Verification On The Web Of Mobile Systems *

Gianluigi Ferrari[1], Stefania Gnesi[2]
Ugo Montanari[1], Roberto Raggi[1]
Gianluca Trentanni[2], and Emilio Tuosto[1]

[1] Dipartimento di Informatica, Università di Pisa
[2] ISTI-CNR, Pisa

Abstract. The vast majority of current available verification environments have been built by sticking to traditional architectural style centralized and without dealing with interoperability and dynamic reconfigurability. In this paper we present a verification toolkit whose design and implementation exploit the Web service architectural paradigm.

1 Introduction

The WEB provides uniform mechanisms to handle computing problems which involve a large number of *heterogeneous* components that are *physically distributed* and *(inter)operate* autonomously. Recently, several software engineering technologies have been introduced to support a programming paradigm where the WEB is exploited as a *service distributor*.

Our issue in this paper is to demonstrate that the design, development and maintenance of semantics-based verification environments are semplified by exploiting WEB services in a modular fashion taking even advantage of the reuse and integration of "old" modules.

As a preliminary answer to this question a prototype version of a verification toolkit has been conceived to support reasoning about the behaviour of mobile processes specified in the π-calculus supporting the dynamic integration of several verification techniques.

2 Service Orchestration

The architecture of the toolkit, called Mihda performing minimization of HD-automata is described in [8], its structure is developed from the co-algebraic formulation of the partition-refinement minimization algorithm and a web interface is avalaible (`http://jordie.di.unipi.it:8080/mihda`).

A semantic-based verification environment for the π-calculus, called *HD Automata Laboratory* (HAL) [1, 2] has been implemented and experimented. HAL is available at `http://fmt.isti.cnr.it:8080/hal`.

* Work partially supported by FET Projects IST-2001-33100 PROFUNDIS and IST-2001-32747 AGILE.

The main issue we have to face consists of making these toolkits accessible and usable via a WEB interface. This is done into two steps: The first step defines the *WEB orchestration interface* which, independently from the implementation technologies, describes the WEB interaction capabilities; the second step defines transforming the program facilities which correspond to publish the orchestration interface on the WEB.

The main programming construct we exploit to program service orchestration is XML-RPC to ensures interoperability among components available over the WEB at the main cost of parsing and serializing XML documents.

In our running example we consider two services, namely the tools HAL and Mihda. The WEB orchestration interface of Mihda provides three interaction capabilities: `Compile` taking a π-calculus agent as input and yielding as output the corresponding HD-automaton; `reduce` performing minimization on HD automata; `Tofc2`: Transforming the Mihda representation of HD-automata into the FC2 format used inside HAL. The WEB orchestration interface of HAL provides three capability: `Check` performing model checking; `unfold` generating a standard automaton out of an HD-automaton; `visualize` allowing to graphically operate over HD-automata. The publication on the WEB of the orchestration interfaces has been performed by exploiting the facilities of the web application server `Zope` providing a comprehensive framework for management of web contents and mechanisms to "publish" information on the WEB.

In our experiment, the service orchestration language is PYTHON whose expressiveness gives us the opportunity of programming service orchestration in the same way traditional programming languages makes use of software libraries. In particular, services are invoked exactly as "local" libraries and all the issues related to data marshaling/unmarshalling and remote invocation are managed by the XML-RPC support. An example of service orchestration is illustrated below to verify a property of a specification, i.e. to test whether a π-calculus process A is a model for a formula F.

```
mihda = Server( "http://jordie.di.unipi.it:8080/mihda/hd" )
hal = Server( "http://fmt.isti.cnr.it:8080/hal" )
hd = mihda.compile( A )
reduced_hd = mihda.reduce( hd )
reduced_hd_fc2 = mihda.Tofc2( reduced_hd )
aut = hal.unfold( reduced_hd_fc2 )
check = hal.check( aut, F )
```

In this few code lines it is clear that the only part of the orchestration that includes network dependencies is represented by the operation that open the connections with the HAL and Mihda servers with the service orchestration program running under WindowsXP, pointing out the interoperability nature of the toolkit.

However, this network dependency can be removed by introducing a further module, namely the *directory of services* together with a simple *trader* facility that perform the binding of services that would allow us to avoid specifying the effective names (and localities) of services into the source code and to dynamically bind services during the execution only on demand, making transparent the distribution of services and allowing to repicate or to re-allocate a service into a new locality without requiring any change into service orchestration programs.

3 Lessons Learned

We started our experiment with the goal of understanding whether the WEB service metaphor could be effectively exploited to develop semantic-based verification environments. In this respect, the prototype implementation of a toolkit supporting verification of mobile processes specified in the π-calculus is a significative 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. To the best of our knowledge, this is the first verification toolkit 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.

We plan to investigate and experiment the .NET framework to design "next generation" semantic-based verification environments.

References

1. G. Ferrari, G. Ferro, S. Gnesi, U. Montanari, M. Pistore, and G. Ristori. An automata based verification environment for mobile processes. In E. Brinksma, editor, TACAS'97, *LNCS* 1217. Springer, Apr. 1997.
2. G. Ferrari, S. Gnesi, U. Montanari, M. Pistore, and G. Ristori. Verifying mobile processes in the HAL environment. In *Proc. 10th International Computer Aided Verification Conference*, pages 511–515, 1998.
3. G. Ferrari, U. Montanari, and M. Pistore. Minimizing transition systems for name passing calculi: A co-algebraic formulation. In M. Nielsen and U. Engberg, editors, *FOSSACS 2002*, volume LNCS 2303, pages 129–143. Springer Verlag, 2002.
4. IBM Software Group. Web services conceptual architecture. In *IBM White Papers*, 2000.
5. U. Montanari and M. Pistore. π-calculus, structured coalgebras and minimal hd-automata. In *Proc. MFCS'2000*, volume 1893 of *LNCS*. Springer, 2000.
6. F. Orava and J. Parrow. An algebraic verification of a mobile network. *Formal Aspects of Computing*, 4(1):497–543, 1992.
7. M. Pistore. *History dependent automata.* PhD thesis, Computer Science Department, Università di Pisa, 1999.
8. E. Tuosto. *Non-Functional Aspects of Wide Area Network Programming.* PhD thesis, Dipartimento di Informatica, Università di Pisa, 2002.