Web Services as a New Approach to Distributing and Coordinating Semantic-based Verification Toolkits

Michael Baldamus[1], Jesper Bengston[1], GianLuigi Ferrari[2] Roberto Raggi[2]

[1]Department of Information Technology, Uppsala University
[2]Dipartimento di Informatica, Università di Pisa

Abstract

We describe a coordination–oriented way of integrating semantic-based verification toolkits, being motivated by a three–pronged observation:

- There is a *potential* for integration since verification toolkits are typically based on well–understood mathematical notions.
- There is a *need* for and a *profit to be gained* from integrating such tools since they are often of rather special and, at the same time, complementary functionality.
- There is a *problem* with such integration since the formal verification community prefers decentralisation and, at the same time, the available resources for working on portability are often limited anyway.

Our approach is distributed, thereby moving the issue to the realm of coordination. To this end, we argue that toolkit integration, while not new, can be given a new life by applying the emerging paradigm of web services. We argue that web services can not only serve as a technological platform for our purposes but, indeed, also as a coordination framework, addressing all the basic issues present in the situation we are dealing with. We give an account of a prototype implementation of these concepts.

1 Introduction

We describe a coordination–oriented way of integrating semantic-based verification toolkits, being motivated by a three–pronged observation:

- There is a *potential* for integration since verification toolkits are typically based on well–understood mathematical notions.

This work is funded as part of the Profundis project within the Global Computing Initiative of the European Union.

- There is a *need* for and a *profit to be gained* from integrating such tools since they are often of rather special and, at the same time, complementary functionality.

- There is a *problem* with such integration since the formal verification community prefers decentralisation and, at the same time, the available resources for working on portability are often limited anyway.

Our approach is distributed, thereby moving the issue to the realm of coordination. For two fundamental reasons, this way of dealing with the problem seems to be natural:

1. Distribution addresses the reluctance of the formal methods community to engage in any centralised effort at tool integration. Some tools are but others are not or only to a limited degree portable or even available for download, and this situation is not likely to change anytime soon, for a variety of reasons. Distribution has some intrinsic advantages over centralised approaches also, such as availability, reconfigurability and openness.

2. Distributed tool integration seems to be best–viewed from coordination standpoint, since we are aiming at what we call *deep integration*: There shall be automatic means for assigning sub–tasks belonging to any verification run to those node that are most appropriate for solving them. That in turn requires both a platform and a language that allow users to direct what should be done where, at different levels of abstraction. This requirement can be regarded as almost synonymous with coordination.

   Shallow integration would just mean that users could choose between different tools for tackling any verification problem as a whole.

In the last couple of years distributed applications over the world–wide web, Web for short, such as file sharing, have indeed attained wide popularity, spawning an increasing demand for evolutionary paradigms in designing and controling them. Uniform mechanisms have been developed for handling computing problems which involve a large number of heterogeneous components that are physically distributed and (inter)operate autonomously.

These developments have begun to coalesce around a paradigm where the Web is exploited as a *service distributor*. A service in this sense is not a monolithic Web server but rather a component available over the Web that others might use to develop other services. Conceptually, Web services are stand–alone components in the Internet. Each Web service has an interface accessible through standard protocols and, at the same time, describing the interaction capabilities of the service. Applications over the Web are developed by combining and integrating Web services. Moreover, no Web service has pre–existing knowledge of what interaction with other Web services may occur.

One main idea of the work presented here is to make semantic-based verification toolkits available as Web services, using standards such as XML, WSDL and SOAP. Another main idea is to establish directories for publishing such Web services. On this basis, then, we will be able to play out the fact that verification tools are semantics–based, as that characteristic should allow us to realize powerful automated matching facilities for services. It is not obvious how more general directory concepts such as UDDI could provide such automation, as their very generality precludes them from relying on formal semantics and, therefore, forces them to count on human intervention in what is called *technical* [service] *discovery* [1].

