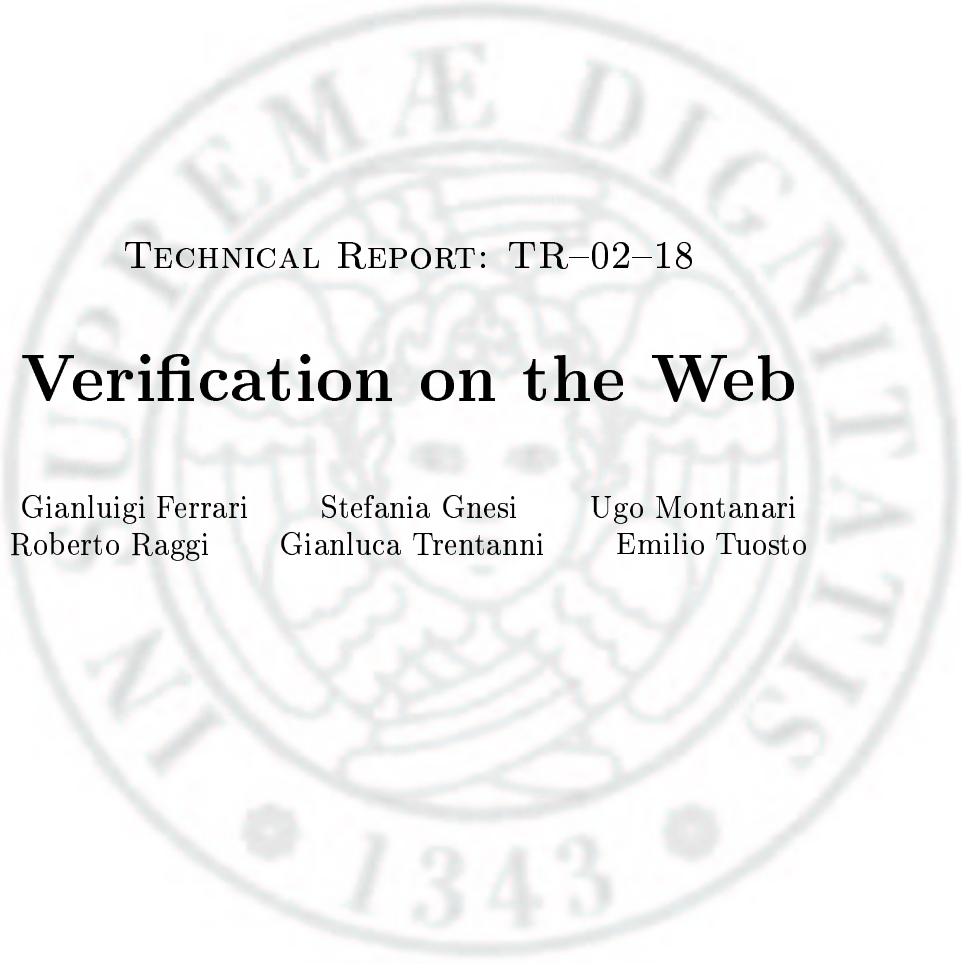
APPENDIX 1.3.1

G. Ferrari, S. Gnesi, U. Montanari, R. Raggi, G. Trentanni, and E. Tuosto. Verification on the web. Technical Report TR-02-18, Dipartimento di Informatica, Università di Pisa, 2002.

Università di Pisa
Dipartimento di Informatica



Technical Report: TR–02–18

Verification on the Web

Gianluigi Ferrari Stefania Gnesi Ugo Montanari
Roberto Raggi Gianluca Trentanni Emilio Tuosto

December 17, 2002

ADDRESS: Via F. Buonarroti 2, 56127 Pisa, Italy.
TEL: +39 050 2212700 — FAX: +39 050 2212726

Verification on the Web

Gianluigi Ferrari Stefania Gnesi Ugo Montanari
Roberto Raggi Gianluca Trentanni Emilio Tuosto

December 17, 2002

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.

