Periodica Polytechnica
Electrical Engineering and Computer Science

A Platform Independent Access Control Metamodel for Web Services

58(3), pp. 93-108, 2014
DOI:10.3311/PPee.2093

Balázs Simon / *Balázs* Goldschmidt / *Károly* Kondorosi

RESEARCH ARTICLE

RECEIVED 18 APRIL 2013, ACCEPTED AFTER REVISION 4 MARCH 2014

Abstract

Web services provide platform independent communication through an XML-based standard family. The major software vendors released their own SOA products implementing these standards. However, the configuration of the WS- protocols differs from product to product. Matching these configurations between different products can be a tedious task. In addition, security protocols are complicated to configure, especially if access control is also required. Although the XACML standard aims to solve this problem, its rules and policies described in XML are not user friendly, and XACML has little support in the major SOA products. Therefore, we have developed a platform independent metamodel for describing distributed systems of web services. From models described in this metamodel the platform specific configurations and program code can be generated for the various SOA products, increasing the productivity of the development. This article introduces an access control extension to this metamodel.*

Keywords

web services · WS- standards · SAML · claims-based identity · metamodeling*

Balázs Simon
Department of Control Engineering and Information Technology, Faculty of Electrical Engineering and Informatics, Budapest University of Technology and Economics, Magyar tudósok krt. 2., H-1117 Budapest, Hungary
e-mail: sbalazs@iit.bme.hu

Balázs Goldschmidt
Department of Control Engineering and Information Technology, Faculty of Electrical Engineering and Informatics,Budapest University of Technology and Economics, Magyar tudósok krt. 2., H-1117 Budapest, Hungary
e-mail: balage@iit.bme.hu

Károly Kondorosi
Department of Control Engineering and Information Technology, Faculty of Electrical Engineering and Informatics, Budapest University of Technology and Economics, Magyar tudósok krt. 2., H-1117 Budapest, Hungary
e-mail: kondorg@iit.bme.hu

1 Introduction

Web services realize distributed communication in a platform independent way by building the related standards on XML. The communication protocol (SOAP), the interface description language (WSDL) and the middleware aspects, called WS-* standards (WS-Addressing, WS-ReliableMessaging, WS-Security, WS-SecureConversation), are all represented in XML. These middleware aspects can also be configured in a platform independent way through XML chunks called WS-Policy assertions which are usually included in the WSDL. The primary aim using XML for communication is to promote interoperability between different platforms.

However, the major drawback of the WS-Policy standard family is that the policy assertions of the WS-* standards can be large XML structures which makes policy assertions 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 be diverse in different 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 WCF) making interoperability more problematic.

Most issues arise with the configuration of security protocols, especially security tokens and security token services, since they require a lot of parameters, like certificates, encryption and digital signature algorithms. The XACML (eXtensible Access Control Markup Language) standard is designed for defining policies and rules for authorization decisions. It provides Attribute-Based Access Control (ABAC) in a platform independent way. However, since it is also based on XML, it is not convenient to write XACML policies by hand. Another problem with XACML is that it is not yet widespread. For example, Microsoft WCF does not yet support it and most open-source implementations rely on the old Sun XACML library.

It is still an open issue [9] to create a general access control metamodel that can unite all the access control approaches, like

role-based access control (RBAC) and attribute/claims-based access control (ABAC/CBAC). Although, it may be possible to construct such a metamodel [2], our goal is not to find the ultimate solution for this problem. Our intention is to define a metamodel for describing the access control of web services with the aim of reducing the development complexity of web services with security aspects.

The considerations above inspired us to create a platform independent metamodel for describing distributed systems built from web services and also a platform independent programming language for configuring such distributed systems. There are propositions to use of UML for this task [5, 11], but UML itself is not suitable, since it lacks most of the concepts of the WS-* standard family even when it is extended with stereotypes [6, 7, 22]. However, a domain specific language designed particularly for web services can be more convenient to use and can increase productivity. Therefore, we created such a metamodel, called SoaMM (SOA Metamodel). It also has a textual concrete syntax called SOAL (SOA Language), which is a programming language for describing distributed systems of web services. We also wrote a compiler that transforms SOAL descriptions into models (instances of the SoaMM metamodel), and also a code generator that produces program code and platform specific configuration files for the various SOA products to implement the described system. The produced artefacts provide consistent configurations between the different SOA products resulting in interoperable applications.

In previous conference proceedings we published [24, 25] some parts of this metamodel. The present article introduces an extension to this metamodel that provides attribute-based access control. The rest of the paper is structured as follows. The second section covers the related work in modeling access control for web services. The third section contains the description of the SoaMM metamodel with the access control extension. The fourth section shows an example how easily the proposed metamodel and programming language can be used to implement a WS-Federation scenario, which is usually a complicated task. The fifth section evaluates the productivity of our framework through some examples. The last section concludes the article and lists the possible future directions.

2 Related Work

The XACML specification [21] contains a metamodel for the access control policies and rules. However, this metamodel does not deal with web services and with the WS-* protocols, and the concrete syntax of XACML is a complex XML structure, which is hard to maintain by hand.

C. Emig et. al. [8] defined a policy model called Web Service Access Control Markup Language (WSACML). The concepts in their model are similar to XACML, however, their model is more specific for web services. Their solution is a combination of RBAC and ABAC. Unfortunately, they chose XML as a concrete syntax for the model, which is harder to maintain than a code written in a domain specific language. It is unclear from the paper, what kind of expressions can be defined in the model, and they do not take other WS-* protocols into consideration.

M. Alam et. al. [1, 12] created the SECTET framework for model driven security. They incorporated all the XACML v3.0 concepts into their framework and they are able to generate XACML policies from the models. They use UML as a visual model and OCL for defining access control conditions. However, they define authorization for roles instead of the more general ABAC solution, and they do not consider other WS-* protocols in their model.

T. Mouelhi et. al. [20] proposed a model-driven approach for specifying, deploying and testing security policies in Java applications. It is based on a generic security meta-model which can be used for early consistency checks in the security policy. This model is then automatically transformed into security policy for the XACML platform and integrated in the application using aspect-oriented programming. However, they also build on RBAC, and they do not cover web services.

M. Giordano et. al. [10] proposed a visual language to specify access and security policies according to the RBAC model. They also point out the complexity of XACML, and they show how XACML policies can be generated from their visual language. Unfortunately, they do not deal with the more general claims-based access control model and with web services.

X. Jin [15] created an MDA approach to model RBAC systems. This approach uses UML with stereotype extensions as a visual language for defining models. The access control rules can be described in OCL. However, this solution is also restricted to RBAC, and it does not deal with web services.

M.E. Jiague et. al. [14] specified a platform independent model, a platform specific model and translation rules between them to generate BPEL processes for authorization decisions. Unfortunately, their model is too high-level, and it is unclear what kind of expressions can be used for defining the authorization conditions.

C. Wolter et. al. [31] proposed to model certain types of security goals in a graphical fashion at the business process modeling level, which in turn can be transformed into corresponding access control and security policies. This is a promising approach, since defining access control in a graphical way is more intuitive than maintaining XML files. However, their solution is too high-level, and most of the generated configuration options cannot be parameterized graphically. For example, the parameters of the WS-Security protocol cannot be specified graphically.

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 provde automated, interoperable and meaningful coordination of web services [16]. The three main directions of SWS are SAWSDL, OWL-S and WSMO.