Verification web services plus automated verification web service directories thus provide a platform within our concept for distributed, deeply integrated and therefore coordinated verification. On top of it, there is a verification scripting facility so that users are able to specify how the sub–tasks within any verification run are to be carried out. Beyond the current prototype, we envisage that an important role in that will eventually be played by the trading functionality embedded in semantics–based directories. Specifically, there should be different levels of abstraction in proof scripting, namely goal–oriented, algorithm–oriented, tool–oriented and node–oriented levels.

The rest of this paper presents the prototype of an environment which integrates and coordinates different verification tools via the Web as a service distributor. The development of the verification environment has been performed inside the Profundis project (see URL `http://www.it.uu.se/profundis`) within the Global Computing Initiative of the European Union. We called *Profundis WEB*, shortly PWeb.

The PWeb implementation has been conceived to support reasoning about the behaviour of systems specified in some dialect of the π-calculus. It supports the dynamic integration of several verification techniques (e.g. standard bisimulation checking and symbolic techniques for cryptographic protocols). The PWeb has been designed by targeting also the goal of extending available verification environments (MobilityWorkbench [10], HAL [5]) with new facilities provided as Web services. This has given us the opportunity to verify the effective power of the Web service approach to deal with the reuse and integration of "old" modules.

2 Preliminaries

Over the years several semantic-based verification toolkits have been designed and experimented to formally address some issues raised by software development. The *Concurrency Workbench* [4], for example, was developed at the University of Edinburgh and performs analysis on the Calculus for Communicating Systems (CCS). The Mobility Workbench (MWB) [10], developed at the university of Uppsala, does similar analysis but on the π-calculus. The *History-Dependent Automata Laboratory* (HAL) [5] supports verification of logical formulae expressing properties of the behaviour of π–calculus agents.

Most of the semantic-based verification environments have been developed independently of each other and there is no guarantee that they run on the same platforms. Moreover, there is also a need to run these tools in conjuncture with one another as results from one tool can be used as input for another. For this to be plausible, a platform independent system has to be created which allows different tools running on different computers, potentially on different platforms, to work with each other.

The PWeb, is designed to solve these problems. The PWeb prototype implementation has been conceived to support reasoning about the behaviour of systems specified in some dialect of the π-calculus. The PWeb integrates and coordinate the facilities of some verification toolkits provided as Web services. The MWB and HAL are two of the services of the PWeb. Hereafter, we briefly review the main features of the other services of the PWeb.

TRUST The TRUST toolkit [9, 8] relies on an exact symbolic reduction method, combined with several techniques aiming at reducing the number of interleaving that have to be considered. Authentication and secrecy properties are specified in a very natural way, and whenever an error is found an intruder attacking the protocol is given.

MIHDA The MIHDA toolkit [6] performs state minimization of HD automata. MIHDA has been exploited to perform finite state verification of π-calculus specifications.

STA STA (Symbolic Trace Analyzer) [2] implements symbolic execution of cryptographic protocols. A successful attack is reported in the form of an execution trace that violates the specified property.

3 The PWeb Directory Service

The core of the PWeb is a *directory service*. A PWeb directory service is a component that maps the description of the Web services represented by suitable XML types into the corresponding network addresses. Moreover, it performs the binding of services.

