View metadata, citation and similar papers at core.ac.uk

brought to you by CORE

provided by Scientific Publications of the University of Toulouse II Le Mirail

Orchestration under Security Constraints

Yannick Chevalier, Mohammed Anis Mekki, Michael Rusinowitch

To cite this version:

Yannick Chevalier, Mohammed Anis Mekki, Michael Rusinowitch. Orchestration under Security Constraints. Bernhard K Aichernig and Frank S. de Boer and Marcello M. Bonsangue. Formal Methods for Components and Objects (FMCO 2010), Nov 2010, Graz, Austria. Springer, 6957, 2011, Lecture Notes in Computer Science; Formal Methods for Components and Objects. <hal-00642855>

HAL Id: hal-00642855

https://hal.inria.fr/hal-00642855

Submitted on 1 Dec 2011

HAL is a multi-disciplinary open access archive for the deposit and dissemination of scientific research documents, whether they are published or not. The documents may come from teaching and research institutions in France or abroad, or from public or private research centers.

L'archive ouverte pluridisciplinaire **HAL**, est destinée au dépôt et à la diffusion de documents scientifiques de niveau recherche, publiés ou non, émanant des établissements d'enseignement et de recherche français ou étrangers, des laboratoires publics ou privés.

Orchestration under Security Constraints

Yannick Chevalier, Mohamed Anis Mekki, Michaël Rusinowitch

LORIA & INRIA Nancy Grand Est, France
E-mail: FirstName.LastName@loria.fr

Abstract. Automatic composition of web services is a challenging task. Many works have considered simplified automata models that abstract away from the structure of messages exchanged by the services. For the domain of secured services (using e.g. digital signing or timestamping) we propose a novel approach to automated composition of services based on their security policies. Given a community of services and a goal service, we reduce the problem of composing the goal from services in the community to a security problem where an intruder should intercept and redirect messages from the service community and a client service till reaching a satisfying state. We have implemented the algorithm in AVANTSSAR Platform [5] and applied the tool to several case studies.

1 Introduction

To meet frequently changing requirements and business needs, for instance in a federation of enterprises, components are replaced by *services* that are distributed over the network (e.g. the Internet) and *composed* in a demand-driven and flexible way. Service-oriented architectures (SOAs) have gained much attention as a unifying technical architecture that can address the challenges of this ever-evolving environment.

Secured Services. Since it is not acceptable in many cases to grant access to a service to any person present on the Internet, one has to regulate the use of services by *policies*. These policies express the context, including the requester's identity, her credentials, the link between the service and the requester, and higher-level business rules to which a service is subject. They also dictate how the information transmitted between services has to be protected on the wire. In the following we call *secured service* a service that is protected by a security policy.

Composition of Secured Services. Each service may rely on the existence and availability of other (possibly dynamically retrieved) partner services to perform its computation. For this one needs dynamic adaptation and explicit combination of applicable policies, which determine the actions to be executed and the messages to be exchanged in order to satisfy the client requests in accordance with the partners policies. For example, a service granting the access to a resource of a business partner may use a local authentication service, trusted by both partners, to assess the identity of a client and rely on authorization services on both ends that combine their policies to decide whether to grant the access or not.

Contribution of this work. For the domain of secured services we propose a novel approach to automated composition of services based on their security policies. Given a community of services and a goal service, we reduce the problem of composing the goal

from services in the community to a security problem where an intruder should intercept and redirect messages from both the community services and the client in such a way that the client service reaches its final state, defined as an insecure one. The approach amounts to collecting the constraints on messages, parameters and control flow from the components services and the goal service requirements. A constraint solver checks the feasibility of the composition, possibly adapting the message structure while preserving the semantics, and displays the service composition as a message sequence chart. The resulting composed service can be verified automatically for ensuring that it cannot be subject to active attacks from intruders. The services that are input to our system are provided in a declarative way using ASLan [2], a high level specification language. The approach is fully automatic and we show on a case-study how it succeeds in deriving a composed service that is currently proposed as a product by a company.

Paper organization. In Section 3 we explain our approach on a simple example. In Section 4 we introduce the formal model we consider for secured Web services. We explain the mediator synthesis problem in Subsection 4.1, how to represent messages in Subsection 4.2, and services in Subsection 4.3. The composition problem is formally stated in Subsection 4.4 and solved in Subsections 4.5 and 4.6. In Subsection 4.7 we formalize the obtained composed service in the ASLan language in order to verify it automatically against classical security properties. In Section 5 we describe the experiments we have performed on three case-studies and put the focus on one provided by *OpenTrust* company. We show how we can derive automatically a Digital Contract Signing service from their components services. Finally we also prove automatically that the resulting service cannot be subject to active attacks. We conclude in Section 6 where we give several perspectives.

2 Related work

Most works on Web service composition are based on automata representations of web services [9, 11, 24], trying to synthesize, from available components, an automata whose behavior simulates the specification of the goal service. These approaches are focusing on control flow. For instance. the Roman model [8] focuses on deterministic atomic actions, and has been extended by data and communication capabilities in the Colombo model [9]. Synthesis of composite services in [9, 8] is based on Propositional Dynamic Logic.

Another approach to composition relies on advanced AI planning techniques. For instance [17] applies these techniques at the knowledge level to address the scalability problem, retaining only the features of services that are relevant to compose them. According to [24] most solutions in the literature involve too much human encoding or do not address the problem of data heterogeneity, hence are still far from automatic generation of executable processes. Given the variations in information representation such as message-level heterogeneity *data mediation* is crucial for handling service composition. Hence our objective is to handle (in some cases) structural and semantic heterogeneity as defined by [18]. Furthermore we take into account the effects of the security policy of the services on the format of the messages. An advantage of our approach is that it can handle automatically message structure adaptation since the orchestrator has capa-

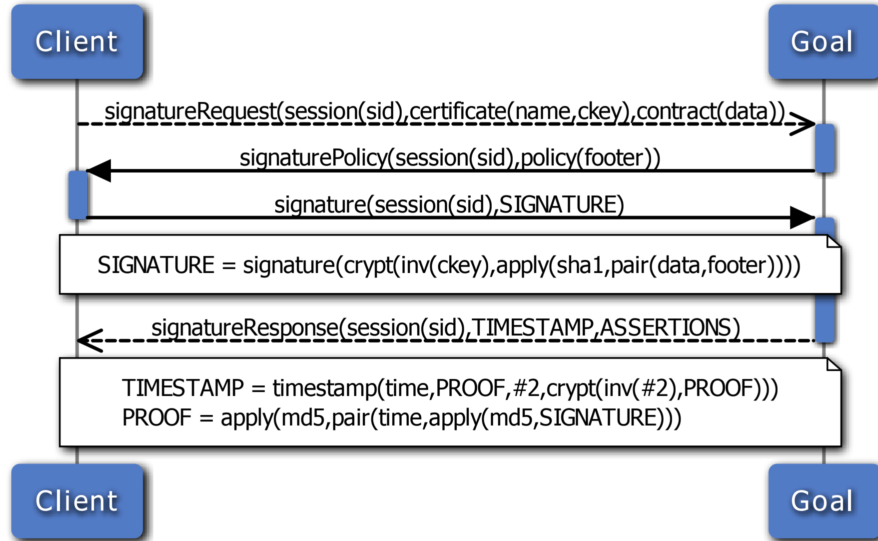


Fig. 1. Time stamping and archiving a digital signature

bilities (presented by a formal deduction system) to apply operations on messages and build new messages. This provides for free automatic adaptation of messages for proper service communications. We also address the problem of checking that the composed service satisfies some security properties. For the validation of the synthesized service we can employ directly our cryptographic protocol validation tools [6, 21].

While our approach focuses on the problems related to message adaptation, it is significantly less expressive than more standard automata-based techniques when considering complex goal services. In particular we consider a bounded orchestration problem in which the number of communications is bounded, thereby excluding iteration completely. As a consequence we believe that the method presented in this paper is more complementary with than a concurrent of these standard methods, and that future work should focus on integrating these approaches in a common framework.