SAWSDL [30] 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 [29] 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 [32] 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, that is 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 post conditions, assumptions and effects. Mediators provide interoperability between components at data, protocol and process level. The reference implementation of WSMO is the WSMX [13] 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 weak in terms of security, transactional, reliability and other non-functional aspects even at the conceptual level [23]. 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 [17, 27, 28], these solutions focus on service discovery and policy matching, and do not resolve the issues related to modeling and implementation.

T.-Y. Chen [4] developed a knowledge access control policy (KACP) language model for virtual enterprises. It is an ontology-based access control approach. Its drawback is that policies written in the proposed KACP language model are difficult to read and analyze, it only describes concepts, it does not deal with web services and WS-* protocols, and the textual concrete representation is also XML, which is hard to maintain.

A domain specific model designed particularly for web services with a friendly concrete syntax is easy to understand, more convenient to use and is more productive for web service development than the solutions listed above. We considered extending one of the existing solutions to match these requirements. Solutions with UML and OCL, and the ones with ontologies are inconvenient for the web service domain, especially for configuring WS-* policies. Solutions that implement only RBAC are too restricted, and their metamodel and concrete syntax require a lot of modification for ABAC and WS-* policy support. The remaining domain-specific solutions have XML as a concrete syntax, which is inconvenient to edit by humans. XACML, which is the most widely adopted solution of these, suffers from this problem, too. Other solutions do not have a significant user base, and the amount of work to extend them is comparable to creating a new framework from scratch. Therefore, we decided to choose the latter. Our target users are the developers who use the SOA products of major software vendors. Our framework does not replace these products, it simply helps these developers with the top-down development of web services by increasing their productivity.

Our solution concentrates on the more general ABAC approach. It has a domain specific metamodel and a simple textual concrete syntax. This syntax is easier to maintain than a verbose XML description. Another advantage of our framework is that it also supports other WS-* protocols.

3 Access Control Metamodel for Web Services

Since almost every web service framework uses a different kind of configuration format it is hard to match the options of the different frameworks. This is the reason why we decided to create a common metamodel for the WS-* standard family in order to be able to model the services in a platform independent way. From this model the platform dependent configurations of the various frameworks can be automatically generated.

This section specifies the SoaMM metamodel which can be used for modeling access control for distributed services in a platform independent way. Some parts of the metamodel have already been published [24, 25]. Since the focus of this article is the access control aspect, the other aspects of the metamodel are not discussed here.

Figure 1 shows the architecture of the SoaMM metamodel. Most of the WSDL parts are mapped to the SoaMM metamodel. The only exception is the message part which is a redundant element in the WSDL description, therefore, it is omitted from the SoaMM metamodel. All the other parts (types, portType, bindings and endpoints) are included in SoaMM and there is an additional declaration for modeling claims to support claims based identity. Bindings define the transport, the encoding and

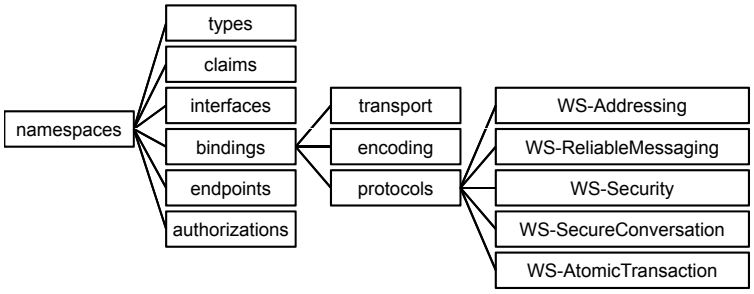

Fig. 1. Architecture of the SoaMM metamodel

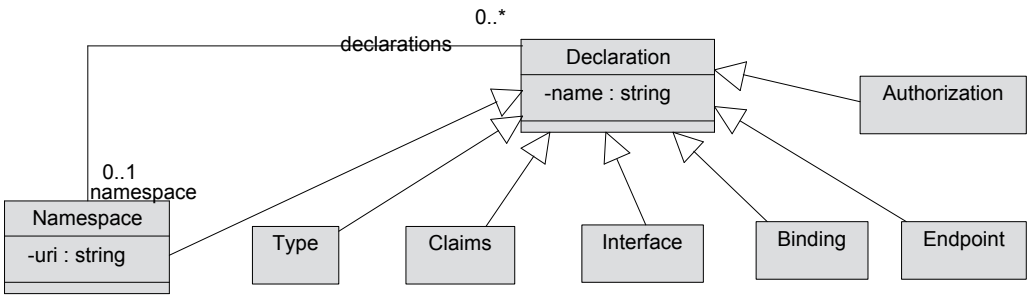

Fig. 2. Namespaces and declarations

the WS-* protocols to be used. The supported protocols are WS-Addressing, WS-ReliableMessaging, WS-Security, WS-SecureConversation, WS-AtomicTransaction. The remaining parts of this section show how the elements in Figure 1 related to access control modeling are defined in the SoaMM metamodel.

There is also a simple programming language called SOAL which provides a textual concrete syntax for the SoaMM metamodel. This language has a similar syntax to C# and Java, but it is specific to the SoaMM domain.

Figure 2 shows the possible declaration elements of SoaMM. SoaMM supports namespaces to avoid name collisions. Namespaces can be hierarchical, therefore, a **`Namespace`** can contain other **`Namespace`** declarations. A **`Type`** is a simple or complex type which constraints the values represented by variables. The **`Claims`** element is a list of claim definitions used in claims based identity management. These claims can be mapped to SAML attributes and WS-Feredation claims. An interface is a collection of operations provided by a service, and it is described by the **`Interface`** element (this is the portType element in the WSDL). Middleware aspects and protocol configurations are specified in the **`Binding`** element. The Endpoint element represents a service running on a specific location (this is the service element in the WSDL). The **`Authorization`** element defines a contract for access control for web services.

The SoaMM modeling language has a strong type system shown in Figure 3. A type can be either a simple type (**`SimpleType`**), a wrapper type (**`WrapperType`**) or a complex type (**`ComplexType`**). Simple types are either built-in types (**`BuiltInType`**) commonly used in most programming languages (int, double, string, etc.) or enumeration types (**`EnumType`**) with a fixed set of values (**`EnumValue`**). Wrapper types are constructed from primitive types by adding some new behavior. Such a wrapper type is the array type (**`ArrayType`**) that represents an array of values and the nullable type (**`NullableType`**) that extends the codomain of non-null simple types with null values. More complex types can be constructed from other types using structured types (**`StructType`**). Structured types also support single inheritance from other structured types. An exception type (**`ExceptionType`**) is similar to a structured type, however, its semantics is different: it represents a fault in SOAP and an exception in conventional programming languages.

One of the main advantages of SoaMM is that claims based identity management is built into the metamodel in a platform- and standard independent way. This means that claims (**`Claims`**) are first class types in the type system. A claim (**`Claim`**) has a name, a uri and a type. These claims can be mapped to WS-Federation claims and also to SAML attributes.

