Verification on the WEB

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] IEI-CNR, Pisa

Abstract. Web services allow the components of applications to be highly decentralized, dynamically reconfigurable. Moreover, Web services can interoperate easily inside an heterogeneous network environment. The vast majority of current available verification environments have been built by sticking to traditional architectural styles. Hence, they are centralized and none of them deal with interoperability and dynamic reconfigurability. In this paper we present a verification toolkit whose design and implementation exploit the Web service architectural paradigm. We describe the architectural design and the discuss in detail the current implementation efforts.

1 Introduction

In the last few years distributed applications over the WEB have gained wider popularity. Several systems (e.g. *Gnutella*) have led to an increasing demand of evolutionary paradigms to design and control the development of applications over the WEB. The main advantages of exploiting the WEB as underlying platform can be summarized as follows. 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*. Here, by service we do not mean a monolithic WEB server but rather a component available over the WEB that others might use to develop other services. Conceptually, WEB services [4] are stand-alone components that reside over the nodes of the network. Each WEB service has an interface which is network accessible through standard network protocols and describes the interaction capabilities of the service. Applications over the WEB are developed by combining and integrating together WEB services. Moreover, no WEB service has pre-existing knowledge of what interaction with other WEB services may occur. Moreover, WEB services are highly portable to adapt to a variety of infrastructures.

In a WEB service scenario, the development of applications can be characterized in terms of the following steps:

1. *Publishing* WEB services;

2. *Finding* the required WEB services;
3. *Binding* the WEB services inside the application;
4. *Running* the application assembled from WEB services.

Indeed, in the next few years evolutionary in-development technologies based on HTTP/XML plus

1. remote invocation (e.g. XML-RPC SOAP),
2. directory and service binding (e.g. UDDI, trader),
3. language to express service features (e.g. WSDL)

will become the standard functional platform to programming applications over the WEB.

The vast majority of currently available semantic-based verification environments have been designed and implemented by sticking to traditional paradigms. Basically, verification environments are monolithic specialized servers which do not easily support interoperability and dynamic reconfiguration. We argue that the research activity in the field of formal verification can take advantage of the shift from the traditional development paradigms to other paradigms which better accomodate and support WEB services. The present paper intends to explore this issue.

In particular, the present paper intends to address the following issue. Can we simplify the design, development and maintenance of semantics-based verification environments in a modular fashion by exploiting WEB services?

This paper provides a preliminary answer to this question by presenting the prototype version of a verification toolkit which directly exploits the WEB as a service distributor. The toolkit has been conceived to support reasoning about the behaviour of mobile processes specified in the π-calculus and it supports the dynamic integration of several verification techniques. In other words, the toolkit can be dynamically encompass a variety of verification techniques. Finally, the toolkit has been developed by targeting also the goal of extending an available verification environment (HAL [1, 2]) 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.

The paper is structured as follows. Section 2 provides the reader with the necessary background information. By several examples, Section 3 introduces the software architecture of the toolkit and describes the current prototype implementation. Finally, Section 4 discusses the strengths and limitations of our proposal.

2 Preliminaries

History Dependent automata (HD-automata in brief) have been introduced in [7, 5] with the aim of providing an automata-like model that supports effective and efficient techniques to certify properties of distributed systems specified by suitable name passing calculi (e.g. the π-calculus). Name passing primitives are

simple but expressive: channel names can be created, communicated and they are subjected to sophisticated scoping rules. However, the possibility of dynamically generating new names leads also to a much more complicated theory. In particular, the usual operational models are infinite-state and infinite branching and therefore they are not amenable for finite-state verification.

Similarly to ordinary automata, HD-automata are made out of states and labeled transitions. However, states and transitions are equipped with local names which are no longer dealt as syntactic components of labels but become explicit semantic components of the model. Indeed, HD automata can be viewed as automata on top of a permutation algebra of states (technically HD-automata are coalgebras over the permutation algebra). The permutation algebra describes the effect of name permutations (i.e. renaming) on state transitions. This information is sufficient to describe in a semantically correct way the creation, communication, and deallocation of names in the case of name-passing calculi.

State transitions of HD-automata have an intuitive graphical representation. For instance, Figure 1 depicts a transition from source state s to destination state d. The transition exposes two names: Name 2 of s and a newly generated name 0. State s has three names, 1, 2 and 3 while d has two names 4 and 5 which correspond to name 1 of s and to the new name 0, respectively. Notice that names 3 is discharged along such transition.