3 Introductory example

Figure 1 illustrates a composition problem corresponding to the creation of a new service (described here by *Goal*) for appending a timestamp to a digital signature performed by a given partner (described here by *Client*) over some data (described here by *data*) and then submitting it together with the signed data and some other proofs for long time conservation by an archiving third party. More precisely *Goal* should expect a first message from *Client* containing a session identifier *sid*, the *Client*'s certificate containing his identity and his public key *ckey* and finally the data he wishes to digitally sign. *Goal* should answer with a message containing the same session identifier and a *footer* value to be appended to the data before the *client*'s signature. This value aims to

capture the fact that the *Client* acknowledges a certain chart (known by *Goal*) before using the service *Goal*. Indeed this is what *Client* is expected to send back to *Goal*. *Goal* should then append to the received digital signature (described by *SIGNATURE*) a timestamp (described by *TIMESTAMP*). The timestamp consists of a *time* value which is bound to the *Client*'s signature (through the use of *md5* hash) and signed by a trusted timestamper's private key *#2*.

Goal should also include a certain number of assertions or proofs about its response message. *ASSERTIONS* is described below and consists of 4 assertions or judgments.

```
ASSERTIONS = ASSRT0,ASSRT1,ASSRT2,ASSRT3
ASSRT0 = assertion(cOCSPR,#0,crypt(inv(#0),cOCSPR))
cOCSPR = ocspr(name,ckey,time)
ASSRT1 = assertion(tsOCSPR,#0,crypt(inv(#0),tsOCSPR))
tsOCSPR = ocspr(#1,#2,time)
ASSRT2 = assertion(arcOCSPR,#0,crypt(inv(#0),arcOCSPR))
arcOCSPR = ocspr(#3,#4,time)
ASSRT3 = assertion(ARCH,#4,crypt(inv(#4),ARCH))
ARCH = archived(session(sid),certificate(name,ckey),
      contract(data), SIGNATURE,TIMESTAMP,ASSRT0,ASSRT1)
#0 in trustedCAKeys
pair(#1,#2) in trustedTSs
pair(#3,#4) in trustedARs
```

ASSRT0 is a judgment made about the validity of the *Client*'s certificate at the time *time* and signed by a certification authority trusted by *Client*. This trust relation is modeled by the fact that the public key of the certification authority is in the set *trusted-CAKeys* representing the public keys of the certification authorities trusted by *Client*. *ASSRT1,ASSRT2* represent similar judgments made about the certificates of the used timestamper and archiving service and signed by the same trusted certification authority. On the other hand *ASSRT3* models the fact that the data to be signed by *Client*, its digital signature together with a timestamp and all the proofs obtained for the different involved certificates have been successfully archived by an archiving third party which is in addition trusted by *Client* for this task: here also this trust relationship is modeled by the constraint: *pair(#3,#4) in trustedARs*. These assertions are encoded as certificates embedded in the messages. They represent either a condition ϕ evaluated before accepting a message or a guarantee ψ ensured by the service sending a message.

Finally the use of dashed communication lines in Figure 1 refers to additional constraints on the communication channels used by *Client* and *Goal*: in our example this turns to be a transport constraint requiring the use of *SSL*. We can express this constraint in our model by requiring that the concerned messages are ciphered by a symmetric key previously shared between both participants (the key establishment phase is not handled by the composed service).

In order to satisfy the requests of *Client*, *Goal* relies on a community of available services ranging from timestampers, and archiving third party to certification authorities.

These services are also given by their interface, i.e. the description of the precise message patterns they accept and they provide in consequence. For instance Figure 2

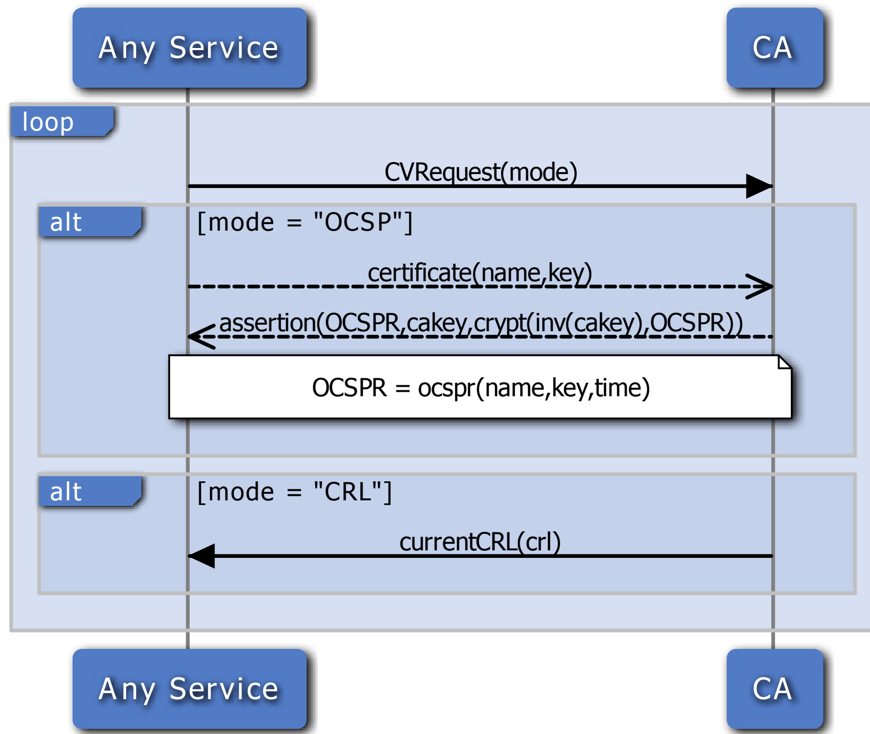


Fig. 2. Available services: Certification Authority

describes a certification authority *CA* capable of providing two sorts of answers when asked about the validity of a certificate: one is *OCSP*-based (i.e. based on the Online Certificate Status Protocol) and returns a proof containing a real-time time-bound for the validity of a given certificate; while the second only provides the classical Certificate Revocation List CRL. Intuitively by inspecting the composition problem one can think that to satisfy the *Client* request the second mode should always be employed with *CA* (provided it is also trusted by the *Client*). One can also deduce that some adaptation should be employed over the *Client*'s messages to obtain the right message patterns (possibly containing assertions) from the community (for example the use of the flag $OCSP$ with *CA*).

The solution we propose computes whenever it is possible the sequence of calls to the service community possibly interleaved with adaptations over the already received messages and permitting to satisfy the *Client*'s requests as specified in the composition problem.

4 Formal Description of Service Composition and Adaptation

4.1 Mediator synthesis

A web service is in standard way described in terms of the interface it presents to the outside world (the possible clients) using the *WSDL* [23] language. This description is structured into ports, each proposing a set of available operations. An operation is then defined by its given in-bound and out-bound message patterns; these patterns are usually described using the *XSD* [26] language and reflects the *XML* message structure. Security constraints can then be defined on top of the service interface description using *WS-SecurityPolicy* [16] annotations. Such annotations can occur at any level in the *WSDL* binding the levels they occur into the security constraints they carry. They range from the service to the message level and typical examples are an *SSL* transport requirement for the whole service or the need to cipher or digitally sign a certain part inside a message pattern (in-bound or out-bound to some operation). We note that the use of *XSD* for the description of message patterns permits the use of the *XPATH* [25] language to write the queries identifying parts inside these message patterns which simplifies the writing of message-level security constraints. We put the focus on SOAP-based (in contrast with RESTful-based) web services. These services rely on the *SOAP* [19] protocol that encapsulates the messages described in the *WSDL* specification of the service. We claim that after (automated) analysis we can collect from the different specification files the descriptions of the different message patterns in-bound and out-bound to all the operations of the service and corresponding to the messages really exchanged by the service (*SOAP* encapsulation included). These descriptions are discussed below.

4.2 Representation of messages and security constraints

We use first-order terms to represent the messages exchanged by a service. We recall this notion below.