The interfaces (**`Interface`**) of the services are described as a set of operations (**`Operation`**). Figure 4 shows the metamodel of interface declarations. Interfaces support multiple inheritance. An operation can have zero or more input parameters (**`Parameter`**) and a return type (**`returnType`**). If the **`oneway`** property

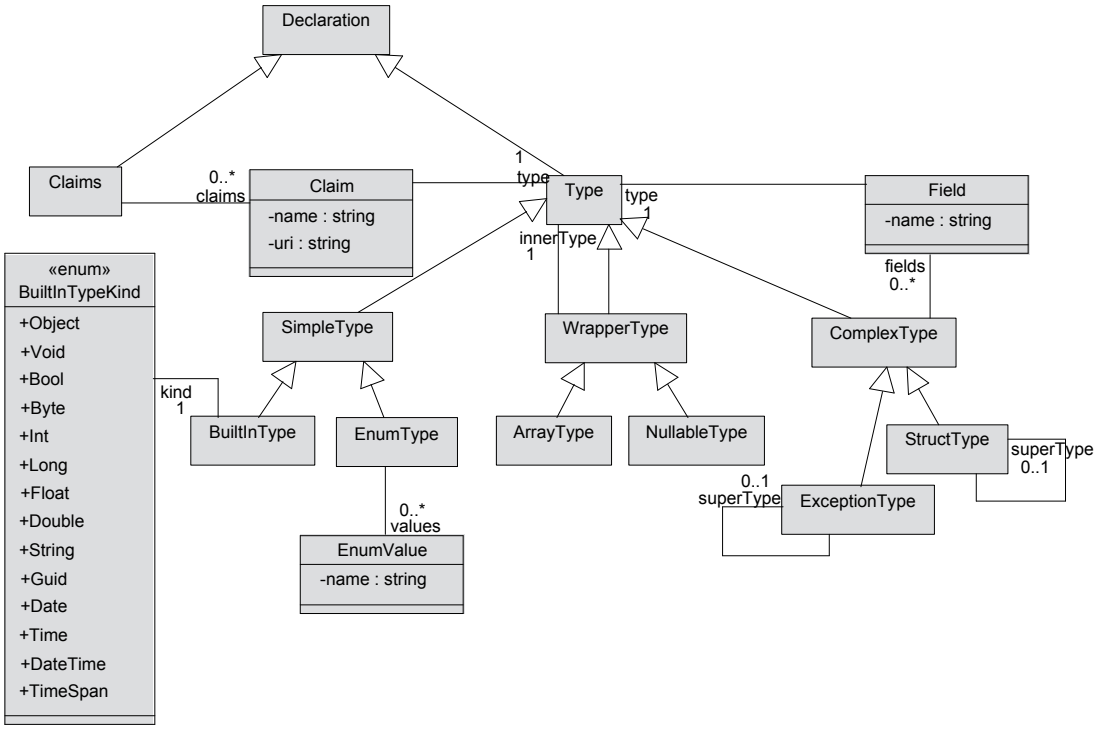

Fig. 3. Types and claims

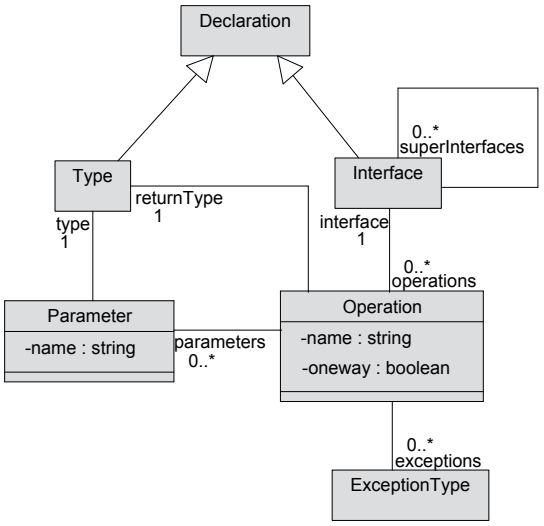

Fig. 4. Interface

is true, the operation follows the one-way message exchange pattern, otherwise it is a request-response operation. One-way operations must have a **`void`** return type and cannot throw exceptions (SOAP faults). Request-response operations can throw exceptions. The possible exception types must be explicitly listed for the operation, just like checked exceptions in Java.

An endpoint implementing a specific interface (**`Interface`**) with a specificbinding (**`Binding`**) running on a specific location is represented by the **`Endpoint`** element (Figure 5). An endpoint contains the address of the web service and may also specify a metadata address, where additional information about the web service may be accessed using the WS-MetadataExchange standard. An endpoint may have an **`Authorization`** element which defines the contract for access control of the given endpoint.

A binding (**`Binding`**) declaration defines through what protocols a service can be accessed. It contains the list of the enabled protocols in the stack. The protocols are represented by binding elements (**`BindingElement`**). A binding has exactly one transport protocol (**`TransportBindingElement`**), exactly one encoding protocol (**`EncodingBindingElement`**) and zero or more WS-* protocols (**`ProtocolBindingElement`**). The SoaMM metamodel currently supports the most common HTTP and HTTPS transport protocols, and SOAP as the encoding protocol.

Figure 6 shows the higher level middleware protocols. **`WsAddressingProtocol`** denotes the settings of the WS-Addressing protocol. The metamodel also supports WS-ReliableMessaging and WS-AtomicTransaction, see [25].

The most complicated protocols to configure are the security protocols (**`WsSecurityProtocol`**). The SoaMM metamodel makes this task a bit easier. At first, the version numbers for the security protocols must be specified: the WS-Security version, the WS-SecurityPolicy version and the WS-Trust version. These are represented in the metamodel by **`WssVersion`**, **`WssSpVersion`** and **`WssTrustVersion`**, respectively (see Figure 6). After this, the security algorithm suite, the layout of the SOAP headers and the protection order of the message elements must be selected. These are represented by **`WssAlgorithmSuite`**, **`WssHeaderLayout`** and **`WssProtectionOrder`**, respectively (see Figure 7). The security elements shown in this Figure are common to all security protocols.

The various security protocols differ in the security tokens that are exchanged between the parties. At first, it must be decided, whether the client and the server will use the same kind of tokens or they will use different ones. If both participants use