Keywords: Formal Verification, Semantic-based Verification Environments, WEB Services

Contents

1 History Dependent Automata

Verification of systems that can be adequately modeled as mobile processes is difficult because many "source of infinity" can be introduced. For instance, let us consider transition systems obtained from π-agents, we have that transitions can generate new names. Indeed, let us consider the $(OPEN)$ rule of π-calculus:

$$\frac{p \xrightarrow{\overline{x}y} q}{(\nu\ y)p \xrightarrow{\overline{x}(y)} q} \quad \text{if } x \neq y. \tag{1}$$

Such rule basically establishes that a state $(\nu\ y)p$ can create a new name and can export it over a channel x. Notice that the state of the transition system corresponding to $(\nu\ y)p$ represents a point of the computation where y "does not exists", while the target state of the bound output transition is a point of the computation where y "becomes available". Rule (1) introduces an infinite branching in the automata corresponding to agents that perform bound output transitions.

The rule for input transition of the early semantics of the π-calculus also introduces infinite branching because it is necessary to consider a transition for *any* name that instantiates the input parameter.

Let us remark that it is of course reasonable (and desirable) to model π-calculus semantics with rules as (1) or by means of the early semantics because such rules account for scope extrusion of names that is one of the major peculiarities of π-calculus and permits to model and reason on many aspects of mobile systems. On the other hand, since those kind of semantics had been introduced without considering verification issues, such rules are problematic when verification purposes are under consideration.

A different phenomenon that produces infinite automata is due to name extrusion in relation with recursion. A possible "implementation" of name extrusion is to reserve an infinite sequence of names from which a new name can be taken when a transition extrudes a fresh name. This approach has been proposed and analyzed in [14, 5]. A drawback of this approach is that an infinite number of states is generated in the case of agents with infinite behaviour. Indeed, let us consider the agent $A(x) = (\nu\ y)\bar{x}y.A(y)$. Agent $A(x)$ generates a new name y, emit it along x and continues as $A(y)$. This behavior is "encoded" in the approach of [14, 5] as

$$A(x) \xrightarrow{\bar{x}(x_0)} A(x_0) \xrightarrow{\bar{x_0}(x_1)} A(x_1) \xrightarrow{\bar{x_1}(x_2)} A(x_2)...$$

Hence, to obtain finite state automata also for agents with infinite behaviour, we need a mechanism to model "resource deallocation". Let us again consider agent A; after each bound output $\bar{x_i}(x_{i+1})$ transition, the name x_i will never be used in future transitions, hence we could re-use it provided that a mechanism for re-stating its freshness is given.

In order to model this kind of evolution in a framework suitable for verifying systems it is necessary to enrich the structure of states and transitions of

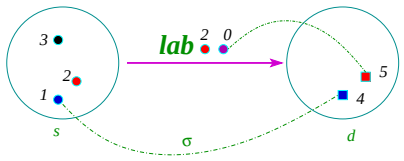


Figure 1: A HD-automaton transition

ordinary transition systems. HD-automata aim at giving a finite representation of otherwise infinite label transition systems. Similarly to ordinary automata, HD-automata are made out of states and labeled transitions. Their peculiarity resides in the fact that states and transitions are equipped with names which are no longer dealt as syntactic components of labels, but become an explicit part of the operational model. This permits to model name creation/deallocation or name extrusion that are typical linguistic mechanisms of name passing calculi.

History Dependent automata (HD-automata in brief) have been proposed in [13, 10, 11, 4] as a new operational model for history dependent calculi, namely those calculi whose semantics is defined in terms of a labeled transition system such that the labels may carry information generated in the past transitions of the system and this "historical" information can influence the future behaviour of the system. Probably the simplest history dependent calculus is CCS with value passing [9], another example is the CCS with locality [1]; finally, as we have seen π-calculus LTS semantics all have labels that can contain names generated in past transitions[1].

An important aspect of HD-automata to emphasize is that names of a state have *local meaning*. For instance, if $A(x, y, z)$ denotes an agent having three free names x, y and z, then agent $A(y, x, z)$ is different from $A(x, y, z)$, however, they can be represented by means of a single state q in a HD-automaton simply by considering a "swapping" operation on the names (corresponding to) x and y of q. More generally, states that differs only for renaming of their local names are identified in the operational model.

Local meaning of names requires a mechanism for describing how names correspond each other along transitions. Graphically, we can represent such correspondences using "wires" that connect names of label, source and target states of transitions. 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 fresh 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.