Terms We consider an infinite set of free constants $Consts$ and an infinite set of variables $\mathcal{X}$. For each signature $\mathcal{F}$ (*i.e.* a set of function symbols with arities), we denote by $\mathrm{T}(\mathcal{F})$ (resp. $\mathrm{T}(\mathcal{F},\mathcal{X})$) the set of terms over $\mathcal{F} \cup Consts$ (resp. $\mathcal{F} \cup Consts \cup \mathcal{X}$). The former is called the set of ground terms (or messages) over $\mathcal{F}$, while the latter is simply called the set of terms over $\mathcal{F}$. Given a term t we denote by $\mathrm{Var}(t)$ the set of variables occurring in t. A substitution σ is an idempotent mapping from $\mathcal{X}$ to $\mathrm{T}(\mathcal{F},\mathcal{X})$ such that $\mathrm{Supp}(\sigma) = \{x \mid \sigma(x) \neq x\}$, the *support* of σ, is a finite set. The application of a substitution σ to a term t (resp. a set of terms E) is denoted $t\sigma$ (resp. $E\sigma$) and is equal to the term t (resp. E) where all variables x have been respectively replaced by the term $x\sigma$. Terms are manipulated by applying *operations* on them. These operations are defined by a subset $\mathcal{F}_p$ of the signature $\mathcal{F}$ called the *set of public symbols*. A context $C[x_1,\ldots,x_n]$ is a term in which all symbols are public and such that its nullary symbols are the variables $x_1,\ldots,x_n$. $C[x_1,\ldots,x_n]$ is also denoted C when there's no ambiguity and n is called its *length*. We define the *application* of a context C of length n over the sequence of messages $\langle m_1,\ldots,m_n\rangle$, denoted by $C\cdot\langle m_1,\ldots,m_n\rangle$, to be the image of $C[X_1,\ldots,X_n]$ by the substitution $\{X_j \to m_j\}_{1\leq j\leq n}$. An *equational theory*

$\mathcal{E}$ is defined by a set E of equations $u = v$ with $u, v \in \mathrm{T}(\mathcal{F}, \mathcal{X})$. We write $s =_{\mathcal{E}} t$ as the congruence relation between two terms s and t. This equational theory is introduced in order to specify the effects of operations on the messages and the properties of messages. We say that a term t *is deducible* from a set of terms E whenever there exists a sequence of elements $\langle t_1, \ldots, t_{k_t} \rangle$ in E and a context C_t of length k_t such that $C \cdot \langle t_1, \ldots, t_{k_t} \rangle =_{\mathcal{E}} t$.

XML messages We aim to represent a significant fragment of *XML* messages as described by the *XSD* language using first-order terms defined over a signature given below. The fragment we address corresponds to *XML* elements, described by sequential complex types, i.e. elements having an ordered and a fixed-cardinality set of children. We also abstract away the attributes in *XML* messages. To represent *XML* messages we define the following signature:

$$\mathcal{F} = \left\{ node_a^n, child_{\frac{i}{a}}^n \mid i \leq a \in \mathbb{N}, n \in Consts \right\} \cup \\ \{scrypt, sdcrypt, crypt, dcrypt, sign, verif, inv, invtest, \top\}$$

where the symbol $node_a^n$ represents an *XML* node named n (ranging over the set of constants $Consts$) and having a children. For each symbol $node_a^n$ we define the set of symbols $child_{\frac{1}{a}}^n, \ldots, child_{\frac{a}{a}}^n$ permitting to extract its children. In order to model security constraints holding over exchanged *XML* messages, we also represent the usual cryptographic primitives through the use of symbols: $scrypt$/$sdcrypt$ for symmetric encryption and decryption, $crypt/dcrypt$ for asymmetric encryption and decryption, $sign/verif$ for digital signature and its verification, inv to denote key inverses and $invtest$ permitting to test whether a pair of terms $\{t, t'\}$ verifies $t' = inv(t)$. The constant $\top$ is the result of a successful test. We denote by $\mathcal{F}_p$, the set of public symbols and assume in the remainder of this chapter that $\mathcal{F}_p = \mathcal{F} \setminus \{inv\}$.

Some of the symbols represent the possible operations on the messages. Their semantics is defined with the following equational theory:

$$\mathcal{E}_{XML} \left\{ \begin{array}{r} sdcrypt(scrypt(x, y), y) = x \ \ (D_s) \\ dcrypt(crypt(x, y), inv(y)) = x \ \ (D_{as}) \\ verif(x, sign(x, inv(y)), y) = \top \ (S_v) \\ child_{\frac{i}{a}}^n(node_a^n(x_1, \ldots, x_a)) = x_i \ (P_{\frac{i}{a}}) \\ invtest(x, inv(x)) = \top \ (I_v) \end{array} \right.$$

We say that a term t *is deducible* (we use also *deduced*) from a set of terms E whenever there exists a sequence of elements $\langle t_1, \ldots, t_{k_t} \rangle$ in E and a context C_t of length k_t such that $C \cdot \langle t_1, \ldots, t_{k_t} \rangle =_{\mathcal{E}_{XML}} t$.

4.3 Representation of services

We note that the *WSDL* specification of a web service does not precisely list any order of invocation for its operations but only gives their exhaustive list. Moreover this specification does not mention how the input parameters are related to the output parameters for a given operation. The *BPEL* [22] language allows reasoning about such

properties by permitting first to specify a certain workflow logic for the service, and second to specify all the manipulations needed to construct the sent messages given the received ones. In this sense *BPEL* describes *business processes* which are structured workflows of activities ranging over invocation of web service operations, providing of web services operations or manipulation of messages.

We consider services that do not contain any iteration or replication. Here we shall also abstract from internal actions and we shall focus on communications. Therefore a service S will be considered as a sequence of in- and out-bound messages denoted respectively *RCV(m)* and *SND(m)*.

We assume that all the services we consider are also described in terms of their respective *BPEL* specification and focus only on services described by linear processes, i.e. sequences of activities. Therefore a service S will be considered as a sequence of in- and out-bound messages denoted respectively $RCV(m)$ and $SND(m)$ as described by the following grammar:

$$\begin{array}{cc} P, Q := services & \\ 0 & \text{null service} \\ RCV(m) \cdot P & \text{input message} \\ SND(m) \cdot P & \text{output message} \\ P \parallel Q & \text{AC parallel composition} \end{array}$$

Parallel composition of services S_1 and S_2 is denoted by $S_1 \parallel S_2$. It is associative and commutative, and has a unit element 0, the null process. We consider a community to be a parallel composition of all its available services.

In the following we say that a term t is deducible (or deduced) from a service S whenever t is deducible from the sequence $\langle m_1, \ldots, m_{k_S} \rangle$ representing the messages received by the service S.

Transition semantics We introduce transition semantics to define how services are executed in interaction with their environment and in particular with clients. The state of a service S can be viewed as the list of remaining operations it has to perform to end properly. For instance the service in state $RCV(r) \cdot S'$ waiting for a message matching r proceeds with $S'\sigma$ if it is added in parallel to a service in state $SND(m) \cdot S$ such that m matches r with some substitution σ. In this case the latter service proceeds then with S. The global configuration is a pair $(\mathcal{S}, \mathcal{E})$ with first component the set of service states, and second component the set of messages that have been sent so far. The evolution of the global configuration is given by the transition rule:

$$\begin{array}{l} (RCV(r) \cdot S \parallel (SND(m) \cdot S' \parallel \ldots, \mathcal{E}) \xrightarrow{m} (S\sigma \parallel S' \parallel \ldots, \mathcal{E} \cup \{m\}) \\ \qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad \text{if } \exists \sigma, r\sigma = m \end{array}$$

Such transitions are called *communication transitions*.

The reception of a message instantiates the variables in the receive pattern. This instantiation is applied on the variables remaining in the process that describes the service. A *derivation* is a sequence of transitions. The *size* of derivation is the size of its sequence of transitions. We say that a service has *ended* in a derivation if it is reduced to a null process.

4.4 Web services composition problem

Composition Goal To answer a client $\mathcal{C}$ request we often need a new service $\mathcal{T}$ to be obtained as a composition of some of the ones that are available in the community. We define the composition goal as the ordered list of messages that $\mathcal{C}$ should receive from $\mathcal{T}$ and that $\mathcal{T}$ should receive from $\mathcal{C}$. Hence the composition goal is also a service that can be specified with the service grammar given above.

Composition mediator We exploit a derivation as follows to generate a mediator. The messages sent by the services are dispatched by the mediator and they can possibly be adapted before assigning them to the proper recipient. In order to express this adaptation capability of the mediator, we simply define the following transition rule:

$$(\mathcal{P},\ \mathcal{E}) \xrightarrow{C} (\mathcal{P},\ \mathcal{E} \cup \{m\})$$

if there exists a context C and $t_1, \ldots, t_n$ in $\mathcal{E}$ s.t $C[t_1, \ldots, t_n] =_{\mathcal{E}_{XML}} m$

We call such transitions *adaptation transitions*.