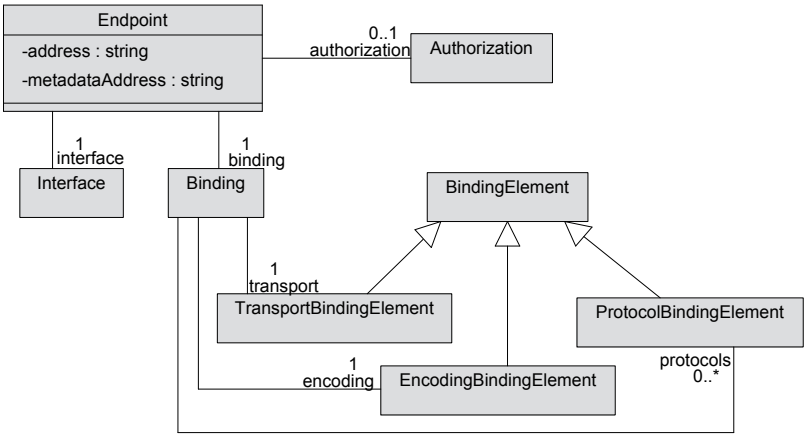

Fig. 5. Endpoint and Binding

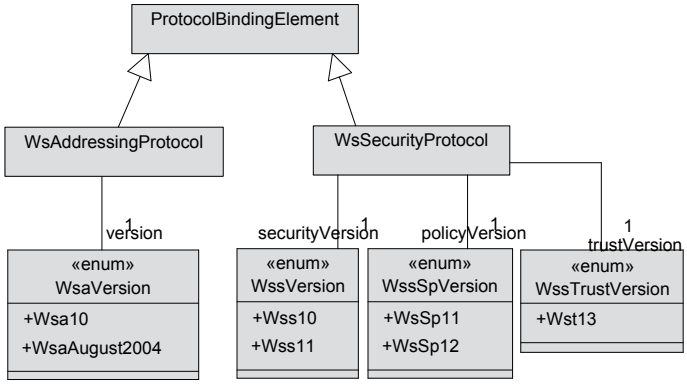

Fig. 6. Protocols

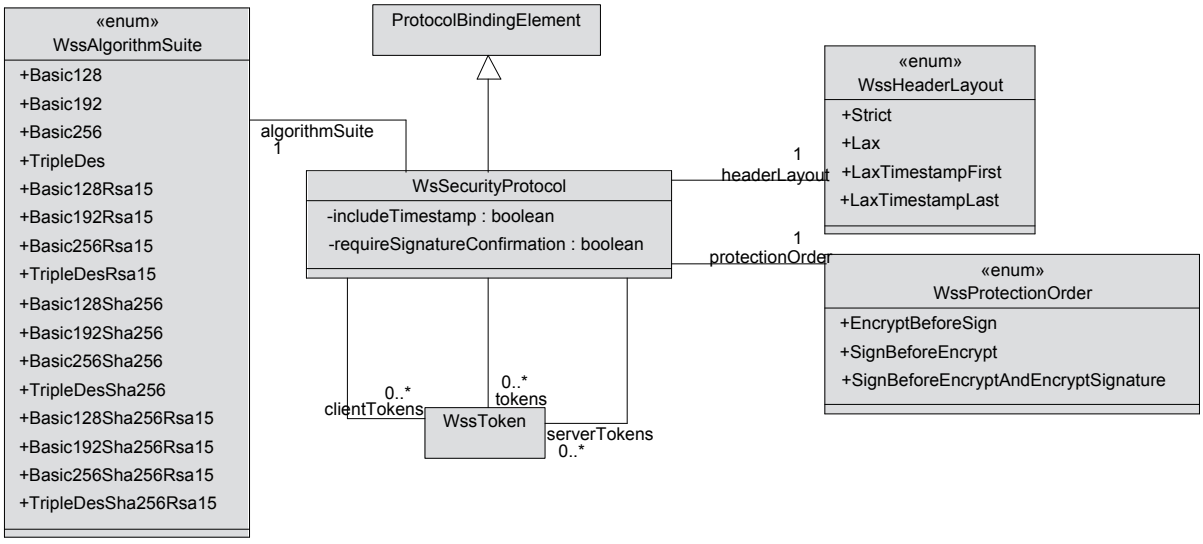

Fig. 7. Security

the same kind of tokens, the security configuration is easier, and only the **`tokens`** attribute has to be specified. If the participants use different kind of tokens, then the client side and the server side tokens must be configured through the **`clientTokens`** and the **`serverTokens`** respectively.

There are multiple possible choices for selecting the security tokens (see Figure 8). The SoaMM metamodel supports username tokens, X.509. certificate tokens, issued tokens, SAML tokens and secure conversation tokens. In this article only **`WssX509Token`** and **`WssSamlToken`** are shown. Through the **`tokenInclusion`** it can also be specified when the different tokens should be included during the communication between the client and the service: always, once or never. The tokens **`WssIssuedToken`** and **`WssSamlToken`**

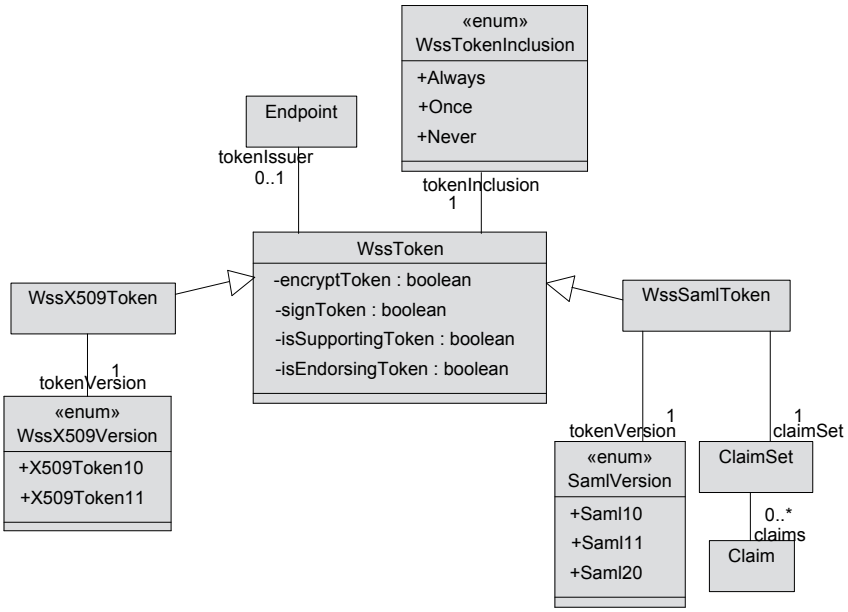

Fig. 8. Security tokens

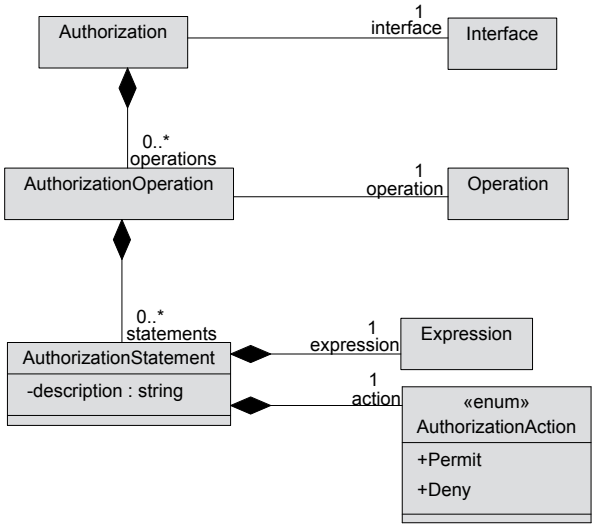

Fig. 9. Authorization contract for access control

are issued by a security token service (STS). In these cases the endpoint of this STS should also be specified through the **`tokenIssuer`**.

An X.509. certificate token contains an X.509. certificate or a reference to such a certificate. In this case only the version of the token must be specified.

A SAML token contains claims based SAML assertions issued by a security token service. In this case the SAML version, the required claims and the token issuer have to be specified.

An authorization contract (**`Authorization`**) specifies the access rights for the operations of a service interface (see Figure 9). Each service operation (**`Operation`**) is implemented by an authorization operation (**`AuthorizationOperation`**) that contains authorization statements (**`AuthorizationStatement`**). An authorization statement either permits or denies access to the service operation, depending on the result of the boolean expression attached to it.

The metamodel for expressions in SoaMM is shown in Figure 10. It is similar to the expressions in .NET [18]. The advantage of this metamodel is that it is independent of the different programming languages, hence, models conforming to this metamodel can be translated to C# and Java, too. An expression tree contains nodes. A node in the expression tree may be an instance of one of the following meta-classes shown in Figure 10 and explained in Table 1.

The methods that can be used on built-in types are the same as the .NET methods on the same .NET types, including the extension methods of the IEnumberable interface [19].

4 Example

In general, the security protocols have the most complicated configuration, especially if security token services are also involved. It is even harder to set the appropriate properties if different products from different vendors are used. This section shows an example how easy it is to configure a scenario with a security token service (STS) using the proposed SoaMM metamodel and its textual concrete syntax called SOAL.