As described in Figure 1, HD-automata relies on the fact that names are local. This allows for a compact representation of agent behaviour by collapsing states that differ only for renaming of local names encompasses the main characteristics of name-passing calculi, namely, creation/deallocation of names.

[1] Also formalism that are not related to process calculi can be considered as history dependent; for instance, Petri nets [7] are a paradigmatic history dependent formalisms.

Indeed, name creation is simply handled by associating in the target state a name not in the source state.

A computation performed on a HD-automaton associates a "history" to names of the states appearing in the computation, in the sense that it is possible to reconstruct the associations which lead to the state containing the name. Clearly, if a state is reached in two different computations, different histories could be assigned to its names.

Various families of HD-automata have been introduced. Roughly speaking each class of HD-automata corresponds to a class of history dependent calculi or different behavioural semantics. The reader is referred to [13] for details. In the next sections we will present a coalgebraic definition of HD-automata for π-calculus for the early semantics. The coalgebraic presentation of HD-automata for π-agents has been introduced in [11]. In order to make this part of the dissertation self-contained, we report (with slight variations on the notation) the presentation appeared in [4] of the minimization algorithm for HD-automata.

2 Preliminaries: HAL

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. Recently a coalgebraic minimization algorithm for HD-automata has been proposed in [4] and a verification, Mihda, environment based on this coalgebraic framework has been implemented [6]. This section briefly reviews HAL the reader is referred to [6] for the description of Mihda.

A semantic-based verification environment for the π-calculus, called *HD Automata Laboratory* (HAL) has been implemented and experimented [2, 3]. HAL is written in C++ and compiled 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 has been done in two stages. First a high level logic with modalities indexed by π-calculus actions has been introduced and then a mapping which translates logical formulae into a classical modal logic for standard automata has been defined. 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 [12]. 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 overcome this problem is to extend the environment with a minimization facility which provides the minimal HD-automata of a given π-calculus processes.

The work reported in [4] tackles the problem of minimizing LTS 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.

3 Verification as a Web-Service

In the last few years distributed applications over the WEB have gained wider popularity. 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*. Rather than a monolithic application, a WEB server should be intended as a component available over the WEB that can possibly be exploited by others (human users or software applications) to develop new services. Conceptually, WEB services [8] are stand-alone components that reside over the nodes of the network and are network accessible through a network interface and standard protocols. Applications over the WEB are developed by combining and integrating WEB services together. WEB service has no pre-existing knowledge of what interactions with other WEB services may occur. Moreover, WEB services are highly portable and can easily be adapted 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 accommodate and support WEB services. We intend to explore the following issue:

> Can we simplify the design, development and maintenance of semantics-based verification environments in a modular fashion by exploiting WEB services?

A preliminary answer to this question is given 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 systems specified as π-calculus processes and it supports the dynamic integration of several verification techniques.

Finally, the toolkit has been developed by targeting also the goal of extending an available verification environment (HAL [2, 3]) 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.

4 Service Coordination

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. The main advantage of this approach consists of making service coordination usable in the context of formal verification.

HAL and Mihda provide several functionalities. The main issue to face with is the definition of WEB interfaces that make these toolkits accessible and usable on the Internet. This is done into two steps:

1. the first step defines the *WEB coordination interface* which, independently from the implementation technologies, describes the WEB interaction capabilities. In other words, the WEB coordination interface describes what a service can do and how to invoke it;

2. the second step transforms the program facilities which correspond to publish the coordination interface on the WEB.

The main programming construct we exploit to program service coordination 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.

4.1 Service Creation

In our running example, the WEB coordination 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 coordination 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 coordination interfaces has been performed by exploiting the facilities of Zope that is a web application server; 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 coordination 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.

4.2 Programming Service Coordination

In our experiment, the service coordination language is python, 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 coordination 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 coordination is illustrated in Figure 3 to verify a property of a specification, i.e. to test whether a π-calculus agent A is a model for a formula F. We can briefly comment on the coordination code of Figure 3. First, XML-RPC connections with the Mihda server and with HAL server are created and respectively recorded in variables `mihda` and `hal`. Then, 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 agent A satisfies formula F or not. This is obtained by invoking the HAL model checking facility `check`. Notice that the coordination code can transparently handle both local and remote exceptions.