The problem we are interested in is to check whether a client $\mathcal{C}$ can be satisfied by a composition of services from the community. More formally we can state it as:

Service Composition Problem

Input:	A community of service $\mathcal{S} = \{S_1, \ldots, S_n\}$ A composition goal $\mathcal{C}$ (specified by the client requests)
Output:	A derivation from initial state $(\mathcal{S} \cup \{\mathcal{C}\},\ \emptyset)$ to a state where $\mathcal{C}$ has ended, and each service in $\mathcal{S}$ has either ended or is in its initial state, if such a derivation exists and the symbol $\perp$ otherwise.

In other word we have to check for the existence of a derivation (applying the transition rules) from an initial state where the client is put in parallel with the community of services and no messages have been sent so far, to a state where all requests from the client have been satisfied ($\mathcal{C}$ has ended) and the services from the community that have been initiated have properly terminated.

4.5 Solving the composition problem

Theorem 1. *The Service Composition Problem is NP-complete.*

Sketch of proof: We reduce the Service Composition Problem to showing the existence of an attack on a protocol built from the services and the client (given the $\mathcal{E}_{XML}$ theory). To ensure proper termination of services that are involved in an interaction with the client, we guess at the beginning whether a service S_i will be employed or not. Let $\{S'_1, \ldots, S'_m\}$ be the subset of services to be really employed. After this guessing step the composition problem is reduced to the reachability of a configuration $(0, \mathcal{E})$ from a configuration $(\mathcal{C} \parallel S'_1 \parallel \ldots \parallel S'_m, \emptyset)$ with $\{S'_1, \ldots, S'_m\} \subseteq \{S_1, \ldots, S_n\}$

For each service $S \cdot 0$ in $\{\mathcal{C}, S'_1, \ldots, S'_m\}$ we introduce a new constant c_S and transform the service $S \cdot 0$ into a service $\overline{S} = S \cdot SND(c_S) \cdot 0$. It is clear that a service

S reduces to the null process if, and only if, $\overline{S}$ sends c_S. Finally we add a monitor service M to the community that checks that all constants are sent. We let

$$M = RCV(c_{\mathcal{C}}) \cdot RCV(c_{S'_1}) \ldots RCV(c_{S'_m}) \cdot SND(secret) \cdot 0$$

It is clear that M sends $secret$ if and only if all the services $\mathcal{C}, S'_1, \ldots, S'_m$ reduce to the null process. Thus we have transformed the problem of the reachability of a configuration $(0, \mathcal{E})$ from a configuration $(\mathcal{C} \parallel S'_1 \parallel \ldots \parallel S'_m, \emptyset)$ into the problem of the reachability of a configuration $(P, \mathcal{E}')$ with $secret \in \mathcal{E}'$ from the initial configuration $(M \parallel C \parallel S'_1 \parallel \ldots \parallel S'_m, \emptyset)$. This latter problem is a classic problem for cryptographic protocols and is called the *Protocol insecurity problem*. Since the existence of an attack on a protocol is a problem known to be in NP [20] we can conclude. $\square$

The protocol insecurity problem corresponding to our composition problem can then be submitted to any state-of-the-art protocol verification tool capable of checking reachability properties. If the composition problem admits a solution we obtain an attack trace (or a conversation trace) describing how the intruder (or the mediator from a composition point of view) succeeded into satisfying the clients requests by applying its adaptation skills on messages exchanged with some services in the community.

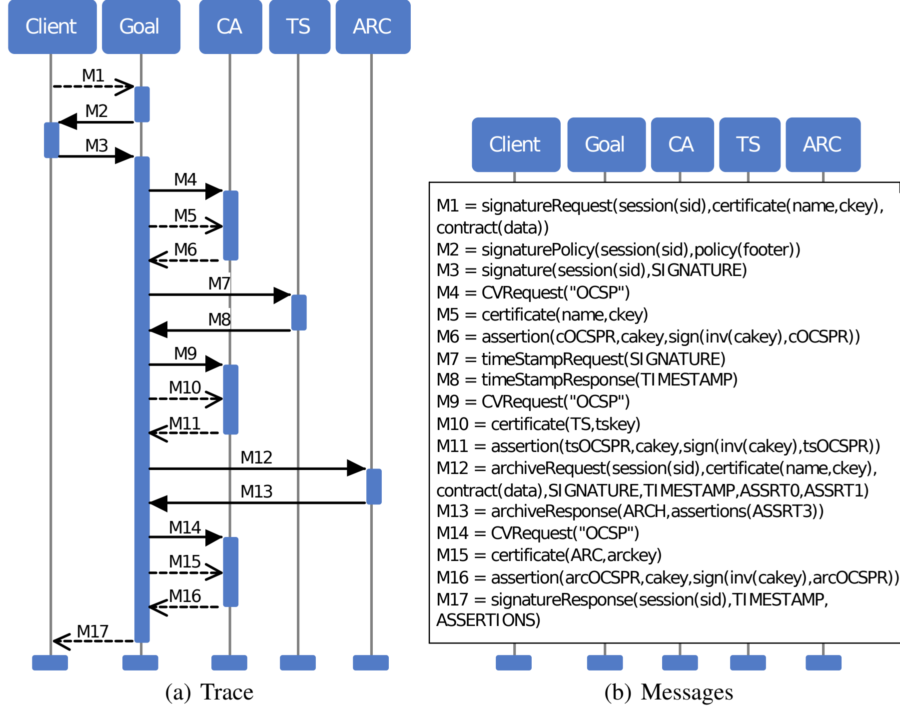
For instance Figure 3 illustrates the solution for the composition problem stated in the introductory example.

To fulfill the *Client*'s requests (messages $M1$ and $M3$) the mediator first calls (with messages $M4$ and $M5$) the certification authority service denoted by *CA* to obtain an assertion (message $M6$) stating the validity of the *Client*'s certificate. Then he calls the timestamper denoted by *TS* and trusted by the *Client* (with message $M7$) to obtain a timestamp (message $M8$) and subsequently *CA* (with messages $M9$ then $M10$) to obtain an assertion (message $M11$) stating the validity of *TS*'s certificate. Then he calls an archiving third party service *ARC* trusted by the *Client* (with message $M12$) to obtain assertions (in message $M13$) stating that the *Client*'s timestamped signature was correctly archived. Finally the mediator calls *CA* (with messages $M14$ and $M15$) to obtain the last needed assertion (message $M16$) stating the validity of *ARC*'s certificate, before successfully answering the last request of the *Client* (with message $M17$).

We remark that the mediator service M is easily extractable from the conversation trace. This can be done by running through all the communication steps in the conversation trace, putting the focus on those involving the mediator and updating its service description as follows:

- if the communication step is $S \rightarrow M : t$, append $RCV(t)$ to M;
- otherwise, append $SND(t)$ to M.

On the other hand, by definition of a composition problem, a corresponding solution should also describe all the adaptation steps that have to be performed by the mediator. These adaptation steps are abstracted away in the conversation trace depicted in Figure 3, which is a typical result of the state-of-the-art protocol verification tools. For instance, one could intuitively state that the message $M5$ sent by the mediator to *CA* can be extracted from the message $M1$ (previously sent to him by *Client*) by taking its second child. We present in Section 4.6 an automated procedure permitting to compute all these adaptation steps from a conversation trace similar to the one illustrated in Figure 3.



(a) Trace (b) Messages

Fig. 3. Solution for the composition problem in the introductory example

4.6 Generating the Mediator's Adaptation Steps

A conversation trace describes partially the solution of a given composition problem. Indeed, it only illustrates the ordered sequence of all communication transitions present in the corresponding derivation. Therefore for all communication step i in the trace, we can extract the following communication transition:

$$\delta_i = (\mathcal{P}_i,\ \mathcal{E}_i) \xrightarrow{m_i} (\mathcal{P}_{i+1},\ \mathcal{E}_i \cup \{m_i\})$$

where $(\mathcal{P}_i, \mathcal{E}_i)$ (resp. m_i) is the configuration before executing step i (resp. the message exchanged at the communication step i). We propose now to enrich this subsequence by interposing the missing adaptation transitions between the existing communication transitions, in order to reconstruct a complete solution of the problem.

