Modular Verification of Systems via Service Coordination*

Gianluigi Ferrari, Ugo Montanari, and Emilio Tuosto

Dipartimento di Informatica, Università di Pisa
{giangi,ugo,etuosto}@di.unipi.it

Abstract. We present a service-oriented approach to the verification of properties of distributed systems specified in dialects of the π-calculus. Our verification methodology allows programming the coordination of the sub-tasks involved in verification runs together with the corresponding verification toolkits. The methodology is supported by a Web-service infrastructure integrating several verification toolkits for checking properties of specifications. Our experimental results have confirmed the potential usefulness of the approach.

1 Introduction

In the last years distributed applications over the World-Wide Web, e.g. peer-to-peer file sharing, have attained wide popularity. Several technologies have been developed for handling computing problems which involve a large number of heterogeneous components that are physically distributed and (inter)operate autonomously. These efforts have begun to coalesce around a paradigm where the Web is exploited as a *service distributor*. A service in this sense is 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 interactions with other Web services may occur. The Web service framework has emerged as the standard and natural architecture to support the so called *Service Oriented Computing* (SOC) [12, 23] paradigm where services are the basic building blocks to construct applications and service coordination becomes the main concern of the whole development process.

In [2, 15] we demonstrate that the SOC paradigm is very effective in addressing the integration issues of verification toolkits. In particular, we developed a Web-service infrastructure integrating verification toolkits for checking properties of mobile systems (e.g., [14, 16, 29]) specified in dialects of the π-calculus [22,

* Work supported by European Union project PROFUNDIS, Contract No. IST-2001-33100, by the MIUR project SP4 "Architetture Software ad Alta Qualità di Servizio per Global Computing su Cooperative Wide Area Networks".

24] and related toolkits for verifying security (e.g., [5, 27]). The development of the verification infrastructure has been performed inside the Profundis project (see URL `http://www.it.uu.se/profundis`) within the Global Computing Initiative of the European Union. For this reason we called it the *Profundis WEB*, PWeb for short. The prototype implementation of the PWeb can be exercised on-line at the URL `http://jordie.di.unipi.it:8080/pweb`.

The main idea of the approach is to make semantic-based verification toolkits available as Web services, and to establish directories for publishing such Web services. This facilitates the easy integration and maintenance of heterogeneous verification toolkits having complementary functionalities.

In the PWeb infrastructure a verification session takes the form of *service coordination* describing the rules a set of verification services have to follow to achieve a certain goal. In other words, the coordination rules are used to specify how the sub-tasks within any verification run are to be carried out, in which order and which are the different toolkits involved. Moreover, there are mechanisms for assigning verification sub-task to the specialized toolkits that are most appropriate to solve them.

Beyond the current prototype implementation, we envisage the important role that will be played by PWeb service coordination. Indeed, service coordination provides several benefits:

- *Model-based verification.* The coordination rules impose constraints on the execution flow of the verification session thus enabling a *model-based* verification methodology where several descriptions are manipulated together. Notice that there is a sound conceptual basis for model-based verification since verification toolkits provide an implementation of well understood semantic theories.
- *Modularity.* The verification of the properties of a large software system can be reduced to the verification of properties over subsystems of manageable complexity: the coordination rules reflect the semantic modularity of system specifications.
- *Flexibility.* The choice of the verification toolkits involved in the verification session may depend on the specific verification requirements.

We argue that service-based approaches have the potential to tackle the tool integration issues of the software engineering process.

The rest of this paper reports on our experience in exploiting the facilities of the PWeb infrastructure in the verification of properties of distributed systems specified in some dialect of the π-calculus. To illustrate the effectiveness and usability of our approach, we consider a simple but illustrative case study: the verification of the cryptographic protocol KSL [19]. KSL provides an abstract representation of Kerberos [20] and has been conceived for the repeated authentication between principals through a trusted server. In particular, the verification of the KSL protocol will allow us to demonstrate how service coordination supports and facilitates modular verification techniques.

Other approaches have been proposed for integrating verification toolkits; we conclude this section by discussing a few of them. In the verification community the standard approach to deal with the integration issue is to provide a coordination infrastructure based on common format. An illustrative example of this approach is provided by the FC2 format [6]. The FC2 format has been designed to represent automata by means of a set of tables that keep the information about state, and transition relations between states. The intermediate language approach has been further developed in the design of the VeriTech framework [18]. In this framework, the integration among verification toolkits is obtained by suitable functions providing faithful translations among models and properties. A key role is played by the *core design language* (CDL): each specification is compiled to and from the CDL representation.

A different approach is exploited by the *Electronic Tool Integration Platform* (ETI) initiative [9, 8]. ETI is a web-based infrastructure for the interactive experimentation of verification toolkits. The coordination middle-ware (HLL) provides the "glue" to integrate the different verification toolkits.