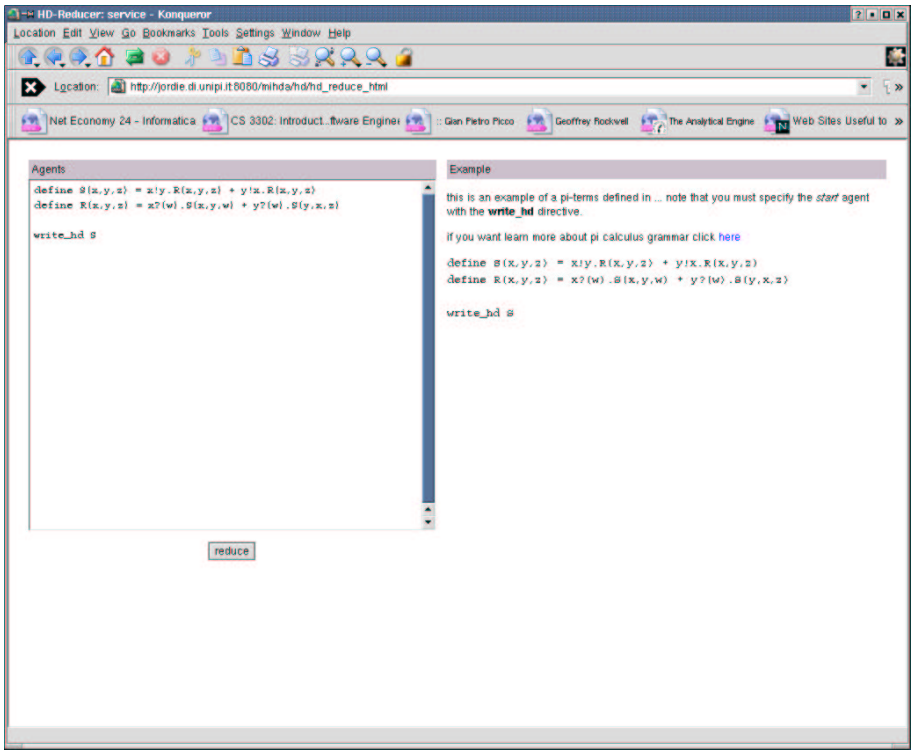


Figure 2: Mihda WEB Service

```
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 ***"
```