Let us first remark that this enrichment may not be unique, and solutions of the composition problem can have an arbitrarily large size. Indeed if the derivation contains at least one adaptation transition, resulting in adding some message m to the environment one can build an infinity of messages (denoted by $T(m)$) by applying public symbols to m and since the environment is finite then one can enrich the derivation with an adaptation transition resulting in adding some new term from $T(m)$ to the environment. To ensure the finiteness of the enrichment we will consider only adaptation transitions that add subterms occurring in the conversation trace to the environment. Since the inference

system associated to the $\mathcal{E}_{XML}$ equational theory enjoys the locality [14] property[1], such an enrichment always exists.

To compute a derivation solving a composition problem given the corresponding conversation trace we first compute for each communication step i, the set $\mathcal{E}_i^{new}$ of the subterms $t_1^i, \ldots, t_k^i$ occurring in the trace and deduced by the mediator only starting from that step. We also keep track of $\mathcal{C}_i^{new}$ the sequence of contexts $\langle C_{t_1^i}, \ldots, C_{t_k^i} \rangle$ that permit to construct these subterms from the current knowledge in the order according to which they were constructed. Then we construct Δ_i for each communication step i as described below:

$$\Delta_i = \xrightarrow{C_{t_1^i}} (\mathcal{P}_{i+1},\ \mathcal{E}_i \cup \{m_i, t_1^i\}) \ldots \xrightarrow{C_{t_{k-1}^i}} (\mathcal{P}_{i+1},\ \mathcal{E}_i \cup \{m_i, t_1^i, \ldots, t_{k-1}^i\}) \xrightarrow{C_{t_k^i}}$$

Finally we construct:

$$\Delta = \delta_1 \Delta_1 \ldots \delta_n \Delta_n$$

which is a solution of the composition problem. To prove the last statement, it is sufficient to prove that *(i)* Δ is a derivation and *(ii)* Δ solves the composition problem and both are true by construction.

We put the focus now on the computation of the set $\mathcal{E}_i^{new}$ and the sequence $\mathcal{C}_i^{new}$ for all communication step i. We recall that $\mathcal{E}_i^{new}$ is the set of subterms occurring in the trace and deduced by the mediator only starting from the communication step i and that by construction we have: $\mathcal{E}_i^{new} = \mathcal{E}_i \setminus \mathcal{E}_{i-1}$ with $\mathcal{E}_0 = \emptyset$. One important remark here is that only receptions steps bring new knowledge to the mediator. After a reception of a message m_i, the mediator tries all the possible applications of equations in $\mathcal{E}_{XML}$ to his current knowledge including the message m_i and possibly computes new subterms occurring in the trace.

We now introduce the notion of *sequents* which we will use to compute the set $\mathcal{E}_i$ for all reception step i.

Definition 1. *Given a service S we call γ a* sequent *of S (denoted by $t_1, \ldots, t_k \vdash_f t_0$) an equality $t_0 =_{\mathcal{E}_{XML}} f(t_1, \ldots, t_k)$ where f is a public symbol and $t_0, \ldots, t_k$ is a sequence of subterms occurring in S. We call respectively t_0, f and the sequence $t_1, \ldots, t_k$ the* head*, the* symbol *and the* tail *of γ and denote them respectively by $h(\gamma)$, $s(\gamma)$ and $t(\gamma)$. We denote the set of all sequents of S by $\Gamma(S)$ and the set of all valid sequents at some step i by $\Gamma_i(S)$.*

We say that γ is *valid* at some step i when for all $0 \leq j \leq k$, $t_j \in \mathcal{E}_i$. We remark that if a sequent γ is valid at step i then its head is an element of $\mathcal{E}_i$. Indeed t_0 is deducible at step i by taking $t_0 =_{\mathcal{E}_{XML}} f(t_1, \ldots, t_k)$. We exploit this property to compute the set $\Gamma_i(Mediator)$ for all reception step i of the mediator service.

First we compute $\Gamma(Mediator)$ by running through all the subterms occurring in it and collecting the corresponding sequents. For example a subterm of the form $t = scrypt(k, m)$ will provide two entries: $k, m \vdash_{scrypt} t$ and $k, t \vdash_{sdcrypt} m$. For each computed sequent γ we define an integer called its *readiness* and initially set to the size of $t(\gamma)$. This integer is used to compute the validity of a sequent as explained

[1] Informally speaking, this property means that whenever a secrecy attack on a subterm t exists for a given protocol, then the intruder can reproduce a secrecy attack on t where he needs to derive only a subset of the subterms occurring in the protocol or in t.

further in this paragraph. For each subterm t we define two fields: $dstep$ which will hold the least reception step i where t is deduced and $context$ which will hold the context that permitted to deduce t. We also define for each subterm t of s a list of sequents $sequents(t)$ which is initialized by all the sequents γ' such that t appears in the tail of γ'.

Then the idea is to perform a fix-point computation per each step i corresponding to the set $\Gamma_i(Mediator)$. The detailed solution is illustrated by Algorithm 1 which relies on Algorithm 2 and both are given below.

Algorithm 1 Compute Deduced Subterms

Require: $Mediator : Service$
1: **for all** $RCV(m_k) \in Mediator$ **do**
2: $\quad deduce(m_k, k)$
3: **end for**

We start from subterms that are trivially deduced at some given step *i.e.* all the received messages clearly deduced at their corresponding reception step and try to deduce the new ones by checking whether there exists some sequents having their tails made only of already deduced subterms. In order to select these sequents we make use of the $readiness$ field attached to each sequent which is decremented each time one element in its tail is deduced (Algorithm 2, line 4). Since the $readiness$ field is initialized by the cardinality of its tail thus whenever $\gamma.readiness$ equals zero at some step then the sequent is also valid at that step.

Algorithm 2 deduce

Require: $t : subterm, i : step$
1: **if** $t.dstep > i$ **then**
2: $\quad t.dstep \leftarrow i$
3: $\quad t.context = \gamma$
4: $\quad$ **for all** $\gamma \in sequents(t)$ **do**
5: $\quad\quad \gamma.readiness - -$
6: $\quad\quad$ **if** $\gamma.readiness = 0$ **then**
7: $\quad\quad\quad \Gamma_i(p).add(\gamma)$
8: $\quad\quad\quad deduce(h(\gamma), i)$
9: $\quad\quad$ **end if**
10: $\quad$ **end for**
11: **end if**

Algorithm 1 runs in linear time in the size of mediator service represented as a directed acyclic graph (DAG).

4.7 Generating the Mediator's ASLan Specification

The ASLan Language

Background [2] ASLan (*AVANTSSAR Specification Language*) is defined by extending the *Intermediate Format* (IF) [7]. IF is an expressive language for specifying security protocols and their properties, based on multiset rewriting. As described in detail in [1], ASLan extends IF with a number of important features so as to express diverse security policies, security goals, communication and intruder models at a suitable abstraction level, and thereby allow for the formal specification and analysis of complex services and service-oriented architectures.

Most notably, ASLan extends IF with support of Horn clauses and LTL formulas. For instance invariants of the system can be defined by a set of (definite) Horn clauses. Horn clauses allow us not only to capture the deduction abilities of the attacker in a natural way, but also, and most importantly, they allow for the incorporation of authorization logics in specifications of services. Moreover, complex security properties can be specified in Linear Temporal Logic. As shown, for instance, in [3], this allows us to express complex security goals that services are expected to meet as well as assumptions on the security offered by the communication channels.

Syntax and Semantics Here, we recall the main features of ASLan, pointing the reader to [2] for more details on the language.

An ASLan file consists of several sections, among which:
emphSection Inits contains one or more initial states of the transition system. A state of a transition system is a set of variable-free facts.
emphSection Rules specifies the transitions of the transition system. A *transition* is a rule containing two parts, a left-hand side (LHS) and right-hand side (RHS). The rule can fire in a state whenever its LHS holds in that state. Moreover, a transition can be labeled with a list of existentially quantified variables whose purpose is to introduce new constants representing fresh data (e.g. nonces).

Example 1. Sample transition.

```
1 step sampleTransition(BankAgent):=
2  state_BankingService(BankAgent,1).
3  iknows(request)
4  =>
5  state_BankingService(BankAgent,2).
6  iknows(response)
```

where

- step is a keyword used to define a new transition;
- sampleTransition is a transition name;
- BankAgent is a parameter of the transition;
- state_BankingService (BankAgent,1), iknows(request), state_BankingService (BankAgent,2), iknows(response) are facts;
- state_BankingService (BankAgent,1).iknows(request) is the LHS of the transition;
- state_BankingService (BankAgent,2).iknows(response) is the RHS of the transition.