The example is based on the sample introduced in [3]. The scenario (see Figure 11) is the following. A **client** would like to buy wine from a **webshop**. In order to be allowed to do this, he must prove that he is an adult. The proof is presented as a SAML security token issued by a trusted third party, the **security token service (STS)**. The client authenticates itself at the STS with his X.509. certificate and receives a SAML token containing his birth date signed by the STS. The webshop checks the birthdate in the SAML token and decides whether to allow the transaction or not.

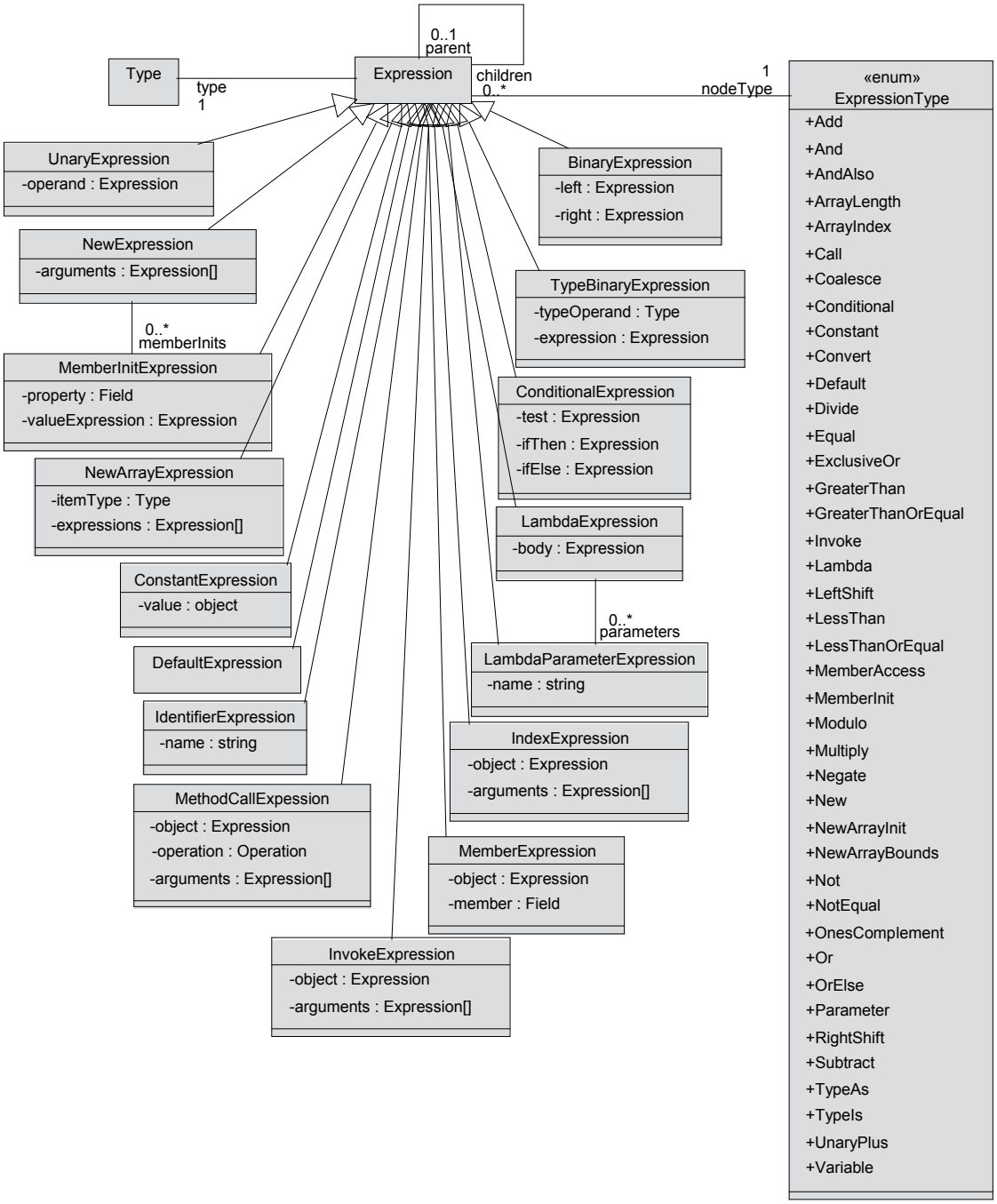

Fig. 10. Expressions

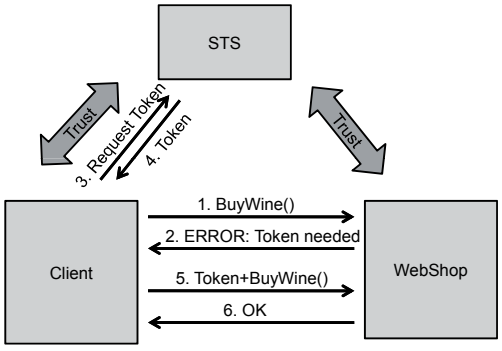

Fig. 11. WS-Trust sample

Tab. 1. Node types of the expression tree

Meta-class	NodeType	Description
UnaryExpression	ArrayLength	An operation that obtains the length of a one-dimensional array.
	Convert	A type-conversion operation.
	Negate	An arithmetic negation operation.
	Not	A bitwise complement or logical negation operation.
	OnesComplement	A ones complement operation.
	TypeAs	An explicit type-conversion in which null is supplied if the conversion fails.
	UnaryPlus	A unary plus operation.
BinaryExpression	Add	An addition operation.
	Divide	A division operation.
	Modulo	An arithmetic remainder operation.
	Multiply	A multiplication operation.
	Power	A mathematical operation that raises a number to a power.
	Subtract	A subtraction operation.
	And	A bitwise or logical AND operation.
	Or	A bitwise or logical OR operation.
	ExclusiveOr	A bitwise or logical XOR operation.
	LeftShift	A bitwise left-shift operation.
	RightShift	A bitwise right-shift operation.
	AndAlso	A conditional AND operation that evaluates the second operand only if the first operand evaluates to true.
	OrElse	A conditional OR operation that evaluates the second operand only if the first operand evaluates to false.
	Equal	A node that represents an equality comparison.
	NotEqual	An inequality comparison.
	GreaterThanOrEqual	A ”greater than or equal to” comparison.
	GreaterThan	A ”greater than” comparison.
	LessThan	A ”less than” comparison.
	LessThanOrEqual	A ”less than or equal to” comparison.
	Coalesce	A null coalescing operation.
	ArrayIndex	An indexing operation in a one-dimensional array.
NewExpression	New	An operation that calls a constructor to create a new object. Fields can be initialized by MemberInitExpressions.
MemberInitExpression	MemberInit	Binds a value to a field of a composite type.
NewArrayExpression	NewArrayInit	An operation that creates a new one-dimensional array and initializes it from a list of elements.
	NewArrayBounds	An operation that creates a new array, in which the bounds for each dimension are specified.
TypeBinaryExpression	TypeIs	A type test.
ConditionalExpression	Conditional	A conditional operation with a then and an else branch.
LambdaExpression	Lambda	A lambda expression.
LambdaParameterExpression	Parameter	A lambda parameter.
ConstantExpression	Constant	A constant value.
DefaultExpression	Default	A default value.
IdentifierExpression	Variable	A variable reference.
IndexExpression	Index	An index operation.
MemberExpression	MemberAccess	An operation that references a field.
MethodCallExpression	Call	A method call.
InvokeExpression	Invoke	An operation that invokes a lambda expression.