The PWeb directory maintains references to the toolkits it works with. Every toolkit has an end-point in the directory service through the WSDL-specification. As expected, the WSDL-specification describes the interaction capabilities of the toolkit; namely which methods are available and the types of their inputs and outputs. In other words, the WSDL-specification describes what a service can do, how to invoke it and the supported XML types (more precisely the XML Schema definitions XSD). For instance, the WSDL-specification of MIHDA provides the description of the `reduce` method. The invocation of this method on a given HD-automata performs the state minimisation of the HD-automata. The WSDL-description of the MIHDA toolkit is displayed in Figure 1.

```
<?xml version="1.0" encoding="UTF-8"?>
<definitions
    name="Mihda"
    targetNamespace="http://jordie.di.unipi.it:8080/pweb/Mihda.wsdl"
    xmlns="http://schemas.xmlsoap.org/wsdl/"
    xmlns:soap="http://schemas.xmlsoap.org/wsdl/soap/"
    xmlns:tns="http://jordie.di.unipi.it:8080/pweb/Mihda.wsdl"
    xmlns:xsd="http://www.w3.org/2001/XMLSchema"
    xmlns:xsd1="http://jordie.di.unipi.it:8080/pweb/schemas">
    <import namespace=''http://http://jordie.di.unipi.it:8080/pweb/schemas''
               location=''http://jordie.di.unipi.it:8080/pweb/hds_over_pi.xsd''/>
    <types>
        <xsd:schema
            targetNamespace="http://jordie.di.unipi.it:8080/pweb/Mihda.xsd"
            xmlns="http://schemas.xmlsoap.org/wsdl/"
            xmlns:SOAP-ENC="http://schemas.xmlsoap.org/soap/encoding/"
            xmlns:xsd="http://www.w3.org/2001/XMLSchema"
            xmlns:xsd1="http://jordie.di.unipi.it:8080/pweb/Mihda.xsd"> </xsd:schema>
    </types>
    <message name="ReduceRequest">
        <part name="contents" type="xsd1:hds_over_pi"/>
    </message>
    <message name="ReduceResponse">
        <part name="return" type="xsd1:hds_over_pi"/>
    </message>
    <portType name="MihdaPortType">
        <operation name="Reduce">
            <documentation>Minimize the automata</documentation>
            <input message="tns:ReduceRequest"/>
            <output message="tns:ReduceResponse"/>
        </operation>
    </portType>
    <binding name="MihdaBinding" type="tns:MihdaPortType">
        <soap:binding style="rpc" transport="http://schemas.xmlsoap.org/soap/http"/>
        <operation name="Reduce">
            <soap:operation soapAction="connect:Mihda:MihdaPortType#Reduce"/>
            <input>
                <soap:body use="literal"/>
            </input>
            <output>
                <soap:body use="literal"/>
            </output>
        </operation>
    </binding>
    <service name="Mihda">
        <port binding="tns:MihdaBinding" name="MihdaPort">
            <soap:address location="http://jordie.di.unipi.it:8080/pweb/mihda"/>
        </port>
    </service>
</definitions>
```

Figure 1: The MIHDA WSDL-specification

Notice that there is nothing preventing several directory services to connect to the same toolkits, or to include references to other directory services. Hence, the PWeb is basically a peer-to-peer system.

The PWEB directory servic has two main facilities. The `publish` facility is invoked to make available a semantic-based toolkit as Web service. The `query` facility, instead, is used by applications to discover which are the services available. Notice that the `query` provides service discovery mechanisms: it yields the list of services that match the parameter (i.e. the XSD type describing the kind of services we are interested in). The current prototype implementation of the PWeb dierectory service can be exercized on-line at the URL `http://jordie.di.unipi.it:8080/pweb`.

The service discovery mechanisms is exploited by the `trader` engine. The trader engine manipulates pool of services distributed over several PWeb directory services. It can be used to obtain a Web service of a certain type and to bind it inside the application. The `trader` engine gives to the PWeb directory service the ability of finding and binding at run-time web services without "hard-coding" the name of the web service inside the application code. In other words, the `trader` engine is the resource discovery mechanism for PWeb directory services.

The following code describes the implementation of a simple trader for the PWeb directory.

```
import Trader

offers = Trader.query( "reducer" )

mihda = offers[ 0 ]             # choose the first

offers = Trader.query( "model-checking" )

hal = find_neighbor(offers)     # choose the service only among neighbors

offers = Trader.query( "bisimulation-checking" )

mwb = offers[ 0 ]               # choose the first
```

The `trader` engine allows one to hide network details in the service coordination code. A further benefit is given by the possibility of replicating the services and maintaining a standard access modality to the Web services under coordination. For instance, the code

```
offers = Trader.query("security checker" )
```

can be used to obtain a polymorphic orchestration code that, at run–time, is able to find, bind and finally invoke any service registered as "security checker". In the PWeb prototype implementation both TRUST and STA are registered as security checkers.