2 Service-Oriented Integration in the PWeb

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* [11], for example, performs analysis on the Calculus for Communicating Systems. The Mobility Workbench (MWB) [29] does similar analysis but on the π-calculus. The *History-Dependent Automata Laboratory* (HAL) [14] supports verification of logical formulae expressing properties of the behaviour of π-calculus agents.

The PWeb [2, 15] proposes itself as an experiment to address the integration issue of verification toolkits by exploiting Web services. The PWeb prototype implementation has been conceived to support reasoning about the behaviour of systems specified in some dialects 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 (MWB and HAL) 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 "exiting" modules. A Web service consists of an interface describing operations accessible by message exchange over the Internet protocol stack. The description of a Web service must cover all details needed to interact with it: the message formats, the transport protocols, and son on. Hence, Web services are a programming technology for distributed systems based on Internet standards. However, Web services are not just another object-based paradigm for distributed systems. Indeed, they promote a *service-oriented* programming style which is different from the standard user-to-program style [25, 30]. The service oriented programming

metaphor is usually characterized in terms of *publishing, finding* and *binding* cycle.

To publish-find-bind in an interoperable way Web services rely on a stack of network protocols. The building block of this protocol is the Simple Object Access Protocol (SOAP) [7]. SOAP is an XML-based messaging protocol defining standard mechanism for remote procedure calls. The Web Service Description Language (WSDL) [10] defines the interface and details service interactions. The Universal Description Discovery and Integration (UDDI) protocol supports publication and discovery facilities [31]. Finally, the Business Process Execution Language for Web Services (BPEL4WS) [26] is exploited to produce a Web service by composing other Web services.

2.1 The PWeb Verification Services

We now briefly list the main features of the new services of the PWeb. Notice that PWeb services have been developed by different groups using different programming technologies and providing complementary verification techniques. Moreover, most of the semantic-based verification environments have been developed independently of each other and there is no guarantee that they can interoperate so that the verification of certain properties is the result of a collaboration among the toolkits.

Mihda [16, 17] performs minimisation of History-Dependent (HD) 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 with as syntactic components of labels, but become explicit part of the operational model. This allows one to model explicitly name creation/deallocation, and name extrusion: these are the distinguished mechanisms of name passing calculi. Mihda has been exploited to perform finite state verification of π-calculus specifications.

$\mathcal{A}$SPASyA [4] relies on a symbolic technique to model check properties of cryptographic protocols. Security properties are expressed via a logic that predicates over data exchanged in the protocol and observed by an intruder in the execution environment, and also over the "presumed" identities of the protocol principals. $\mathcal{A}$SPASyA allows varying the intruder's knowledge, the portion of the state space to be explored, and the specification of implicit assumptions that are very frequent in security. The user can opportunely mix those three ingredients for checking the correctness of the protocol without modifying neither the protocol specification nor the specification of the desired properties.

TRUST [28, 27] 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 using the correspondence assertions [13], and whenever an error is found an intruder attacking the protocol is given.

STA [5] implements symbolic execution of cryptographic protocols. A successful attack is reported in the form of an execution trace that violates the specified property expressed in terms of correspondence assertions.

2.2 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 into the corresponding network addresses. Moreover, it supports 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` service. The description of the `reduce` service refers to the XML description of the HD-automaton. The invocation of this service on a given HD-automata performs the state minimisation of the HD-automata. The WSDL-description of the `reduce` service of the Mihda toolkit is displayed in Figure 1.

The PWeb directory service has two main facilities. The `publish` facility is invoked to make a toolkit available as Web service. The `query` facility, instead, is used to discover which are the services available. The `query` provides the service discovery mechanism: 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 service discovery mechanisms is exploited by the `trader` engine. The trader engine manipulates pools 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-wiring" the name of the web service inside the application code. In other words, the `trader` engine provides the resource discovery mechanism for PWeb directory services. A further benefit is given by the possibility of replicating the services and maintaining a standard access modality to the Web services under coordination.

The `trader` engine allows one to hide network details in the service coordination code. For instance, consider the following code:

```
0. import Trader
1. offers = Trader.query( "reduce" )
2. red = offers[ 0 ]
3. offers = Trader.query( "model-checking" )
4. mc = find_neighbor(offers)
5. offers = Trader.query( "bisimulation-checking" )
6. bis_chk = offers[ 0 ]
```

```
<?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>
```

Fig. 1. The Mihda WSDL-specification

it describes the implementation of a simple interaction with the trader of the PWeb directory. The trader is invoked for searching a reduce service (lines 1,2), a model-checking service (line 3) which is selected among the neighbor sites (line 4), and a bisimulation checker (lines 5, 6).

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. The main difference with respect to the CORBA approach is that we are interested in proving programmable trading policies.

2.3 Service Coordination

The fundamental technique 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 coordination). In our experiment, the service coordination language is PYTHON, an interpreted object-based scripting language which is widely used to connect existing components together.