The SOAL code describing the system is the following (note that **ISecurityTokenService** is a pseudo interface identifying an STS):

```
namespace WineShoppingSample
{
  [Uri("http://www.example.com/webshop/SampleClaims")]
  claims SampleClaims {
    DateTime BirthDate;
  }
  interface IWebShop {
    bool BuyWine();
  }
  authorization WebShopAuth : IWebShop {
    bool BuyWine() {
      requires SampleClaims.BirthDate;
      deny "Underage people cannot buy wine." {
        DateTime.Now < SampleClaims.BirthDate.AddYears(18);
      }
      permit;
    }
  }
  binding WebShopBinding {
    transport HTTP;
    encoding SOAP;
    protocol WsAddressing;
    protocol WsSecurity {
      serverTokens { token WssX509Token; }
      clientTokens {
        token WssSamlToken {
          tokenIssuer Sts;
          claims SampleClaims.BirthDate;
        }
      }
    }
  }
  binding StsBinding {
    transport HTTP;
    encoding SOAP;
    protocol WsAddressing;
    protocol WsSecurity {
      tokens { token WssX509Token; }
    }
  }
  endpoint WebShop : IWebShop {
    binding WebShopBinding;
    authorization WebShopAuth;
    address "http://www.example.com/webshop/ws";
  }
```

```
  endpoint Sts : ISecurityTokenService {
    binding StsBinding;
    address "http://www.example.com/webshop/sts";
  }
}
```

A claim (**Claim**) has a name (end of the *Uri* in WS-Federation, *AttributeName* in SAML 1.1 and the end of *Name* in SAML 2.0), a URI (start of the *Uri* in WS-Federation, *AttributeNamespace* in SAML 1.1 and the start of *Name* in SAML 2.0) and a type (none in WS-Federation, *xsi:type* in SAML 1.1 and SAML 2.0).

In the above example the claim has the following attributes:

- name=*BirthDate*
- uri=*http://www.example.com/webshop/SampleClaims/BirthDate*
- type=*DateTime*

This claim is mapped to the different claims based identity standards as:

WS-Federation:

```
<fed:ClaimType
      Uri="http://www.example.com/webshop/SampleClaims/BirthDate">
</fed:ClaimType>
```

SAML 1.1:

```
<saml1:AttributeStatement>
  <saml1:Attribute AttributeName="BirthDate"
     AttributeNamespace="http://www.example.com/webshop/SampleClaims">
    <saml1:AttributeValue xsi:type="xs:dateTime">...</saml1:AttributeValue>
  </saml1:Attribute>
</saml1:AttributeStatement>
```

SAML 2.0:

```
<saml2:AttributeStatement>
  <saml2:Attribute
      Name="http://www.example.com/webshop/SampleClaims/BirthDate"
      FriendlyName="BirthDate">
  <saml2:AttributeValue xsi:type="xs:dateTime">...</saml2:AttributeValue>
  </saml2:Attribute>
</saml2:AttributeStatement>
```

XACML Attribute:

```
<xacml:Attribute IncludeInResult="true"
   AttributeId="http://www.example.com/webshop/SampleClaims/BirthDate">
    <xacml:AttributeValue DataType="xs:dateTime">...</xacml:AttributeValue>
</xacml:Attribute>
```

From the authorization **WebShopAuth** a C# class is generated, which is used as a decorator on the service class. This code provides the desired access control for the service:

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Web;
using Microsoft.IdentityModel.Claims;
using System.Threading;
```

```
namespace WineShoppingSample
{
    public class WebShopAuth : IWebShop
    {
        private IWebShop inner;

        public WebShopAuth(IWebShop
        inner)
        {
            this.inner = inner;
        }

        public bool BuyWine()
        {
            IClaimsPrincipal principal = IClaimsPrincipal)Thread.CurrentPrincipal;
            IClaimsIdentity identity = (IClaimsIdentity)principal.Identity;
            ClaimCollection claims = identity.Claims;
            Claim SampleClaims_BirthDate_Claim = claims.FirstOrDefault(
               c => c.ClaimType = SampleClaims.BirthDateClaimType);
            if (SampleClaims_BirthDate_Claim == null)
            {
                throw new Exception(string.Format("Claim '{0}' is required.",
                   SampleClaims.BirthDateClaimType));
            }
            DateTime SampleClaims_BirthDate =
                DateTime.Parse(SampleClaims_BirthDate_Claim.Value);
            if (DateTime.Now < SampleClaims_BirthDate.AddYears(18))
            {
                throw new Exception("Underage people cannot buy wine.");
            }
            return this.inner.BuyWine();
        }
    }
}
```

The code above retrieves the required **BirthDate** claim through the Windows Identity Foundation API, then it extracts the value of the claim and checks whether the condition for denial is met. If the condition is true, it throws an exception signaling an authorization error, otherwise it delegates the call to the actual implementation of the service.

The corresponding XACML code is the following:

```
<?xml version="1.0" encoding="utf-8"?>
<Policy xmlns="urn:oasis:names:tc:xacml:2.0:policy:schema:os"
    PolicyId="BuyWinePolicy"
    RuleCombiningAlgId=
      "urn:oasis:names:tc:xacml:1.0:rule-combining-algorithm:deny-overrides">
  <Target>
    <Subjects>
      <AnySubject/>
    </Subjects>
    <Resources>
      <ResourceMatch MatchId="urn:oasis:names:tc:xacml:1.0:function:string-equal">
        <AttributeValue DataType="http://www.w3.org/2001/XMLSchema#string">
```