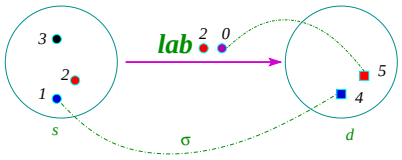

Fig. 1. A HD-automaton transition

The names of the states occurring in a computation are basically equipped with the "history" of the name associations performed during the computation. Clearly, if a state is reached in two different computations, different histories could be assigned to its names. *Symmetries* , i.e. permutation groups of names, are used to declare the name correspondences which do not affect behaviours. Symmetries are necessary to define the minimal HD-automata thus providing a more compact representation of the state space of the computations [5, 3]

Hence, HD-automata provide a finite state, finite branching representation of the behaviour of name passing calculi. The finiteness property given by the HD-automata has been exploited to automatize the check of behavioral properties. Indeed, a semantic-based verification environment for the π-calculus, called *HD Automata Laboratory* (HAL) [1, 2] has been implemented and experimented. HAL is written in C++ and compiles with the GNU C++ compiler (the GUI is written in Tcl/Tk), and runs on SUN stations (under SUN-OS).

HAL supports verification of logical formulae expressing properties of the behaviour of π-calculus agents. The construction of the HAL model checker facility is done in two stages. We first introduce an high level logic with modalities indexed by π-calculus actions and we provide a mapping which translates logical formulae into a classical modal logic for standard automata. The distinguished and innovative feature of the approach is that translation mapping is driven by the finite state representation of the system (the π-calculus process) to be verified.

HAL has been used to perform the verification of several case studies, as for example the GSM handover protocol [6]. However, a main limitation of the current implementation of HAL is due to the state explosion problem that arises when dealing with real systems. A way to solve this problem is to extend the environment with a minimization facility which provides the minimal HD-automata associated of a given π-calculus processes.

The work reported in [3] tackles the problem of minimizing labelled transition systems for name passing calculi in the abstract setting of coalgebraic theories. The main result of the paper is to provide a concrete representation of the terminal coalgebra giving the minimal HD-automaton.

The architecture of the toolkit, called MIHDA performing minimization of HD-automata is described in [8]. The structure of the toolkit is developed from the co-algebraic formulation of the partition-refinement minimization algorithm. Indeed, the concrete software architecture of the minimization toolkit is directly suggested by the abstract semantical structure of the coalgebraic specification. MIHDA is written in OCAML, runs under Linux and it is available at `http://jordie.di.unipi.it:8080/mihda`, where also an interactive WEB interface is accessible.

3 Service Orchestration

This section describes the issues related to the development of a verification toolkit which exploits the WEB as a service distributor. Here, we consider only two services, namely HAL and MIHDA; however the same techniques can be exploited to integrate in a modular fashion a variety of services. The fundamental techniques which enables the dynamic integration of services is the separation between the service facilities (what the service provides) and the mechanisms that coordinate the way services interact (service orchestration). Indeed, our main contribution consists in making service orchestration usable in the context of formal verification.

HAL and MIHDA provide several functionalities. 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. In other words, the WEB orchestration interface describes what a service can do and how to invoke it. The second step transforms

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. XML-RPC is a protocol that defines a way to perform remote procedure calls using HTTP as underlying communication protocol and XML for encoding data. XML-RPC ensures interoperability among components available over the WEB at the main cost of parsing and serializing XML documents.

3.1 Service Creation

In our running example, the WEB orchestration interface of MIHDA provides three interaction capabilities: `compile`, `reduce` and `Tofc2`. The first interaction capability takes a π-calculus agent as input and yields as output the corresponding HD-automaton. The capability `reduce` performs minimization. Finally, the capability `Tofc2` transforms the MIHDA representation of HD-automata into the FC2 format used inside HAL. The WEB orchestration interface of HAL provides the `check` capability to perform model checking, the capability `unfold` which generates a standard automaton out of an HD-automaton, and the capability `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 `Zope`. Zope is a web application server. Hence, it provides mechanisms to "publish" information on the WEB. However Zope is much more. Indeed, Zope provides a comprehensive framework for management of web contents ranging from simple HTML pages to complete components. In particular, through Zope mechanisms the calls to the capabilities of the orchestration interface are dynamically transformed into calls of the corresponding programs (e.g. via XML-RPC).

Figure 2 illustrates the WEB interface of MIHDA as provided by the Zope implementation.

3.2 Programming Service Orchestration

In our experiment, the service orchestration language is PYTHON. PYTHON is an interpreted object oriented scripting language which is widely used to connect existing components together. Expressiveness of PYTHON 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 in Figure 3 to verify a property of a specification, i.e. to test whether a π-calculus process A is a model for a formula F.