Below we illustrate an example of service coordination to verify a property of a specification, i.e. to test whether a π-calculus process A is a model for a formula F.

```
:
try:
  aut = red.compile( A )
  reduced = red.reduce( aut )
  if mc.name = 'hal':
     reduced_fc2 = red.to_fc2( reduced )
     aut = mc.unfold( reduced_fc2 )
     if mc.check( aut, F ):
          print 'ok'
    else:
         print 'ko'
  else:
    mc.check(reduced, F)
except Exception, e: print "*** error ***"
```

We briefly comment on the coordination code above. Variables `red` and `mc`, have been linked by the trader engine to the required services. The compile facility `compile` of `red` is invoked to yield the automaton (stored in variable `aut`), then, the automaton is minimized. If the available model-checker is HAL, then the minimal automaton is first transformed into the FC2 format (`to_fc2`) and afterward in a format suitable (`unfold`) for being processes by the HAL model checker. Finally, a message on the standard output is printed. The message depends on whether π-calculus process `A` satisfies the formula `A` or not. This is obtained by invoking the model checking facility `check`.

3 Modular Verification: A Case Study

To illustrate the effectiveness and usability of our approach, we consider as a case study the verification of a cryptographic protocol. This will allow us to demonstrate how the PWeb supports and facilitates modular verification.

3.1 The KSL protocol

The cryptographic protocol we consider is the KSL protocol [19]. KSL provides an abstract representation of Kerberos [20] and has been conceived for the repeated authentication between principals A and B through a trusted server S. It is assumed that the trusted server shares symmetric key k_{as} and k_{bs} with principals A and B, respectively. Repeated authentication is performed by means of an expiring ticket generated by B for A. The secure communication of the ticket

relies on a session-key that A and B establish with the help of S. Until the ticket is valid (not expired), A can re-authenticate itself with B without requesting a new session key from S.

We briefly describes the informal specification of KSL as list of exchanged messages of the form $Source \rightarrow Destination : Payload$

$$\left.\begin{array}{l} 1.\ A \rightarrow B : na, A \\ 2.\ B \rightarrow S : na, A, nb, B \\ 3.\ S \rightarrow B : \{nb, A, k_{ab}\}_{k_{bs}}, \{na, B, k_{ab}\}_{k_{as}} \\ 4.\ B \rightarrow A : \{na, B, k_{ab}\}_{k_{as}}, \{Tb, A, k_{ab}\}_{k_{bb}}, nc, \{na\}_{k_{ab}} \\ 5.\ A \rightarrow B : \{nc\}_{k_{ab}} \end{array}\right\} \text{I phase}$$

$$\left.\begin{array}{l} 6.\ A \rightarrow B : ma, \{Tb, A, k_{ab}\}_{k_{bb}} \\ 7.\ B \rightarrow A : mb, \{ma\}_{k_{ab}} \\ 8.\ A \rightarrow B : \{mb\}_{k_{ab}} \end{array}\right\} \text{II phase}$$

Looking at the structure of the protocol, we can distinguish two parts:

- messages $1 \div 5$ constitutes the initial session-key exchanging phase,
- messages $6 \div 8$ are the repeated authentication part, namely, each further interaction between A and B starts from message 6.

Notice that S does not play any role in the second phase: the trusted server is involved only in generating and communicating the session key (messages $1 \div 5$).

Initiator A generates a nonce na, and sends it to B which, on turn, asks S for a new session key. In message 3, S generates a fresh session key k_{ab} and encrypts it into two cryptograms $\{nb, A, k_{ab}\}_{k_{bs}}$ and $\{na, B, k_{ab}\}_{k_{as}}$ sent to B. After decrypting $\{nb, A, k_{ab}\}_{k_{bs}}$, B assumes that k_{ab} is the fresh session key generated by S and meant to be shared with A.

Message 4 is rather involved and crucial to establish correctness of the protocol. In this step, principal B sends to A a message containing: (i) the cryptogram $\{na, B, k_{ab}\}_{k_{as}}$ generated by S, (ii) the "ticket" $\{Tb, A, k_{ab}\}_{k_{bb}}$, (iii) a new nonce nc and (iv) the nonce na encrypted with k_{ab}. The ticket is a cryptogram encrypted with a key k_{bb} that *only* B knows and will be used in the second part of KSL for achieving repeated authentication. Apart from the identity of A, the ticket contains a time-stamp and the session key so that B can check the validity of the ticket itself. The nonce nc will be used to prove to B that A really asked for the session key k_{ab}, while the cryptogram $\{na\}_{k_{ab}}$ is generated to witness A that B has acquired k_{ab}. Message 5 closes the first part of KSL: A sends back nc encrypted with k_{ab} so that B is granted that A acquired the session key.

