, `ToDictionary`, `ToList<TSource>`, `ToLookup`, `Union`, `Where`.

All of the OCL expressions are covered by the expressions above, except for the ones dealing with messages, object states and associations, which themselves are specific to UML. Since OCL has proved itself to be powerful enough in practice and its UML independent part is a subset of the LINQ expressions, it can be stated that our extensions to SOAL will suffice in most practical cases.

C. Architecture

The grammar for the contract 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 Modeling Services [21] (formerly Oslo) framework, which

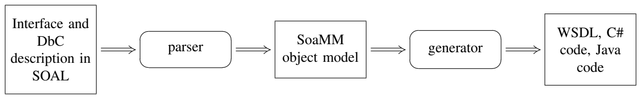

Figure 1. The architecture of the platform independent DbC for SOAL framework

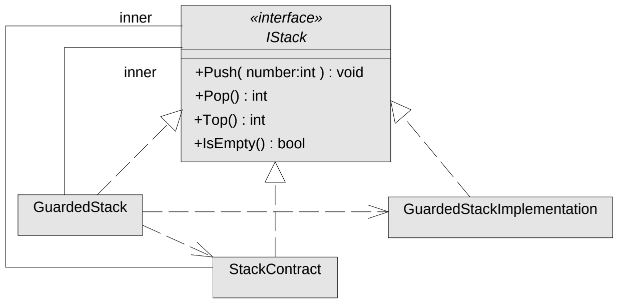

Figure 2. The design pattern of the generated C#/Java classes demonstrated on the stack example

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 contract 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, which check the specified pre- and post-conditions automatically; the programmer only has to insert 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 1.

D. Generated code

Our framework generates code for two popular web service technologies: Windows Communication Foundation (WCF [22]) and Java API for XML-Based Web Services 2.0 (JAX-WS [23]). The generated 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).

Figure 2 shows the design pattern of the target code regardless of the target platform (i.e., C# for WCF or Java for JAX-WS). From the `IStack` interface in SOAL an interface with the same name is generated in the target language. The contract `StackContract` is mapped to a class with the same name. The endpoint `GuardedStack` is transformed into two classes. The first one is the class `GuardedStack`, which is the web sevice endpoint published by the SOA products. The second one is the class `GuardedStackImplementation`, which contains the implementation of the service. Its operations must be filled by the programmer. The `GuardedStack` class uses the delegate design pattern to check the implementation by the specified contract using the following chain of calls: `GuardedStack` → `StackContract` → `GuardedStackImplementation`.

The following C# interface annotated with WCF attributes is generated from the `IStack` interface:

```
[ServiceContract(...)]
public interface IStack {
  [OperationContract(...)]
  void Push(int number);
  [OperationContract(...)]
  int Pop();
  [OperationContract(...)]
  int Top();
  [OperationContract(...)]
  bool IsEmpty();
}
```

The generated Java interface is similar, but of course it uses JAX-WS annotations, i.e., `@WebService` instead of `[ServiceContract]` and `@WebMethod` instead of `[OperationContract]`.

From the `StackContract` contract specification the following C# class is produced (the attribute named `inner` will contain the implementation of the service and every call is delegated to this object):

```
public class StackContract : IStack {
  private IStack inner;
  public StackContract(IStack inner) {
    this.inner = inner;
  }
  public void Push(int number) {
    this.inner.Push(number);
    if (!(!this.IsEmpty())) {
      throw new PostConditionViolationException(
                "stack is not empty");
    }
    if (!(this.Top() == number)) {
      throw new PostConditionViolationException(
                "top equals to number");
    }
  }
  public int Pop() {
    if (!(!this.IsEmpty())) {
      throw new PreConditionViolationException(
                "stack is not empty");
    }
    int temp1 = this.Top();
    int result = this.inner.Pop();
    if (!(result == temp1)) {
      throw new PostConditionViolationException(
                "result is the old top element");
    }
    return result;
  }
  public int Top() {
    if (!(!this.IsEmpty())) {
      throw new PreConditionViolationException(
                "stack is not empty");
    }
    int result = this.inner.Top();
    return result;
  }
  public bool IsEmpty() {
    bool result = this.inner.IsEmpty();
    return result;
  }
}
```

As it can be seen, the `old` expressions are evaluated into temporary variables before the call is delegated to the implementation. After the execution of the implementation the post-conditions are checked correctly. The Java version of this class is similar. The expressions specified in Section III-B are translated to Java as well: the LINQ Standard Query Operators are backed up by a custom utility class, the lambda expressions are transformed into anonymous classes.

The `GuardedStackImplementation` class:

```
public class GuardedStackImplementation : IStack {
  public void Push(int number) {
    throw new NotImplementedException();
  }
  public int Pop() {
    throw new NotImplementedException();
  }
  public int Top() {
    throw new NotImplementedException();
  }
  public bool IsEmpty() {
    throw new NotImplementedException();
  }
}
```

This code is very clean, since the pre- and post-conditions are generated into the `StackContract` class. The programmer has to fill in the missing implementations in order to have a functioning web service.