```
            http://www.example.com/webshop/ws
          </AttributeValue>
          <ResourceAttributeDesignator
              DataType="http://www.w3.org/2001/XMLSchema#string"
              AttributeId="urn:oasis:names:tc:xacml:1.0:resource:resource-id"/>
        </ResourceMatch>
      </Resources>
      <Actions>
        <ActionMatch MatchId="urn:oasis:names:tc:xacml:1.0:function:string-equal">
          <AttributeValue DataType="http://www.w3.org/2001/XMLSchema#string">
            http://www.example.com/webshop/ws/BuyWine
          </AttributeValue>
          <ActionAttributeDesignator DataType="http://www.w3.org/2001/XMLSchema#string"
                AttributeId="urn:oasis:names:tc:xacml:1.0:action:action-id"/>
        </ActionMatch>
      </Actions>
    </Target>
    <Rule RuleId="BuyWineRule" Effect="Deny">
      <Target/>
      <Condition FunctionId="urn:oasis:names:tc:xacml:1.0:function:dateTime-less-than">
        <Apply FunctionId="urn:oasis:names:tc:xacml:1.0:function:time-one-and-only">
          <EnvironmentAttributeSelector
            DataType="http://www.w3.org/2001/XMLSchema#dateTime"
            AttributeId="urn:oasis:names:tc:xacml:1.0:environment:current-dateTime"/>
        </Apply>
        <Apply
           FunctionId=
             "urn:oasis:names:tc:xacml:1.0:function:dateTime-add-yearMonthDuration">
          <SubjectAttributeDesignator
            DataType="http://www.w3.org/2001/XMLSchema#string"
            AttributeId="http://www.example.com/webshop/SampleClaims/BirthDate"/>
          <AttributeValue DataType="http://www.w3.org/2001/XMLSchema#dateTime">
            18Y
          </AttributeValue>
        </Apply>
      </Condition>
    </Rule>
    <Rule RuleId="FinalRule" Effect="Permit"/>
  </Policy>
```

The SOAL code in the above example is merely 47 lines long and it is easy to read. If the default values for the omitted parameters are not suitable for us, they can also be overridden in the code. From this code our SOAL compiler builds a model as an instance of the SoaMM metamodel. This model is then used to generate WSDL, program code and configuration files for the various SOA products. In addition, it couples these files together into projects, which can be directly opened in the SOA products, while only the application logic has to be implemented, that is, the body of the **`BuyWine()`** method. The projects then can be immediately deployed to the appropriate application servers.

It took only six lines to define the condition for the access control of the webshop web service in SOAL. The corresponding XACML policy is a complex 49 lines long XML description, which is hard to write and maintain by hand. Although this XACML policy can be generated from models conforming to the SoaMM metamodel, a lot of SOA frameworks (e.g. Microsoft WCF) still have no support for XACML. Therefore, it is more useful to generate code for the specific frameworks, like the authorization class for C# in the above example, while the access control condition can be easily defined and maintained as a SOAL code.

Tab. 2. Productivity of the framework with different examples in number of lines

Application	SOAL	WSDL	C#	.NET config	Java	NB config	Total gen.
STS example	**39**	172	241	53	199	1566	**2231**
WS-* performance test	**2168**	56276	22083	8927	29806	18118	**135210**
Hungarian e-Gov pilot	**146**	614	917	233	1642	2026	**5432**

In addition, the WS-Policy assertions for the WebShop service and for the Sts security token service are themselves longer than the entire SOAL code above. The corresponding WSDL is 172 lines long, and our framework even generates product specific configuration files and source code, too. This means that it is easier to describe and configure distributed SOA systems through SOAL than to do it manually. In addition, our code generator ensures that the configurations of the different SOA products will conform to each other, therefore, the web services implemented in different SOA products will be able to communicate with each other immediately.

5 Evaluation

This section evaluates our SoaMM metamodel, SOAL language and the framework built around them. Our aim is to provide interoperability between SOA products without interfering with their operation. Our framework is only for offline use, that is, generating source code and configuration files. Our framework does not extend the SOA products and it does not provide any runtime components either. Hence, this evaluation only covers development productivity and does not deal with runtime performance issues.

We have examined every detail of the WS-* standards and created sample applications with interoperable configurations in the various SOA products for the WS-* standards. The products under examination were Microsoft WCF, GlassFish ESB, Oracle SOA Suite, IBM WebSphere with RAD, Apache Axis2, Apache CXF and JBossWS. As a result, we have an extensive overview of the peculiarities of the individual SOA products, and we could use these experiences we gained to construct a platform independent metamodel above all WS-* standards and SOA products. The end result is the SoaMM metamodel described in Section 3. There has not yet been such a comprehensive metamodel published until now.

Our proposed SoaMM metamodel and SOAL programming language provides a platform independent description of claims-based access control for distributed systems of web services. This platform independent description is transformed into WSDL files, configuration files, program code and projects for the various SOA products by our code generator. The code generator makes sure that the produced projects are directly openable in the targeted SOA products and they are immediately deployable to the appropriate application servers, while they are also interoperable with each other even between different products of different software vendors. The code generator currently supports Microsoft WCF and GlassFish ESB, however, other products are planned to be supported, too.

As the example in the previous section showed, the proposed SOAL programming language provides a compact, easily readable and maintainable description of a distributed system built from web services. Security protocols are easy to configure even if WS-Federation and SAML are present. This can be beneficial for setting up security in grid systems, too.

The true power of our framework came out in another project we are currently working on. The project is about measuring and predicting the overhead of using different parameter types (int, double, string, etc.) and different WS-* standards for web services in the various SOA products. This task required a large number of web services to be implemented in different combinations of types and WS-* standards and SOA products. As a result, 280 web services had to be implemented per SOA product. The performance overhead is measured not only within a single SOA product but between different SOA products as well. This would not be possible without this framework described in this article.

Another application of our framework was a pilot project for the Hungarian e-Government Infrastructure, where a real-life public administration process (foundation of a private entrepreneurship) had to be implemented using web services and BPEL. The pilot system included the simulation of four government agencies. There were seven web services, a BPEL process and a web site in the pilot system. At first, the implementation of the process and the services was done manually, and it took about 1 month to complete. We published our results and the detailed description of the pilot system in [26]. After our framework was ready, we reimplemented all the services by specifying them in SOAL at first, and then we generated WSDL files, program code and configuration files for the various SOA products we had to use. In this second round we also included the test system of the real Hungarian Electronic Governmental Portal in the pilot system. This second approach took three days, which shows, that our framework can greatly increase productivity in the development of distributed SOA systems.

Table 2 shows the productivity of the framework based on the previous examples. For each application the number of lines is listed for the SOAL source code and also for the generated artefacts (XSD+WSDL, C# code, .NET configuration, Java code, Netbeans configuration, and the total number of generated lines, respectively). It can be seen from the examples how compact SOAL is compared to the configurations of the

SOA products. Changing the service descriptions in SOAL and regenerating the source code and configuration files is more productive than keeping the program code and configurations in sync by hand for each of the various SOA products.

6 Conclusion

Our goal was to provide a top-down development method for distributed systems of web services with WS-* protocols. This goal was inspired by the fact that the different SOA products provide different methods for configuring web services making it difficult to match the various configuration options between SOA products. Although WS-Policy assertions provide a platform independent way for configuration, they are hard to construct and to maintain manually. Setting up security and access control is also a tedious task. Therefore, we proposed a more intuitive modeling approach.