Principal A knowing k_{ab} and the ticket issued by B can re-authenticate itself performing messages 6, 7 and 8. In message 6, B receives a nonce, ma, and the ticket that B has previously generated for A. If the ticket is valid, B sends ma encrypted with k_{ab} to A together with a new nonce mb, used to ensure the identity of A (message 8).

3.2 Verifying KSL

Model checking techniques have been exploited in the verification of cryptographic protocols because they can provide a counterexample when some property fails to hold: *attack generation*. However, model checkers usually require to

limit various "quantity" of the protocol (e.g., the number of participants, the length of the messages) in order to maintain the search space finite.

Traditional model checking techniques can hardly afford the complexity of protocols with many steps (as the KSL) because of the state explosion problem. In the case of KSL, also the use of symbolic techniques results harmless and can only handle sessions with a very limited number of participants.

To cope with this problem we exploit the modularity of the specification by splitting the verification session of KSL into two parts (reflecting the two phases of the protocol). Standard semantic arguments ensures that the second phase of KSL can be checked under the hypothesis that the first phase is safe. More precisely, the repeated authentication property relies on the secrecy of the session key and the tickets exchanged in the first phase. Hence, the second phase can be verified under the assumption that the session-key (and the validity ticket) are not corrupted. We, therefore, check the repeated authentication phase only when the secrecy of k_{ab} has been assessed.

The KSL verification session can be roughly described by means of the following *pseudo-code*.

```
1. safe:= false;
2. while not safe [
3.   get(property);
4.   safe:= test(property, KSL[1-5]);
   ]
5. return test(repeat_auth, KSL[6-8]).
```

This code could be easily implemented in PYTHON, the PWeb coordination language, as described in Section 2.3,

Notice that the coordination schemata allows us to combine two different verification techniques, namely bisimulation checking and model checking. The first phase of KSL is verified by means of a bisimulation checking technique and then model checking is exploited in the second phase (detailed in Section 3.4 and 3.5, respectively).

3.3 Encoding KSL in π-calculus

The π-calculus is a name-passing process calculus where names model communication ports along which process send/receive other (port) names. Conventionally, output of a name x on port a is written as $\overline{a}\langle x\rangle$ while $a(y)$ is the input action. Consider the process $(\nu\ x)(\overline{a}\langle x\rangle.P) \mid a(y).Q$ which represents the parallel composition of $(\nu\ x)(\overline{a}\langle x\rangle.P)$ and $a(y).Q$. The former outputs a name x on a and continues as P; notice that x is in the scope of the binder ν which models the fact that x is a freshly-generated local name. The process $a(y).Q$ receives on a and continues as the process Q where the received name replaces y. The reduction representing this behaviour is written as

$$(\nu\ x)(\overline{a}\langle x\rangle.P) \mid a(y).Q \rightarrow (\nu\ x)(P \mid Q[^{x}/_{y}]).$$

Note that the scope of the binder ν has been enlarged in the reduction step and after the communication it contains the continuation of the input process also. This is called *scope extrusion* and is one of the main features of π-calculus. In this paper we only give an informal description of the π- calculus; the interested reader is referred to [22, 21, 24] for more detailed presentations.

The bisimilation checking tools of the PWeb can only deal with versions of the π-calculus where only names and not terms can be exchanged. Hence, in order to use such tools for verifying KSL, we must somehow encode tuples and cryptograms. For the sake of simplicity, we do not consider here a mapping from the spi-calculus to π-calculus (like [3]) and prefer to use a simpler and more specific mapping. Communication of tuples is simply regarded as the separated communication of each field of the tuple while the case of cryptograms is more involved. We limit ourselves to symmetric cryptography which is the only type of encryption used in KSL. Sending a cryptogram $\{M\}_k$, where k is a symmetric key, can be interpreted as sending M over k. This guarantees that a process can acquire M only if it knows k. The delicate modelling of sharing/communicating of secrets is resolved by exploiting restriction of names together with *scope extrusion*.

We now describe the π-calculus specification of the KSL server. Hereafter, we use $\overline{a}\langle x_1, \ldots, x_n\rangle$ (resp. $a(x_1, \ldots, x_n)$) as a shorthand for $\overline{a}\langle x_1\rangle.\ldots.\overline{a}\langle x_n\rangle$ (resp. $a(x_1).\ldots.a(x_n)$).

The KSL server can be written as the following π-calculus process (lines starting with `#` are comments).

$$
\begin{array}{lll}
S(p,a,k_{as},b,k_{bs}) \triangleq p(na,\_a,nb,\_b).[\_a=a][\_b=b] & \texttt{\# receive\&check request} \\
\quad (\nu\ k_{ab}) & \texttt{\# generate a session key} \\
\quad \overline{k_{bs}}\langle nb,a,k_{ab}\rangle. & \texttt{\# send } \{nb,a,k_{ab}\}_{k_{bs}} \texttt{ to } B \\
\quad \overline{k_{as}}\langle na,b,k_{ab}\rangle. & \texttt{\# send } \{na,b,k_{ab}\}_{k_{as}} \texttt{ to } A \\
\quad S(p,a,k_{as},b,k_{bs})) & \texttt{\# restart.}
\end{array}
$$

The server specification is parameterised with respect to five names:

- p represents the public channel over which all public messages are sent/received;
- a and b do not represent communication channels but they are used to denote the identities of the principals A and B of KSL;
- k_{as} and k_{bs} are the keys that S shares with A and B, respectively.