[2] excerpts from [2]

This transition represents the behavior of a banking service that receives a request and then reacts by replying with a response and moving to another state. More precisely, the transition can be fired if there exists a value val of variable BankAgent such that state_BankingService (val ,1) and iknows(request) are in the current state. The result of firing the transition is to replace the fact state_BankingService (val ,1) by the fact state_BankingService (val ,2) and add a new fact iknows(response).

Message sending and receiving are specified using iknows facts: the iknows in the LHS of a transition stands for receiving a message, while in the RHS of a transition it stands for sending a message. The fact iknows(request) of Example 1 will not disappear from the current state, because the predicate iknows is persistent: once a message is emitted, it becomes a part of the knowledge of the environment (i.e., of the network or of the intruder) and the environment does not "forget" it. If the LHS of a transition holds in the current state, then it is assumed that the knowledge (represented by a set of ground facts) of the corresponding service is enough to build the messages stated in iknows in the RHS of the transition.

We use one predicate per service to specify the service states. By convention the predicate name starts with state_ followed by the service name, e.g. state_BankingService from Example 1.

– Section Goals: contains security goals that can be defined as attack states (special states of the transition system) or by means of LTL formulae.

  Example 2. Sample attack state.

```
1    attack_state stateName(Msg):=
2        fact1(Msg).
3        fact2(Msg)
```

  Here, attack state stateName is reached, if there exists a value val of variable Msg such that fact1 (val) and fact2 (val) are in the current state of the transition system. Section HornClauses: contains a finite set of Horn clauses. They can specify, for instance, the authorization logic.

ASLan Generation Procedure We build a transition system specified in ASLan and representing a prudent implementation of the mediator service M. First we consider a list $A = \langle a_1, \ldots, a_n \rangle$ of all the subterms occurring in M deduced by M (at some communication step) such that for all $1 \leq i < j \leq n$, $a_i.dstep \leq a_j.dstep$ and we associate a fresh variable name per each atom in A through a bijective mapping $\sigma^{-1} : a_i \mapsto X_i$. We then create a state fact $state_M$ for M with the following profile type: $type(a_1) * \ldots * type(a_n) \rightarrow fact$. For each reception $RCV(m)$ in M we generate an ASLan transition τ having only the fact $iknows(\sigma^{-1}(m))$ in its RHS. We note that $\sigma^{-1}(m)$ is well defined, since every message m received by p is trivially reachable by her. For each emission $SND(m)$ in M we generate an ASLan transition τ having only the fact $iknows(\sigma^{-1}(m))$ in its LHS. Again we note here that every message m sent by the mediator M has been already deduced by him and thus $\sigma^{-1}(m)$ is well defined.

We introduce the variable renaming functions $\{VName_j\}_{1 \leq j \leq length(c)}$ to distinguish whether a value has been assigned to the variable X_m or not yet in a transition. For

each transition labeled by step j we respectively append to its LHS and RHS the facts $state_M(\langle VName_{j-1}(X_i)\rangle_{1\leq i\leq n})$ and $state_M(\langle VName_j(X_i)\rangle_{1\leq i\leq n})$ where the functions $VName_j$ map variables to ASLan variable names as follows:

$$VName_j(X_i) = \begin{cases} NI_X_i, & \text{if } \sigma(X_i).dstep \geq j; \\ X_i, & \text{otherwise.} \end{cases}$$

Finally we specify the initial state of the partner: $state_n(\langle ni_X_i\rangle_{1\leq i\leq n})$. Informally speaking we initialize (with dummy values) the variables corresponding to atoms that will be seen in received messages or generated as nonces in messages to be sent.

5 Experimental Results

5.1 Avantssar Platform

The above mediator construction has been implemented in AVANTSSAR Platform [5], as the Orchestrator module. A solution of the orchestration problem (the description of the mediator) is automatically extracted from the attack trace and then translated to ASLan using the *Trace2ASLan* module. It provides us with an operational implementation of the new feature provided by the composed service (or mediator). The combination of this implementation and the available services is validated with respect to regular security properties (and in prescript of all other partner services) in presence of an active intruder. The validation task is performed by the AVANTSSAR backends. The overall architecture is displayed in Figure 5. We have applied the AVANTSSAR Platform to three case-studies from the AVANTSSAR Test Library [4] (putting the focus on composition problems). In the following we describe one of these case-studies then we provide the execution times needed to solve the three corresponding orchestration problems as well as the one in the introductory example.

5.2 Running Case Study

Description. We have applied the AVANTSSAR Platform to a *digital contract signing* (DCS) case study, provided by OpenTrust. A *Business Portal* (BP) is provided to parties that plan to digitally sign a contract. The goal of this case study is to automatically compose a *security server* (SeS) that will interact with this security portal as well as with available services to satisfy the security constraints.We assume that the community of services available to compose SeS contains the following services:

Timestamper: An external service that provides a timestamping functionality. We abstract the protocol employed to communicate with this service with a simple payload exchange with an assertion guaranteeing the timestamp's freshness;

PKI: A *Public Key Infrastructure* (PKI) is employed to check the validation keys of the customers. Again, we abstract away the protocols employed to communicate with this or these PKI, and rely on an assertion to characterize the PKI functionality.

Archiver: An archiver service is also accessible. We are doing coarse grain composition, and simply abstract this service with an assertion stating that this service is trusted for long-term storage.

Security constraints. There were several constraints on the SeS. We list here the main ones: i) The exchange between the BP and the SeS must be secured using the HTTPS protocol ii) The contracts have to be stored securely.

Client service. In addition to the compliance with the above constraints, the SeS provider that we generate has to be able to be the *security server partner* in the BP process. To this end we have extracted the message exchange session with the *SeS partner* from the BP definition, and imposed that the generated service interacts with the BP. In the process the customers were completely abstracted away.

Orchestration. We run CL-Atse [21], one of the back-end of Avispa Tool suite, with the ASLan specification described above. It returns an attack on the secrecy of a newly introduced constant (signaling the end of the client service), which corresponds to the trace of the messages exchanged between the component services (and the Business Portal). This trace is displayed in Figure 4 and can be directly translated to a composed service for Digital Contract Signing.

Validation of the composed service. The trace computed for the composition and adaptation of services has been automatically translated to an ASLan specification of the mediator which has been added in parallel with the specification modeling the available services and the client. We have applied CL-Atse to verify some security properties of the resulting composition to validate the secrecy of the proof record and the authentication of the BP by the security server.

5.3 Testing benchmark

We briefly describe two other composition problems from the AVANTSSAR Library.

- *Public Bidding (PB)*: PB illustrates a secure document exchange, and aims at providing a web portal (the *Bidding Portal (BiP)*) to manage an online call for tender, and also Bidders' proposal submissions. The composition problem associated to this case-study amounts to generating the BiP's behavior satisfying the requests of two bidders.
- *Car Registration Process (CRP):* CRP models an e-government scenario, where a citizen have a secure access point, enabling communication with government officers (with a hierarchical chief, the *Officer Head* (OfH)) and service providers in an easily usable and secure way. From this portal, citizens may access a great variety of services with different authentication, authorization, and protection requirements. The composition problem associated to this case-study amounts to generating the OfH's behavior that leads the government officers to satisfy some citizen's car registration request.

One common concern in these case-studies (including DCS) is to guarantee the legal value of the electronic documents produced. The security requirements imposed on such

Table 1. Execution Times

Case-Study	Composition Time	ASLan Generation Time	Verification Time
Introductory Example	30 s	163 ms	708 ms
DCS	1 m	659 ms	3 m
PB	1 m	4 s	18 m
CRP	2 m	3 s	4 m

platforms are highly critical since the probative value of digitally signed documents relies on the conditions under which they have been produced and validated.

Table 1 illustrates per each case-study the different execution times needed to solve the composition problem, to generate the ASLan specification of the mediator (and add it in parallel to the initial specification) and finally to verify the resulting specification.

6 Conclusion

Relying on cryptographic protocols analysis methods we succeeded into solving a class of web services composition problem. The next step is to generate an executable mediator service from the produced ASLan specification. We also need to ensure for security reasons that the generated mediator code controls and protects as much as possible the messages it sends and receives. We have already obtained initial results in this direction. A second research line is to extend the presented works to services with more complex workflows.