We can briefly comment on the orchestration code of Figure 3. First, XML-RPC connections with the MIHDA server and with HAL server are created and

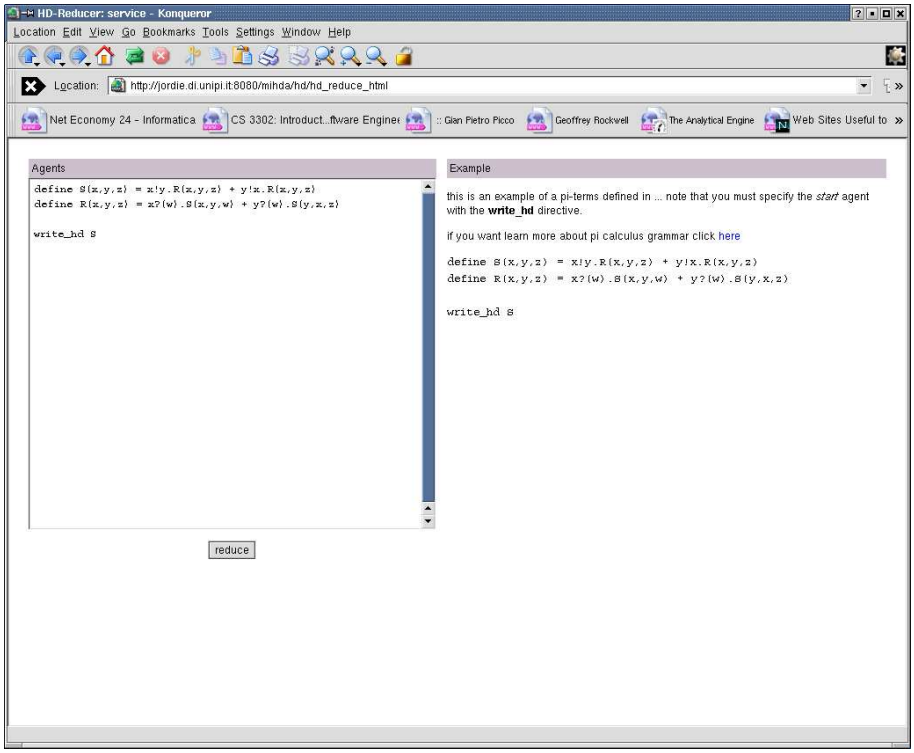

Fig. 2. MIHDA WEB Service

```
from xmlrpclib import *
import sys

try:
    mihda = Server( "http://jordie.di.unipi.it:8080/mihda/hd" )

    hal = Server( "http://bladerunner.iei.pi.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 )

    if hal.check( aut, F ):
        print 'ok'
    else:
        print 'ko'

except Exception, e:
    print "*** error ***"
```

Fig. 3. Orchestrating HAL and MIHDA services

recorded in variables `mihda` and `hal`, respectively. Now, a service of MIHDA is invoked. More precisely, the result of executing the service `compile` is stored in the variable `hd`.

Next, `hd` is minimized, by invoking the service `reduce` of MIHDA; and, by applying the MIHDA service `Tofc2`, the minimal automaton is transformed into the FC2 format. Variable `reduced_hd_fc2` contains a HD-automaton in a format suitable for being processes by the HAL service `unfold` that generate an ordinary automaton from a HD-automaton represented in FC2 format.

Finally, a message on the standard output is printed. The message depends on whether π-calculus process A satisfies the formula F or not. This is obtained by invoking the HAL model checking facility `check`. Notice that the orchestration code may handle both local and remote exceptions.

Figure 4 and Figure 5 illustrate the compiling of the π-calculus process specifying the GSM handover protocol, and the minimization step. Notice that the service orchestration program runs under WindowsXP, thus pointing out the interoperability nature of the toolkit.