The server $S(p,a,k_{as},b,k_{bs})$ waits on the public channel p for the request of B (message 2 of KSL). It reads in variables na and nb the values of the nonces sent by B while the identities of the principals are stored in $\_a$ and $\_b$. Then, $S(p,a,k_{as},b,k_{bs})$ checks the identities (the construct $[\_a = a]$ can be read as **if** $\_a = a$ **then** $\cdots$) and if both are correct the session key k_{ab} is generated and extruded to the other principals on the corresponding private keys.

The other principals of KSL are reported in Table 1. Notice that they are all recursive processes that continously repeat the first phase of KSL. Cryptographic protocols often relies upon implicit assumption that is better to take into account in their verification. For instance, it is usually assumed that a principal can

$A_I(pa, qa, a, k_{as}, b) \triangleq$	$B_I(pa, p, qa, b, k_{bs}) \triangleq$
$(\nu\ na)($	$qa(na, a).(\nu\ nb)($
$\overline{pa}\langle na, a\rangle.$	$\overline{p}\langle na, a, nb, b\rangle.$
$k_{as}(n, r, k_{ab}).$	$k_{bs}(n, i, k_{ab}).[n = nb][i = a]$
$[n = na][r = b]$	$(\nu\ nc, k_{bb}, Tb, e)(\overline{pa}\langle e, nc\rangle.$
$qa(t, nc).$	$\overline{k_{ab}}\langle na\rangle.$
$k_{ab}(n).[n = na]$	$k_{ab}(n).[n = nc]$
$\overline{k_{ab}}\langle nc\rangle.$	$B_I(pa, p, qa, b, k_{bs})))$
$A_I(pa, qa, a, k_{as}, b))$	

Table 1. Principals of KSL in π-calculus

recognize whether messages are intended for himself or for other principals and, in the latter case, reject them. In π-calculus this can be modelled by using different public channels; for instance B_I uses p to communicate with S, pa for receiving from A_I and qa for sending to A_I.

Finally, the session of (first phase of) the protocol is described by the following process:

$$KSL(pa, qa, a, b) \triangleq (\nu\ p,\ \ k_{as}, k_{bs}, p)(S(p, a, k_{as}, b, k_{bs}) \mid A_I(pa, qa, a, k_{as}, b) \mid B_I(pa, p, qa, b, k_{bs})).$$

Note that the non-restricted names are only the public channels pa, qa and the principal identities a and b. The public channel p (that B_I uses for communicating with S) has been restricted for modelling the trustedness of the server. This is a standard assumption in cryptography and holds for KSL as well.

3.4 Secrecy via bisimulation checking

The secrecy of the session key k_{ab} of KSL is checked by contrasting the π-calculus specification of KSL with a slightly different version specifying a protocol where the principals "knows" the session key in advance. More precisely, we turn $KSL(pa, qa, a, b)$ into a "magic" process where the session-key cannot be compromised. Then we check that the magic specification is bisimilar to the original one. This approach has been introduced in [1] for the spi-calculus and ensures the secrecy of the session key because in the magic version the correct key-exchange is forced.

Let us consider the following process

$$\widehat{KSL}(pa, qa, a, b) \triangleq (\nu\ p, k_{ab}, k_{as}, k_{bs})(S_{k_{ab}}(p, a, k_{as}, b, k_{bs}) \mid A_{I,k_{ab}}(pa, qa, a, k_{as}, b, k_{ab}) \mid B_{I,k_{ab}}(pa, p, qa, b, k_{bs}, k_{ab})),$$

where $S_{k_{ab}}$, $A_{I,k_{ab}}$ and $B_{I,k_{ab}}$ are reported in Table 2. The components of $\widehat{KSL}$ differs from those in KSL because they share k_{ab} in advance and $A_{I,k_{ab}}$ and

$B_{I,k_{ab}}$ check whether a correct key exchange has taken place or not: they stop executing the protocol in the later case and continue in the former one. Therefore, the secrecy of k_{ab} can be assessed by verifying that KSL and $\widehat{KSL}$ are bisimilar.