Figure 3: Orchestrating HAL and Mihda services

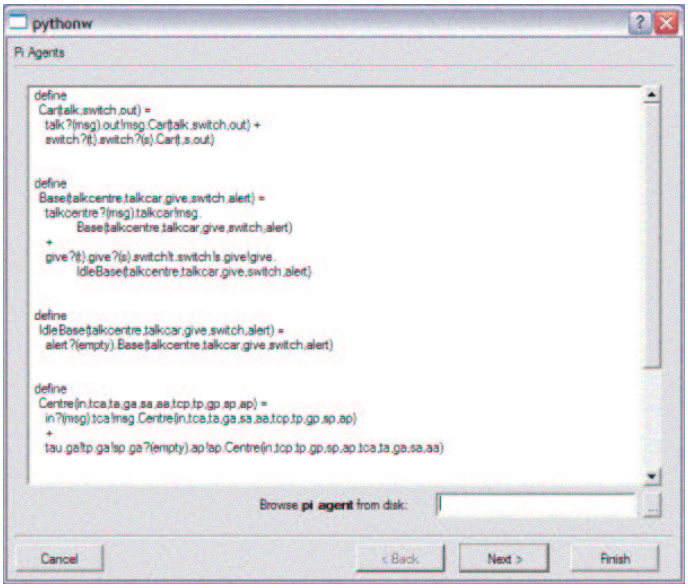


Figure 4: Compiling

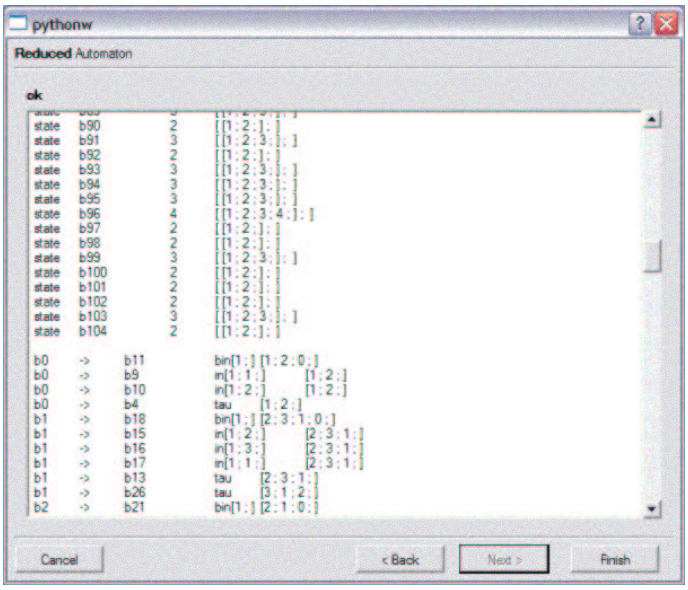


Figure 5: Minimizing

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 coordination program runs under WindowsXP, thus pointing out the interoperability nature of the toolkit. Indeed, we recall that Mihda is written in ocaml and runs over linux machines, HAL is a GNU C++ application executed in SUN-OS and both are used by executing the code in Figure 4

We remark that the only part of the coordination 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" )
```

namely, the operation that opens 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 used by applications to discover which are the available services. 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 coordination 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 coordination 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 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, by substituting the assignment to `offers` in the previous code with

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

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

5 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 coordination 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 coordination 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. 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 turns them into native code. We plan to investigate and experiment the .NET framework to design "next generation" semantic-based verification environments.

References

[1] Gérard Boudol, Ilaria Castellani, Matthew Hennessy, and Astrid Kiehn. Observing localities. *Theoretical Computer Science*, 114(1):31–61, June 1993.

[2] Gianluigi Ferrari, Giovanni Ferro, Stefania Gnesi, Ugo Montanari, Marco Pistore, and Gioia Ristori. An automata based verification environment for mobile processes. In E[d] 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, April 1997.

[3] Gianluigi Ferrari, Stefania Gnesi, Ugo Montanari, Marco Pistore, and Gioia Ristori. Verifying mobile processes in the HAL environment. In *Proc. 10th International Computer Aided Verification Conference*, pages 511–515, 1998.

[4] GianLuigi Ferrari, Ugo Montanari, and Marco Pistore. Minimizing transition systems for name passing calculi: A co-algebraic formulation. In Mogens Nielsen and Uffe Engberg, editors, *FOSSACS 2002*, volume LNCS 2303, pages 129–143. Springer Verlag, 2002.

[5] Gianluigi Ferrari, Ugo Montanari, and Paola Quaglia. A π-calculus with explicit substitutions. *Theoretical Computer Science*, 168(1):53–103, November 1996.

[6] Gianluigi Ferrari, Ugo Montanari, Roberto Raggi, and Emilio Tuosto. From coalgebraic specification to toolkit developement. Technical Report TR–02–19, Dipartimento di Informatica, Università di Pisa, Via Buonarroti, 2 – Pisa – Italy, December 2002.

[7] Ursula Goltz and Wolfgang Reisig. The non-sequential behaviour of petri nets. *Information and Computation*, 57(2/3):125–147, 1983.

[8] IBM Software Group. Web services conceptual architecture. In *IBM White Papers*, 2000.

[9] Bengt Jonsson and Joachim Parrow. Deciding bisimulation equivalences for a class of Non-Finite-State programs. *Information and Computation*, 107(2):272–302, December 1993.

[10] Ugo Montanari and Marco Pistore. History dependent automata. Technical report, Computer Science Department, Università di Pisa, 1998. TR-11-98.

[11] Ugo Montanari and Marco Pistore. pi-calculus, structured coalgebras, and minimal HD-automata. In *MFCS: Symposium on Mathematical Foundations of Computer Science*, 2000.

[12] Fredrik Orava and Joachim Parrow. An algebraic verification of a mobile network. *Formal Aspects of Computing*, 4(5):497–543, 1992.

[13] Marco Pistore. *History dependent automata*. PhD thesis, Computer Science Department, Università di Pisa, 1999.

[14] Paola Quagli. *The π-calculus with explicit substitutions*. PhD thesis, Università di Pisa, Dipartimento di Informatica, 1996.