The PWeb directory service is powered using Zope [11]. Zope is a (open source) framework for building web applications and is designed to allow administrators to build complex and easily maintainable web servers with a minimum amount of work. Dynamic content is supported through the use of databases which in turn can be updated through web interfaces. Zope is also highly configurable and fully object oriented. New objects can be added and inherited from if the need arises allowing for existing features to be tailored to the users need. There is also a robust security system which allows administrators to manage user privileges. One of the main advantages of Zope is that it is portable. It runs on a variety of machines infrastructures including Windows 2000/NT/XP, Linux, Solaris and Max OS X.

As a final remark we want to point out that the `trader` engine provides facilities which are similar to the CORBA trader. The CORBA trader is used to query object infrastructures for specific applications and components.

3.1 Querying XML Types

The PWeb directory service includes a database of XML types which keeps track of the relationships among the XSD types which can be exploited as arguments of messagges. Whenever a new XSD type is added to the PWeb directory service (e.g as a result of a service registration), it is compared with the existing XSD types and its relationships with the other registrered types are stored in the database. Whenever an application performs a query, the trader engine will provide a list of types which are compatible with the type of the query. In the prototype implementation, this is obtained by a simple script code written in Python.

We are currently investigating more expressive and powerful mechanisms for querying XML types. In particular, we started some experiments in using programming languages specifically designed to manipulate and querying XML data [7, 3].

4 Lessons Learned

We started our experiment with the goal of understanding whether the Web service metaphor could be effectively exploited to integrate in a distributed and coordinated fashion semantics–based verification toolkits. In this respect, the prototype implementation of the PWeb significant example.

Our approach adopts a service orchestration model whose main advantage resides in reducing the impact of network dependencies and of dynamic addition/removal of Web services by the well–identified notions of directory of services and trader engine. To the best of our knowledge, this is the first verification environment that specifically addresses the problem of exploiting Web services.

The service orchestration mechanisms presented in this paper, however, have some disadvantages. In particular, they do not exploit the full expressive power of SOAP to handle types and signatures. For instance, the so called "version consistency" problem (namely the client program can work with one version of the service and not with others) can be solved by types and signatures.

References

[1] UDDI Technical White Paper, 2000.

[2] M. Boreale and M. Buscemi. *STA, a Tool for the Analysis of Cryptographic Protocols (Online version).* Dipartimento di Sistemi ed Informatica, Università di Firenze, and Dipartimento di Informatica, Università di Pisa,, http://www.dsi.unifi.it/ boreale/tool.html, 2002.

[3] Luca Cardelli and Giorgio Ghelli. Tql: A query language for semistructured data based o n the ambient logic. *To appear in Mathematical Structures in Computer Science*, 2003.

[4] Rance Cleaveland, Joachim Parrow, and Bernhard Steffen. The Concurrency Workbench: A Semantics-Based Tool for the Verification of Concurrent Systems. *ACM Transactions on Programming Languages and Systems*, 15(1):36–72, January 1993.

[5] G. Ferrari, S. Gnesi, Ugo Montanari, and Marco Pistore. A model checking verification environment for mobile processes. *To appear in ACM TOSEM*, 2003.

[6] Gianluigi Ferrari, Ugo Montanari, Roberto Raggi, and Emilio Tuosto. From co-algebraic specification to implementation: the mihda toolkit. In *First International Workshop on Methods for Components and Objects (FMCO)*, Lecture Notes in Computer Science, pages 428–440. Springer-Verlag, 2003.

[7] Haruo Hosoya and Benjamin C. Pierce. Xduce: A typed xml processing language, 2003.

[8] V. Vanackere. *The TRUST protocol analyser.* Lab. Informatique de Marseille, http://www.cmi.univ-mrs.fr/ vvanacke/trust.html, 2002.

[9] V. Vanackere. The trust protocol analyser, automatic and efficient verification of cryptographic protocols. In *Verification Workshop - Verify02*, 2002.

[10] Björn Victor and Faron Moller. The Mobility Workbench — a tool for the π-calculus. In David Dill, editor, *CAV'94: Computer Aided Verification*, volume 818 of *Lecture Notes in Computer Science*, pages 428–440. Springer-Verlag, 1994.

[11] Zope, http://www.zope.org.