$A_{I,k_{ab}}(pa, qa, a, k_{as}, b) \triangleq$ $(\nu\ na)($ $\overline{pa}\langle na, a\rangle.$ $k_{as}(n, r, x).[x = k_{ab}]$ $[n = na][r = b]$ $qa(t, nc).$ $k_{ab}(n).[n = na]$ $\overline{k_{ab}}\langle nc\rangle.$ $A_{I,k_{ab}}(pa, qa, a, k_{as}, b))$	$B_{I,k_{ab}}(p, b, k_{bs}) \triangleq$ $qa(na, a).(\nu\ nb)($ $\overline{p}\langle na, a, nb, b\rangle.$ $k_{bs}(n, i, x).[n = nb][i = a][x = k_{ab}]$ $(\nu\ nc, k_{bb}, Tb, e)(\overline{pa}\langle e, nc\rangle.$ $\overline{k_{ab}}\langle na\rangle.$ $k_{ab}(n).[n = nc]$ $B_{I,k_{ab}}(p, b, k_{bs})))$
$S_{k_{ab}}(p, a, k_{as}, b, k_{bs}) \triangleq$ $p(na, \_a, nb, \_b).[\_a = a][\_b = b]$ $\overline{k_{bs}}\langle nb, a, k_{ab}\rangle.$ $\overline{k_{as}}\langle nb, a, k_{ab}\rangle.$ $S_{k_{ab}}(p, a, k_{as}, b, k_{bs})$	

Table 2. The magic version of KSL

To check bisimilarity we exploit two facilities of Mihda: `pi-to-hd` and `reduce`. The former transforms π-calculus processes into a HD-automata (according to the early-semantics of π-calculus) while `reduce` minimises the HD-automata by applying a partition refinement algorithm.

Table 3 reports the execution time (in seconds) and the size of the HD-automata generated by the two facilities for KSL and $\widehat{KSL}$. The output of Mihda

	Mihda.`pi-to-hd`			Mihda.`reduce`		
	States	Transitions	Time	States	Transitions	Time
KSL	2937	3919	14.5s	154	322	7.39s
$\widehat{KSL}$	2353	3243	13.8s	154	322	301.7s

Table 3. Results of Mihda: the tests have been executed on an Linux system (Kernel 2.4) running on a AMD Athlon(TM) XP 1800MHz with 1Gb of RAM memory

verification formally confirms that KSL ensures the secrecy of the session key k_{ab}, namely $\widehat{KSL}$ and KSL are bisimilar.

Notice that the size of KSL is (more than) one order of magnitude greater than the corresponding minimal realisation (and similarly for $\widehat{KSL}$). This is valuable if one considers that the minimal realisation of a transition system preserves the set of properties of the original system (they are logically indistinguishable),

therefore, model checkers (e.g., HAL or $\mathcal{A}$SPASyA) might visit the minimal realisation instead of the full state space for checking other properties.

3.5 Model checking KSL

The second phase of KSL is verified by the model-checking facilities of $\mathcal{A}$SPASyA. In this case the methodology consists of four steps: 1) Specification of the behaviour of the principals and the desired property, 2) specification of the conditions on intended sharing of secrets, 3) specification of the power of the intruder, in terms of its initial knowledge, 4) automatic verification of whether the protocol executions, as they have been formalised, do or do not satisfy the property. The results of step 4) can be exploited to iterate steps 2), 3), and 4), according to the insights gained in the previous iterations about the actual, and often unexpected, behaviour of the protocol.

Step 1) is performed by exploiting a dialect of π-calculus specifically designed to tackle security issues. Moreover, properties are expressed in an *ad-hoc* logic. For the sake of simplicity, we continue to describe protocol principals in π-calculus: Since we are assuming that the interactions of A and B with S are trusted and not compromised, we can consider only the principals for A and B:

$$\begin{aligned} A_{II}(p,b,t,k_{ab}) \triangleq\ & (\nu\ nma, e_1, e_2)(\\ & \overline{p}\langle nma, e_1\rangle. \\ & p(m,e).\overline{e}\langle e\rangle.k_{ab}?n.[n = nma] \\ & \overline{p}\langle e_2\rangle \mid Enc(e_1, k_{ab}, t, b, A) \mid Enc(e_2, k_{ab}, n)) \end{aligned}$$

$$\begin{aligned} B_{II}(p,a,k_{ab},b,Tb,k_{bb}) \triangleq\ & (\nu\ nmb, e_1)(\\ & p(n,e).\overline{e}\langle e\rangle.k_{bb}(t,i,k,).[t = Tb][i = a][k = k_{ab}] \\ & \overline{p}\langle nmb, e_1\rangle. \\ & p(e).\overline{e}\langle e\rangle.k(nmb).[k = k_{ab}]\tau \mid Enc(e_1, k_{ab}, nma)), \end{aligned}$$

where $Enc(e,k,x_1,\ldots,x_n) \triangleq e?u.\overline{k}\langle x_1,\ldots,x_n\rangle$ is the encoding of the cryptogram $\{x_1,\ldots,x_n\}_k$.