The service endpoint class `GuardedStack` has to be published. It is also very simple; it builds up the delegation chain and delegates the calls to the other classes:

```
public class GuardedStack : IStack {
  private IStack inner;
  public GuardedStack() {
    this.inner =
      new StackContract(
        new GuardedStackImplementation());
  }
  public void Push(int number) {
    this.inner.Push(number);
  }
  public int Pop() {
    return this.inner.Pop();
  }
  public int Top() {
    return this.inner.Top();
  }
  public bool IsEmpty() {
    return this.inner.IsEmpty();
  }
}
```

The Java versions of the generated classes are similar to the C# examples above.

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 and program code can be automatically generated. 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 the XML or graphical WSDL representations provided by the products.

Although Design by Contract is a key concept in SOA, the major software vendors do not provide any tools to make this task easier. In this paper we proposed a contract extension to SOAL. This extension offers the same maintainability and readability as the original version of the language. Pre- and post-conditions can be specified for each operation. These conditions are then woven as aspects into the generated code, while the programmer has only a single task: provide the implementation of the web service by filling in the methods of the implementation class. Every other configuration and source files for the SOA products are automatically produced by our framework. Currently two SOA products are supported: Microsoft Visual Studio for WCF and GlassFish ESB for JAX-WS. However, with the appropriate configuration files the Java code generated by our framework can be directly used in other products (e.g., Oracle SOA Suite and IBM WebSphere) as well.

Our proposition offers an extensive set of operators that can be utilized in pre- and post-condition expressions. These operators do not introduce any new concepts, therefore they are easy to learn. They also have a well defined semantics based on the C# language specification. The expressions can also contain lambda functions and thus the benefits of the LINQ standard query operators can also be harvested. These query operators build a superset of OCL (without the UML specific parts), which itself is also a powerful constraint description language. Although the expression syntax in SOAL is the same as in C#, the expressions are translated by our framework to Java as well.

The design pattern generated by the framework separates the interface, the publication part, the contract validation part and the implementation part of a service. This results in a clean, easily maintainable code. Although the delegation of method calls in this design pattern introduces a minor overhead, this is negligible compared to the time consumed by transforming the incoming and outgoing SOAP XML messages.

There is one weak point of the framework: the expressions are too powerful to be translated into BPEL, therefore BPEL code cannot be produced from them. However, a proxy web service can be generated that delegates calls towards the BPEL process instead of towards a local implementation class. 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 other web services.

IV. Conclusion

We proposed a Design by Contract solution for SOA. Our solution is based on a simple, human readable domain specific language called SOAL. Our framework generates C# or Java code from this description, that can be directly imported into the SOA products of the major software vendors. The generated artifacts follow the delegation design pattern, which results in a clean and easily maintainable code. The syntax and semantics of the condition expressions are based on C# and the LINQ standard query operators are also fully supported. Design by Contract can also be applied to BPEL though proxy web services at the cost of some performance loss.

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

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] B. Meyer, “Applying ”design by contract”,” *Computer*, vol. 25, no. 10, pp. 40–51, 1992.

[4] ——, *Touch of Class: Learning to Program Well with Objects and Contracts*. Springer, 2009.

[5] A. G. D. Florescu and D. Kossmann, “Xl: an xml programming language for web service specification and composition,” in *Computer Networks: The International Journal of Computer and Telecommunications Networking*, 2003, pp. 1–25.

[6] L. Baresi, S. Guinea, M. Pistore, and M. Trainotti, “Dynamo + astro: An integrated approach for bpel monitoring,” *Web Services, IEEE International Conference on*, vol. 0, pp. 230–237, 2009.

[7] L. Baresi, D. Bianculli, C. Ghezzi, S. Guinea, and P. Spoletini, “A timed extension of wscol,” *Web Services, IEEE International Conference on*, vol. 0, pp. 663–670, 2007.

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

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

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

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

[12] 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.

[13] R. Heckel and M. Lohmann, “Towards contract-based testing of web services,” *Electronic Notes in Theoretical Computer Science*, vol. 82, p. 2003, 2004.

[14] J. T. E. Timm, “Specifying semantic web service compositions using uml and ocl,” in *In 5th International Conference on Web Services*. IEEE press, 2007.

[15] 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.

[16] 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.

[17] 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.

[18] Microsoft, *.NET LINQ ExpressionType Enumeration*, http://msdn.microsoft.com/en-us/library/bb361179.aspx, accessed: 8.12.2010.

[19] ——, *C# Language Specification Version 4.0*, http://www.microsoft.com/downloads/en/details.aspx?familyid=DFBF523C-F98C-4804-AFBD-459E846B268E&displaylang=en, accessed: 8.12.2010.

[20] ——, *.NET LINQ Standard Query Operators*, http://msdn.microsoft.com/en-us/library/bb882641.aspx, accessed: 8.12.2010.

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

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

[23] 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.