References

1. AVANTSSAR. Deliverable 2.1: Requirements for modeling and ASLan v.1 `http://www.avantssar.eu`, 2008.
2. AVANTSSAR. Deliverable 2.3: ASLan final version with dynamic service and policy composition `http://www.avantssar.eu`, 2010.
3. AVANTSSAR. Deliverable 5.1: Problem cases and their trust and security requirements `http://www.avantssar.eu`, 2008.
4. AVANTSSAR. Deliverable 5.4: Assessment of the AVANTSSAR Validation Platform `http://www.avantssar.eu`, 2010.
5. AVANTSSAR. AVANTSSAR Platform `http://www.avantssar.eu`, 2010.
6. Armando, A., et al. The Avispa Tool for the automated validation of internet security protocols and applications, `http://www.avispa-project.org/`
7. AVISPA Deliverable 2.3: The Intermediate Format `www.avispa-project.org`, 2003
8. Berardi, D., Calvanese, D., De Giacomo, G., Lenzerini, M., Mecella, M. Automatic Composition of E-services That Export Their Behavior. ICSOC 2003:43-58.
9. Berardi, D., Calvanese, D., De Giacomo, G., Hull, R. and Mecella, M. Automatic composition of transition-based semantic web services with messaging. In *Proceedings of the 31st International Conference on Very Large Data Bases, Trondheim, Norway, August 30 - September 2, 2005*, pages 613–624. ACM, 2005.
10. Bultan, T. and Su, J. and Fu, X. Analyzing conversations of Web services In *Proceedings of the Internet Computing, IEEE*, pages 18–25, 2006.
11. Bultan, T., Fu, X., Hull, R., and Su, J. Conversation specification: a new approach to design and analysis of e-service composition. In *Proceedings of the International Conference on World Wide Web, WWW 2003*, pages 403–410, 2003.
12. Colomb, M., Di Nitto, E., and Mauri, M. CENE: A Service Composition Execution Environment Supporting Dynamic Changes Disciplined Through Rules Service-Oriented Computing - ICSOC 2006, 4th International Conference, Chicago, IL, USA, December 4-7, 2006,
13. Dolev, D., Yao, A. On the Security of Public-Key Protocols. IEEE Transactions on Information Theory **2**(29) (1983)
14. David A. McAllester Automatic Recognition of Tractability in Inference Relations Journal of the ACM. **40** (1993) 284–303
15. Monfroy, E., Perrin, O., Ringeissen, C. Dynamic Web Services Provisioning with Constraints. OTM Conferences (1) 2008: 26-43.
16. Oasis Technical Committee on Secure Exchange Ws-securitypolicy 1.2. `http://doc.oasis-open.org/ws-sx/ws-securitypolicy/200702/ws-securitypolicy-1.2-spec-cd-02.pdf`, 2007.
17. Pistore, M., Marconi, A., Bertoli, P., Traverso, P. Automated Composition of Web Services by Planning at the Knowledge Level, International Joint Conference on Artificial Intelligence (IJCAI) 2005.
18. Sheth, A.P., Kashyap, V.: So far (schematically) yet so near (semantically). In Hsiao, D.K., Neuhold, E.J., Sacks-Davis, R., eds.: DS-5. Volume A-25 of IFIP Transactions., North-Holland (1992) 283–312
19. World Wide Web Consortium. Simple Object Access Protocol 1.2. `http://www.w3.org/TR/soap12-part1`, Apr 2007.
20. M. Rusinowitch and M. Turuani. Protocol insecurity with finite number of sessions is NP-complete. In *Proc.14th IEEE Computer Security Foundations Workshop*, Cape Breton, Nova Scotia, June 2001.
21. Turuani, M. The CL-Atse Protocol Analyser. In Term Rewriting and Applications - Proc. of RTA, volume 4098 of Lecture Notes in Computer Science, pages 277-286, Seattle, WA, USA, 2006.

22. Oasis Consortium Web Services Business Process Execution Language Version 2.0. http://www.oasis-open.org/committees/documents.php?wg_abbrev=wsbpel, 23 January, 2006.
23. World Wide Web Consortium. Web Services Description Language (WSDL) 1.1. http://www.w3.org/TR/wsdl, 15 March, 2001.
24. Wu, Z., Gomadam, K., Ranabahu, A., Sheth, A., Miller, J. Automatic Composition of Semantic Web Services Using Process Mediation. ICEIS (4) 2007: 453-462
25. World Wide Web Consortium. XML Path Language (XPath) 2.0. http://www.w3.org/TR/xpath20/, 23 January, 2007.
26. World Wide Web Consortium. XML Schema Definition (XSD). http://www.w3.org/XML/Schema, March 2005.

7 Appendix

In Fig. 4 we have employed the following abbreviations:

$$
\begin{aligned}
M_1 = {} & \{signature_sent(s1 \cdot contract \cdot \{sha1(contract \cdot s1 \cdot signaturepolicy)\}_inv(pks1)) \\
& \cdot\{sha1(contract \cdot s1 \cdot signaturepolicy)\}_(inv(pks1))\}_kbpss \\
M_2 = {} & timestamp_ok(sha1(contract \cdot s1 \cdot signaturepolicy) \cdot n267(N) \\
& \cdot sha1(sha1(contract \cdot s1 \cdot signaturepolicy) \cdot n267(N))_(inv(pkts))) \cdot n267(N) \\
& \cdot\{sha1(sha1(contract \cdot s1 \cdot signaturepolicy) \cdot n267(N))\}_(inv(pkts)) \\
M_3 = {} & timestamp_ok(sha1(contract \cdot s1 \cdot signaturepolicy) \cdot n267(N) \\
& \cdot\{sha1(sha1(contract \cdot s1 \cdot signaturepolicy) \cdot n267(N))\}_inv(pkts)) \\
& \cdot timestamp_ok(sha1(contract \cdot s2 \cdot signaturepolicy) \cdot n258(N) \\
& \cdot\{sha1(sha1(contract \cdot s2 \cdot signaturepolicy) \cdot n258(N))\}_inv(pkts) \cdot n267(N) \\
& \cdot\{sha1(sha1(contract \cdot s1 \cdot signaturepolicy) \cdot n267(N))\}_inv(pkts) \cdot n258(N) \\
& \cdot\{sha1(sha1(contract \cdot s2 \cdot signaturepolicy) \cdot n258(N))\}_inv(pkts) \\
M_4 = {} & signature_sent(s1 \cdot contract \cdot \{sha1(contract \cdot s1 \cdot signaturepolicy)\}_inv(pks1)) \\
& \cdot signature_sent(s2 \cdot contract \cdot \{sha1(contract \cdot s2 \cdot signaturepolicy)\}_inv(pks2) \\
& \cdot\{sha1(contract \cdot s1 \cdot signaturepolicy)\}_inv(pks1) \cdot \{sha1(contract \cdot s2 \cdot signaturepolicy)\}_inv(pks2) \\
M_5 = {} & \{contract \cdot s1 \cdot signaturepolicy \cdot \{sha1(contract \cdot s1 \cdot signaturepolicy)\}_(inv(pks1)) \\
& \cdot n267(N) \cdot \{sha1(sha1(contract \cdot s1 \cdot signaturepolicy) \cdot n267(N))\}_(inv(pkts)) \\
& \cdot\{sha1(s1 \cdot pks1)\}_inv(pkpki) \cdot \{sha1(contract \cdot s2 \cdot signaturepolicy)\}_(inv(pks2)) \cdot n258(N) \\
& \cdot\{sha1(sha1(contract \cdot s2 \cdot signaturepolicy) \cdot n258(N))\}_inv(pkts) \cdot \{sha1(s2 \cdot pks2)\}_inv(pkpki)\}_kssarc \\
M_6 = {} & archive_ok(contract \cdot s1 \cdot signaturepolicy \cdot \{sha1(contract \cdot s1 \cdot SignaturePolicy(3))\}_inv(pks1) \\
& \cdot n267(N) \cdot \{sha1(sha1(contract \cdot s1 \cdot signaturepolicy) \cdot n267(N))\}_(inv(pkts)) \cdot \{sha1(s1 \cdot pks1)\}_inv(pkpki) \\
& \cdot\{sha1(contract \cdot s2 \cdot signaturepolicy)\}_inv(pks2) \cdot n258(N) \\
& \cdot\{sha1(sha1(contract \cdot s2 \cdot signaturepolicy) \cdot n258(N))\}_inv(pkts) \cdot \{sha1(s2 \cdot pks2)\}_inv(pkpki)) \\
M_7 = {} & archive_ok(contract \cdot s1 \cdot signaturepolicy \cdot \{sha1(contract \cdot s1 \cdot signaturepolicy)\}_inv(pks1) \\
& \cdot n267(N) \cdot \{sha1(sha1(contract \cdot s1 \cdot signaturepolicy) \cdot n267(N))\}_inv(pkts) \cdot \\
& \{sha1(s1 \cdot pks1)\}_inv(pkpki) \cdot \{sha1(contract \cdot s2 \cdot signaturepolicy)\}_inv(pks2) \cdot n258(N) \\
& \cdot\{sha1(contract \cdot s2 \cdot signaturepolicy \cdot n258(N))\}_inv(pkts) \cdot \{sha1(s2 \cdot pks2)\}_(inv(pkpki)))
\end{aligned}
$$