A distinguished features of $\mathcal{A}$SPASyA is that the portion of the state space to be explored can be ruled by means of the so-called *connection formulae* which impose constraints on the possible assignments of the variable by relating them to other variables or values. For instance, the assumptions on the secrets shared by A and B can be formalised by means of the following connection formula:

$$\bar{\phi}_{KSL} = \exists B.l : \exists A.j : b_j = a_l \rightarrow kab_j = kab_l,$$

stating that, if there are two instances of A and B that have executed the first phase as initiator and responder, then they share a session key ($kab_j = kab_l$). $\mathcal{A}$SPASyA will reject any session of the protocol that does not respect this constraint.

The formula for the repeated authentication can be similarly stated as

$$\bar{\psi}_{KSL} \triangleq \forall B.l : \exists A.j : b_j = B_l\ \wedge\ a_l = A_j \rightarrow ma_l = nma_j\ \wedge\ mb_j = nmb_l,$$

stating that if two instances of A and B aimed at communicating each other then the nonce challenge has been respected.

Similarly to other toolkits performing symbolic analysis (e.g. STA, TRUST) $\mathcal{A}$SPASyA allows varying the intruder's knowledge and the specification of implicit assumptions (which are very frequent in security). However, differently from STA and TRUST, in $\mathcal{A}$SPASyA these ingredients can be opportunely mixed without modifying neither the protocol specification nor the specification of the desired properties.

Table 4 reports the results of the verification of the second phase in the cases of two and three instances of principals. Since the former case does not yield any attack, we focus on the latter one.

	2 Instances			3 Instances		
Knowl.	Conf.	Time (s)	Attacks	Conf.	Time (s)	Attacks
$true, \kappa_0$	104	0.69	0	3878	1.53	8
$true, \bar{\kappa}_0$	104	0.85	0	3878	1.89	8
$\bar{\phi}_{KSL}, \kappa_0$	71	0.64	0	3220	1.50	6
$\bar{\phi}_{KSL}, \bar{\kappa}_0$	71	0.80	0	3220	1.85	6

Table 4. Attack report for KSL repeated authentication part

The initial knowledge κ_0 contains the information communicated on the public channel p during the first phase (for instance the names of each instance, i.e., $\{B_1, B_2, A_3\} \subseteq \kappa_0$). In this case, $\mathcal{A}$SPASyA finds (and reports) the following attack:

1. $A_3 \rightarrow I : nma_3, \{B_2, A_3, ks\}_{kb2}$
2. $I \rightarrow B_2 : nma_3, \{B_2, A_3, ks\}_{kb2}$
3. $B_2 \rightarrow I : nmb_2, \{nma_3\}_{ks}$
4. $I \rightarrow B_1 : nmb_2, \{B_1, A_3, ks\}_{kb1}$
5. $B_1 \rightarrow I : nmb_1, \{nmb_2\}_{ks}$
6. $I \rightarrow B_2 : \{nmb_2\}_{ks}$
7. $I \rightarrow A_3 : nmb_1, \{nma_3\}_{ks}$
8. $A_3 \rightarrow I : \{nmb_1\}_{ks}$
9. $I \rightarrow B_1 : \{nmb_1\}_{ks}$

where I plays the role of the intruder that can corresponds to the public channel p in our π-calculus specification. In messages $1 \div 3$, A_3 and B_2 begin the authentication phase; the communications are possible because of the ticket has been sent over p. In messages $4 \div 5$, the intruder I, playing the role of A_3, uses B_1 for encrypting nmb_2 with ks. At this point, I can match the input data requested by B_2 and can subsequently use A_3 to obtain $\{nmb_1\}_{ks}$. Hence, I has been able to let B_1 believe he was interacting with A_3 while he was interacting with I, violating $\bar{\psi}_{KSL}$.

4 Concluding Remarks

We started our experiment with the goal of understanding whether the SOC paradigm could be effectively exploited to integrate verification toolkits. In this respect, the prototype implementation of the PWeb is a significant example. The main advantage of our coordination model resides in providing an abstract layer to support semantic-based verification methodologies. The experiments we

have performed, including the one reported in this paper, have confirmed the potential usefulness of the approach.

The basic PWeb framework can be extended in several directions. In conclusion, we list some of the area of future research (*i*) abstraction techniques for automatic decomposition, (*ii*) advanced discovery mechanisms, (*iii*) advanced coordination mechanisms and trading facilities.