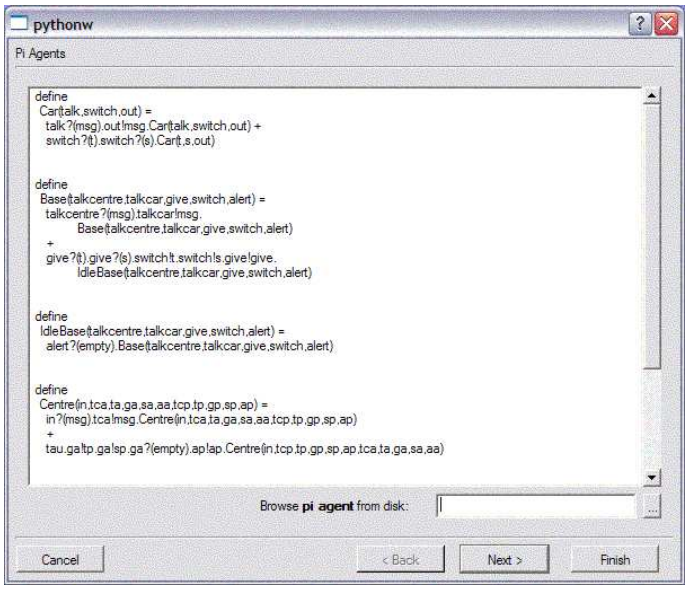

Fig. 4. Compiling

We want to point out that the only part of the orchestration code in Figure 3 that includes network dependencies is

```
mihda = Server( "http://jordie.di.unipi.it:8080/mihda/hd" )

hal = Server( "http://bladerunner.iei.pi.cnr.it:8080/hal" )
```

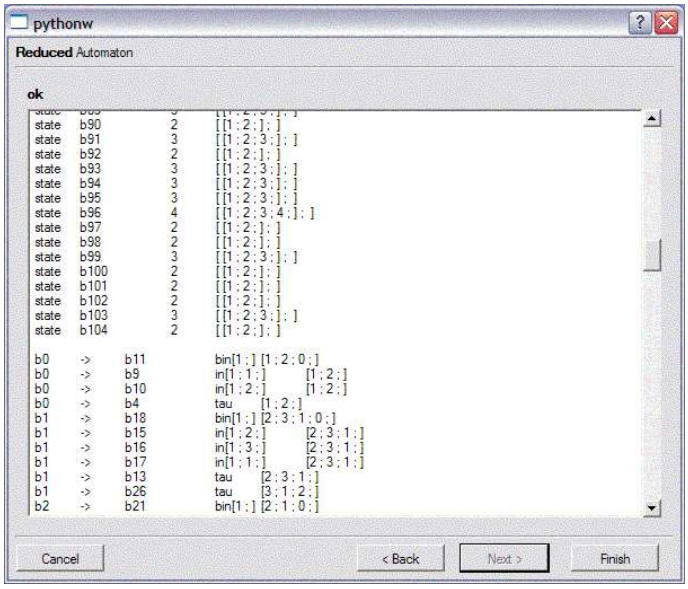

Fig. 5. Minimizing

namely, the operation that open the connections with the HAL and MIHDA servers. However, this network dependency can be removed by introducing a further module, namely the *directory of services* together with a simple *trader* facility. A directory of services is a structure that maps the description of the WEB services represented by suitable types into the corresponding network addresses. Moreover, the directory of services performs the binding of services. In other words, the directory of services can be thought of as being a sort of enriched DNS for WEB services. The directory has two facilities. The `publish` facility is invoked to make available WEB service. The `query` facility which is is used by applications to discover which are the services available. Hence, the `trader` can be used to obtain a WEB service of a certain type and to bind it inside the application.

The directory of services and the trader 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. Moreover, this mechanism makes transparent the distribution of services: when writing the orchestration code the programmer is not aware of the localities of services. Hence, a service can also be replicated or re-allocated into a new locality without requiring any change into service orchestration programs.

In our running example, to use a trader it is sufficient to substitute the assignments to `mihda` and `hal` variables with the following code:

```
import Trader

offers = Trader.query( "reducer/mihda" )
```

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

offers = Trader.query( "hal" )

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

The invocation of the `query` procedure of the `Trader` library yields the list of services that match the parameter (i.e. the string describing the kind of services we are interested in).

Directories and traders permits to hide network details in the service orchestration code. A further benefit is given by the possibility of replicating the services and maintaining a standard access modality to the WEB services under orchestration. For instance, by substituting the assignment to `offers` in the previous code with

```
offers = Trader.query( "reducer" )
```

we obtain a polymorphic orchestration code that, at run-time, is able to find, bind and finally invoke any service registered as "reducer".

4 Lessons Learned

We started our experiment with the goal of understanding whether the WEB service metaphor could be effectively exploited to develop in a modular fashion 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.

SOAP is well integrated inside the .NET framework which provides other powerful mechanisms to deal with types and metadata (i.e. the description of types). In particular, metadata information can be extracted from programs at run time, and supplied to the *emitter* to generate the corresponding data structures together with their operations. Furthermore, the *Just-in-time* compiler to makes them native code. 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, *Proceedings of the Conference on Tools and Algorithms for the Construction and Analysis of Systems (TACAS'97), Enschede, The Netherlands*, volume 1217 of *LNCS*, pages 275–289. 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. I. S. 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. To appear.