UML itself is not suitable for modeling web service standards, since it lacks most of the concepts required to describe these elements. Although by using stereotypes the task can be managed, WS-* protocols cannot be modeled easily. Semantic web service technologies are not designed for modeling either. Therefore, we proposed a domain specific language for modeling distributed systems of web services.

In this article we presented an extension to our SoaMM metamodel for describing claims-based access control in a platform independent way. We also created a programming language called SOAL to describe models in a textual concrete syntax. We have carefully examined the WS-* standards and the SOA products implementing them to be able to construct a truly platform independent metamodel for modeling web services. There has not yet been such a comprehensive metamodel for claims-based access control for web services published until now.

The framework built around the metamodel contains a compiler from SOAL to SoaMM models, and a code generator to produce configuration files and program code for the various SOA products. Only the application logic has to be implemented, and the projects can be immediately deployed to the appropriate application servers. The framework primarily aids the top-down development of distributed systems. From the compact, easily readable and maintainable SOAL descriptions it can produce hundreds of configuration files and program code for the SOA products. As it was mentioned in the previous section, the framework can be powerful, if a large number of interoperable web services have to be created. The development time of such distributed systems can also be greatly reduced.

We will extend the framework with XACML reverse engineering capabilities so that the descriptions of already existing systems can also be imported into the framework. Another future direction is to support other advanced access control features (e.g. obligations in XACML), too.

Acknowledgment

This work was partially supported by the European Union and the European Social Fund through project FuturICT.hu (grant no.: TAMOP-4.2.2.C-11/1/KONV-2012-0013) organized by VIKING Zrt. Balatonfüred.

References

1 **Alam M., Breu R., Hafner M.,** *Model-Driven Security Engineering for Trust Management in SECTET*. Journal of Software, 2 (1), pp. 47–59, (2007).
DOI: 10.4304/jsw.2.1.47-59

2 **Barker S.,** *The next 700 access control models or a unifying meta-model?* in 'Proceedings of the14th ACM symposiumon Access controlmodels and technologies'. ACM, pp. 187–196, (2009).
DOI: 10.1145/1542207.1542238

3 **Bertocci V.,** *WS-Trust – Under the Hood.* (2006) [Online Video]. 4th October. Available from: http://channel9.msdn.com/Shows/Going+Deep/Vittorio-Bertocci-WS-Trust-Under-the-Hood [Accessed: 19th November 2013].

4 **Chen T-Y.,** *Knowledge sharing in virtual enterprises via an ontology-based access control approach*. Computers in Industry, 59 (5), pp. 502–519, (2008).
DOI: 10.1016/j.compind.2007.12.004

5 **De Castro V., Marcos E., Vela B.,** *Representing WSDL with Extended UML*. Revista Colombiana de Computación, (2004).

6 **Dumez C., Nait-Sidi-Moh A., Gaber J., Wack M.,** *Modeling and Specification of Web Services Composition Using UML-S*. in 'Proceedings of the 2008 4th International Conferenceon Next Generation Web Services Practices, NWESP '08.' Washington: IEEE Computer Society. pp. 15–20, (2008).
DOI: 10.1109/nwesp.2008.17

7 **Elgammal A., El-Sharkawi M.,** *Using UML to Model Web Services for Automatic Composition*. International Journal of Software Engineering, 3 (2), pp. 87–113, (2010).

8 **Emig C., Kreuzer S., Abeck S., Biermann J., Klarl H.,** *Model-Driven Development of Access Control Policies for Web Services*. in 'Proceedings of the 9th IASTED International Conference Software Engineering and Applications. (ed.: Khoshgoftaar T.)' Orlando: ACTA Press, pp.165–171, (2008).

9 **Ferraiolo D., Atluri V.,** *A meta model for access control*. in 'Proceedings of the 13th ACM symposium on Access control models and technologies'. ACM. pp. 153–154, (2008).
DOI: 10.1145/1377836.1377860

10 **Giordano M., Polese G., Scanniello G., Tortora G.,** *A system for visual role-based policy modelling*. Journal of Visual Languages & Computing, 21 (1). pp. 41–64, (2010).
DOI: 10.1016/j.jvlc.2009.11.002

11 **Gronmo R., Skogan D., Solheim I., Oldevik J.,** *Model-Driven Web Services Development*. in 'The 2004 IEEE International Conferenceone-Technology, e-Commerce and e-Service(EEE-04)' (2004).

12 **Hafner M., Memon M., Alam M.,** *Modeling and Enforcing Advanced Access Control Policies in Healthcare systems with Sectet*. in 'Lecture Notes in Computer Science, Springer, pp. 132–144, (2008).
DOI: 10.1007/978-3-540-69073-3_15

13 **Haller A., Cimpian E., Mocan A., Oren E., Bussler C.,** *WSMX- a semantic service-oriented architecture*. in 'Proceedings of the IEEE International Conference on Web Service (ICWS 2005)' pp. 321–328, (2005). DOI: 10.1109/icws.2005.139

14 **Jiague M. E., Milhau J., Laleau R., Gervais F.,** *Specification of translation rules from a PIM to a PSM for access control policies models*. in 'Programme Systemes Embarqueset Grandes Infrastructures - "ARPEGE" Appel a projets general' (2011)

15 **Jin X.,** *Applying Model Driven Architecture approach to Model Role Based Access Control System*. Thesis submitted to the Faculty of Graduate and Postdoctoral Studies in partial fulfillment of the requirements for degree of Master of Science in System Science. University of Ottawa, (2006).

16 **Klusch M.,** *Semantic Web Service Coordination*. in 'CASCOM - Intelligent Service Coordination in the Semantic Web. (eds.: Schumacher M., Schuldt H., Helin H.) Springer, pp. 59-104, (2008). DOI: 10.1007/978-3-7643-8575-0_4

17 **Kolovksi V., Parsia B., Katz Y., Hendler J.,** *Representing Web Service Policies in OWL-DL*. in 'International Semantic Web Conference (ISWC)' pp. 6–10, (2005).

18 Microsoft Developer Network, Expression Type Enumeration. (2013) [Online] Available from: http://msdn.microsoft.com/en-us/library/bb361179.aspx. [Accessed: 16th April 2013]

19 Microsoft Developer Network, IEnumerable <T> Interface. (2013) [Online] Available from: http://msdn.microsoft.com/en-us/library/9eekhta0.aspx. [Accessed: 16th April 2013]

20 **Mouelhi T., Fleurey F., Baudry B., Le Traon Y.,** *A Model-Based Framework for Security Policy Specification, Deployment and Testing*. in 'Model Driven Engineering Languages and Systems', Springer, pp. 537–552, (2008). DOI: 10.1007/978-3-540-87875-9_38

21 **OASIS,** OASIS eXtensible Access Control MarkupLanguage (XACML) TC. (2013). Available from: http://www.oasis-open.org/committees/tc_home.php?wg_abbrev=xacml. [Accessed: 16th April 2013]

22 **OMG** Documents Associated With Service Oriented Architecture Modeling Language (SoaML), Version 1.0.1. (2013) Available from: http://www.omg.org/spec/SoaML/Current. [Accessed: 16th April 2013]

23 **Shafiq O., Moran M., Cimpian E., Mocan A., Zaremba M., Fensel D.,** *Investigating Semantic Web Service Execution Environments: A Comparison between WSMX and OWL-S Tools*. in 'Internet and Web Applications and Services, International Conference'. (2007)

24 **Simon B., Goldschmidt B.** *A Human Readable Platform Independent Domain Specific Language for WSDL*. in 'Computer and Information Science'. pp. 529–536, (2010). DOI: 10.1007/978-3-642-14292-5_54

25 **Simon B., Goldschmidt B., Budai P., Hartung I., Kondorosi K., Laszlo Z., Risztics P.,** *A Metamodel of the WS-Policy Standard Family*. in 'ICDS 2011, The Fifth International Conference on Digital Society.' pp. 57–62, (2011).

26 **Simon B., Laszlo Z., Goldschmidt B.,** *SOA Interoperability, A Case Study*. in 'Proceedings of the IADIS International Conference Informatics.' pp. 131–138, (2008).

27 **Sriharee N., Senivongse T., Verma K., Sheth A.,** *On Using WS-Policy, Ontology, and Rule Reasoning to Discover Web Services*. in 'Intelligence in Communication Systems (eds.: Aagesen F. A., Anutarya C., Wuwongse V.)', Volume 3283, Berlin / Heidelberg: Springer, (2004). DOI: 10.1007/978-3-540-30179-0 22.

28 **Verma K., Akkiraju R., Goodwin R.,** *Semantic Matching of Web Service Policies*. in 'Proceedings of the Second Workshop on SDWP.' pp. 79–90, (2005).

29 **W3C. MEMBER SUBMISSION** OWL-S: Semantic Markup for Web Services. (2012) Available from: http://www.w3.org/Submission/OWL-S/. [Accessed: 29th January 2012]

30 **W3C** Semantic Annotations for WSDL and XML Schema (SAWSDL). (2012). Available from: http://www.w3.org/TR/sawsdl/. [Accessed: 29th January 2012]

31 **Wolter C., Menzel M., Schaad A., Miseldine P., Meinel C.,** *Model-driven business process security requirement specification*. Journal of Systems Architecture. 55 (4). pp. 211–223, (2009). DOI: 10.1016/j.sysarc.2008.10.002

32 Web Service Modeling Ontology, ESSI WSMO working group. (2012) Available from: http://www.wsmo.org/. [Accessed: 29th January 2012]