References

1. M. Abadi and A. Gordon. A calculus for cryptographic protocols: The spi calculus. *Information and Computation*, 148(1):1–70, January 1999.
2. M. Baldamus, J. Bengston, G. Ferrari, and R. Raggi. Web services as a new approach to distributing and coordinating semantics-based verification toolkits. In *Web Services and Formal Methods*, ENTCS. Elsevier, 2004.
3. M. Baldamus, J. Parrow, and B. Victor. Spi calculus translated to p–calculus preserving may-tests. In *Annual Symposium on Logic in Computer Science LICS*, volume 19th, pages 22–31. IEEE Computer Society, July, 14 – 17 2004.
4. G. Baldi, A. Bracciali, G. Ferrari, and E. Tuosto. A coordination-based methodology for security protocol verification. In *WISP 2004 (Busi, Gorrieri, Martinelli eds)*, Electronic Notes in Theoretical Computer Science. Elsevier, 2004.
5. 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.
6. A. Bouali, A. Ressouche, V. Roy, and R. D. Simone. The fc2tools set. In *CAV*, volume 1102 of *Lecture Notes in Computer Science*. Springer-Verlag, 1996.
7. D. Box, D. Ehnebuske, G. Kakivaya, A. Layman, M. Mendelsohn, H. Nielsen, S. Thatte, and D. Winer. *Simple Object Access Protocol (SOAP) 1.1*. WRC Note, http://www.w3.org/TR/2000/NOTE-SOAP-2000058/, 2000.
8. V. Braun, J. Kreileder, T. Margaria, and B. Steffen. The ETI online service in action. In *TACAS*, volume 1579 of *Lecture Notes in Computer Science*, pages 439–443. Springer-Verlag, 1999.
9. V. Braun, T. Margaria, and B. Steffen. The electronic tool integration platform: Concepts and design. *Software Tools for Technology Transfer*, 1(1-2):31–48, 1997.
10. R. Chinnici, M. Gudgina, J. Moreau, and S. Weerawarana. Web service description language (wsdl), version 1.2. Technical report, 2002.
11. R. Cleaveland, J. Parrow, and B. 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.
12. F. Curbera, K. R, N. Mukhi, S. Tai, and S. Weerawarane. The next step in web services. *Com. ACM*, 46(10), 2003.
13. D. Dolev and A. Yao. On the security of public key protocols. *IEEE Transactions on Information Theory*, 29(2):198–208, 1983.
14. G. Ferrari, S. Gnesi, U. Montanari, and M. Pistore. A model checking verification environment for mobile processes. *ACM Transactions on Software Engineering and Methodology*, 12(4), 2004.
15. G. Ferrari, S. Gnesi, U. Montanari, R. Raggi, G. Trentanni, and E. Tuosto. Verification on the web. In J. Augusto and U. Ultes-Nitsche, editors, *2nd International Workshop on Verification and Validation of Enterprise Information Sys-*

tems, VVEIS 2004, pages 72 – 74, Porto, Portugal, April 2004. INSTICC Press. In conjunction with ICEIS 2004.
16. G. Ferrari, U. Montanari, R. Raggi, and E. 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.
17. G. Ferrari, U. Montanari, and E. Tuosto. Coalgebraic minimisation of HD-automata for the π-calculus in a polymorphic λ-calculus. *Theoretical Computer Science*, 2003. To appear.
18. S. Katz and O. Grumberg. A framework for translating models and specifications. In P. Butler and Sere, editors, *IFM 2002*, volume 2335 of *Lecture Notes in Computer Science*, pages 145–164. Springer-Verlag, 2002.
19. A. Kehne, J. Schönwälder, and H. Langendörfer. Multiple authentications with a nonce-based protocol using generalized timestamps. In *Proc. ICCC '92*, Genua, 1992.
20. J. Kohl and B. Neuman. The kerberos network authentication service (version 5). Internet Request for Comment RFC-1510, 1993.
21. R. Milner. *Commuticating and Mobile Systems: the π-calculus*. Cambridge University Press, 1999.
22. R. Milner, J. Parrow, and D. Walker. A calculus of mobile processes, I and II. *Information and Computation*, 100(1):1–40,41–77, September 1992.
23. M. P. Papazoglou. Service-oriented computing: Concepts, characteristics and directions. In *WISE 2003*, Lecture Notes in Computer Science, pages 3–12, 2003.
24. D. Sangiorgi and D. Walker. *The π-calculus: a Theory of Mobile Processes*. Cambridge University Press, 2002.
25. M. Stal. Web services: Beyond component-based computing. *Communications of ACM*, 55(10):71–76, 2002.
26. A. T. and et al. Business process execution language for web services (bpel4ws), version 1.1. Technical report, 2003.
27. V. Vanackere. *The TRUST protocol analyser*. Lab. Informatique de Marseille, http://www.cmi.univ-mrs.fr/ vvanacke/trust.html, 2002.
28. V. Vanackere. The trust protocol analyser, automatic and efficient verification of cryptographic protocols. In *Verification Workshop - Verify02*, 2002.
29. B. Victor and F. Moller. The Mobility Workbench — a tool for the π-calculus. In D. Dill, editor, *CAV'94: Computer Aided Verification*, volume 818 of *Lecture Notes in Computer Science*, pages 428–440. Springer-Verlag, 1994.
30. W. Vogels. Web services are not distributed objects. *IEEE Internet Computing*, 7(6):59–66, 2003.
31. W3C. UDDI Technical White Paper. Technical report, 2000.