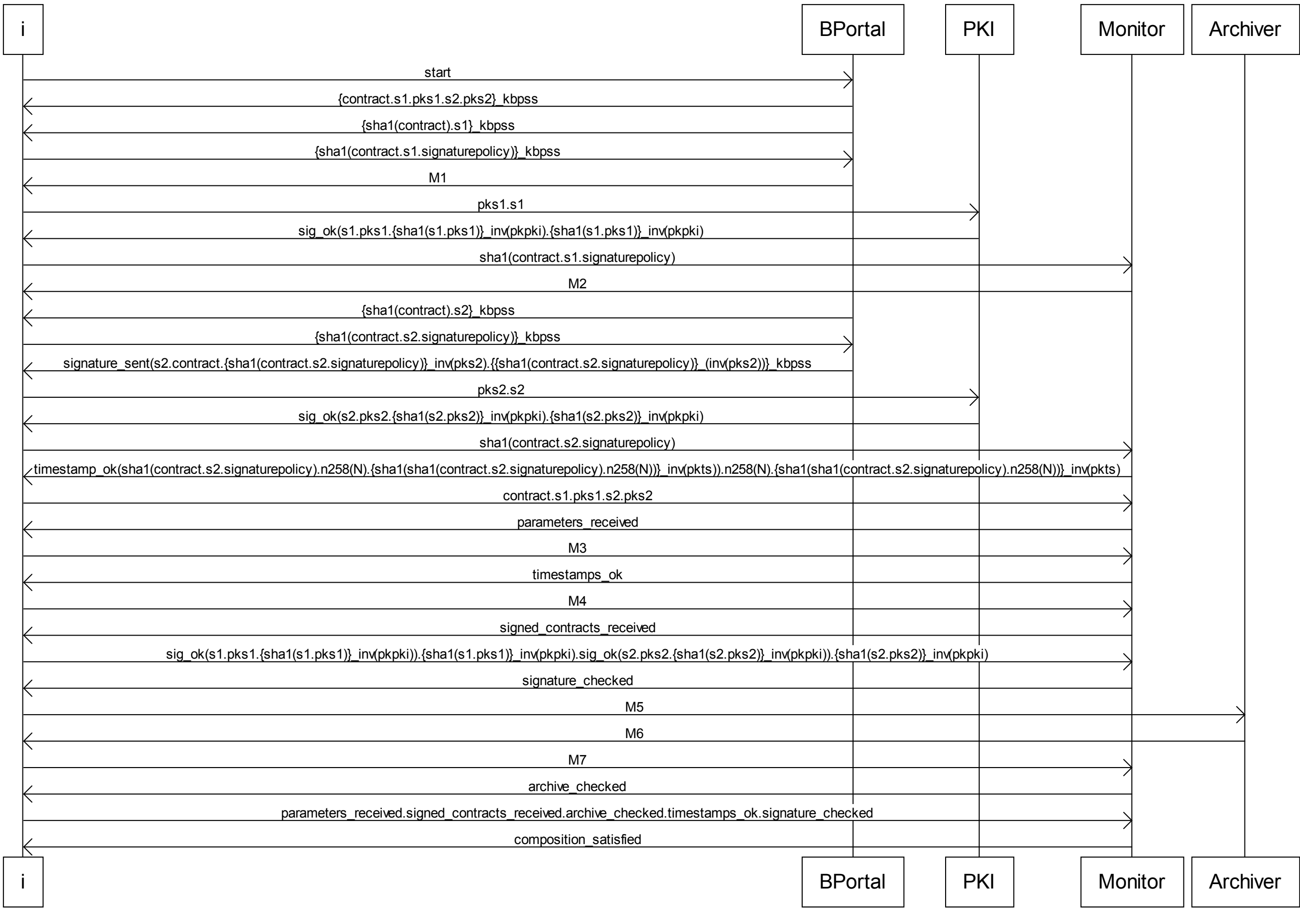
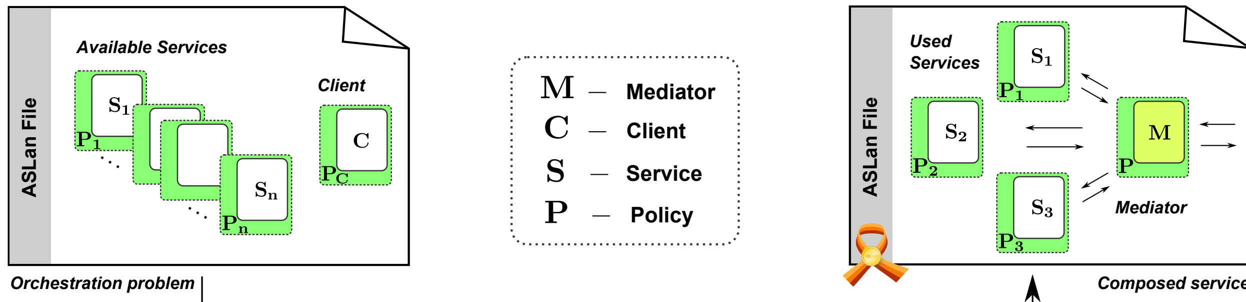
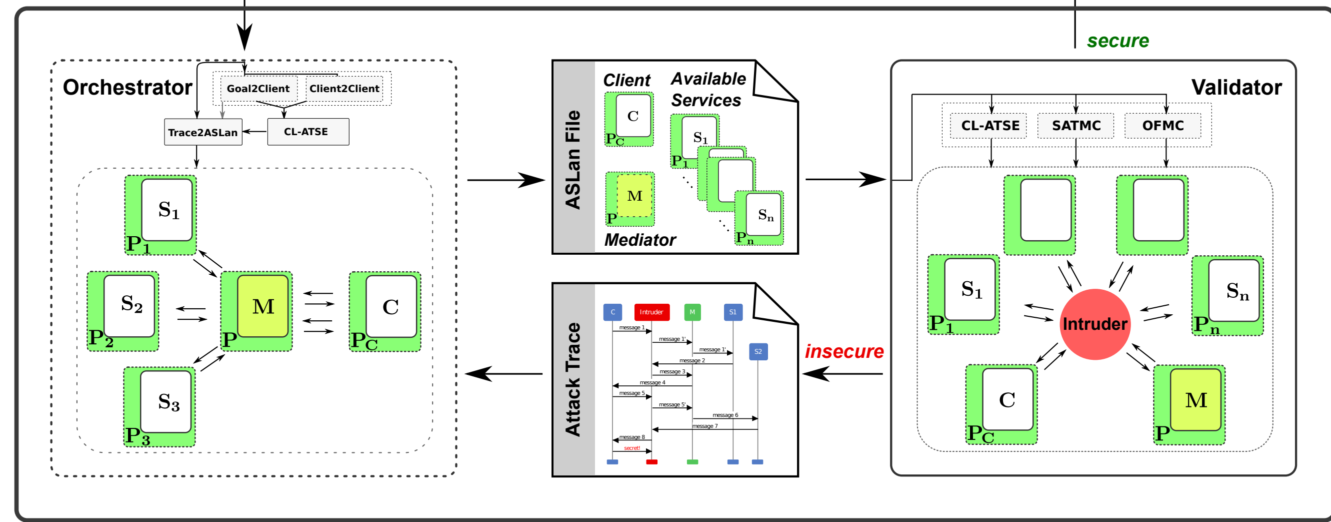


Fig. 4. Sequence Diagram for Digital Contract Signing (the intruder i stands for the security server)





AVANTSSAR Validation Platform

Fig. 5. Avantssar Platform