WSSMT: Towards the Automated Analysis of Security-Sensitive Services and Applications

Michele Barletta*, Alberto Calvi*, Silvio Ranise†, Luca Viganò* and Luca Zanetti†
*Dipartimento di Informatica, Università di Verona, Italy
Email: michele.barletta | alberto.calvi | luca.vigano @univr.it
†FBK-Irst, Trento, Italy
Email: ranise | zanetti @fbk.eu

***Abstract*—The security of service-oriented applications is crucial in several contexts such as e-commerce or e-governance. The design, implementation, and deployment of this kind of services are often so complex that serious vulnerabilities remain even after extensive application of traditional validation techniques, such as testing. Given the importance of security-sensitive service applications, the development of techniques for their automated validation is growing. In this paper, we illustrate our formal framework to the specification and validation of security-sensitive applications structured in two levels: the workflow level (dealing with the control of flow and the manipulation of data) and the policy level (describing trust relationships and access control). In our framework, verification problems are reduced to logical problems whose decidability can be shown under suitable assumptions, which permit the validation of practically relevant classes of services. As a by-product, it is possible to re-use state-of-the-art theorem-proving technology to mechanize the verification process. This is the idea underlying our prototype validation tool *WSSMT*, "Web-Service (validation by) Satisfiability Modulo Theories", which has been successfully used on two industrial case studies taken from the FP7 European project AVANTSSAR.**

I. Introduction

A service is a piece of software with a clearly defined set of interface functionalities that can be invoked according to a certain ordering specified by a *workflow* (WF). The *WF level* of a service establishes whether a certain operation can be executed if the values of the state variables (the *data flow*) of the service satisfy certain conditions (the *control flow*) and the values stored in the state variables may be updated. This situation is further complicated by the fact that one may create several instances of the same service: all the instances will share the same behavior but, at any given time, they may be in different states, i.e., distinct control locations and values of the state variables. Thus, (unique) identifiers are required to name the different particular instances. For example, in the case of web-services, several (executable) specification languages are available: the data part can be described by, e.g., WSDL [1], the control part by, e.g., BPEL [2], and the identifiers by URIs.

While the adoption of services augments the possibility of using resources and functionalities provided by third-party applications, it also augments the possibility of unauthorized use of shared resources and functionalities. This problem is particularly relevant for security-sensitive applications but can be a source of problems for virtually any system. An additional source of security problems is the fact that many deployed services work over a network (e.g., web-services over the Internet) where identities should be certified and trusted so as to enable the deployment of flexible access policies. In fact, one of the most relevant and hard-to-design parts of the security level of services is their *policy management* (PM) level. Policies specify what operations a service is granted or denied the right to execute, are usually expressed in terms of a set of basic facts, and are combined to form certain access rules. The basic facts depend on the particular application domain and are usually encapsulated in certificates whose possession enables the application of access rules. Since certificates can be produced or revoked at different time points, PM is an essentially dynamic activity. So, the PM level should be able to inspect part of the state of the WF of the service and, in turn, operations performed at the WF level can update the basic facts used to specify policies so that we have an interplay also from the WF to the PM.

A widespread design approach to control the delicate interplay between the WF and PM levels of a service consists of clearly separating them and identifying where and how they interact. This separation is beneficial in several respects for the design and maintenance of services, and also for their validation. However, the design, implementation, and deployment of services are often so complex that serious vulnerabilities remain even after extensive application of traditional validation techniques, such as testing. Hence, given the importance of security-sensitive service applications, a number of techniques for their automated validation have been developed.

This paper offers an overview of our approach to formally specify and validate security-sensitive services that are specified in the bi-dimensional space comprising the WF and the PM levels. First (Section II), we briefly describe the key ingredients of our declarative framework. Then (Section III), we introduce in our framework two formal verification problems that are crucial for the validation of services: *symbolic execution*, which is an extension of standard execution where many possible behaviors of the service are considered simultaneously, and *invariant verification*, which is the activity of checking that certain properties hold in any state of a service, in spite of the complexity of the computations taking place at the WF and PM levels. We also discuss (Section IV) how to mechanize the solution for the two verification prob-

lems considered before and describe a prototype tool, called *WSSMT*, which allows designers of service-oriented (SO) applications to gain confidence in their designs. Since our framework reduces verification problems to logical (satisfiability) problems, it is possible to use off-the-shelf state-of-the-art automated reasoning systems to automatically discharge the proof obligations encoding the satisfiability problems. The predictability of the behavior of the automated provers on the generated proof obligations is obtained by constraining the class of formulae used to describe the WF and PM levels together with those describing the possible executions of the service. In this way, it is possible to reuse decidability results for fragments of first-order logic that are supported by state-of-the-art theorem provers. Finally (Section V), we illustrate the practical viability of our approach by considering the validation of two case studies inspired by industrial systems, which have been considered in the context of the FP7 European project AVANTSSAR [3].

II. A FORMAL TWO-LEVEL SPECIFICATION FRAMEWORK

We regard a security-sensitive service as structured in two levels: the *workflow* level (WF)—dealing with the control of the flow and the manipulation of data—and the *policy management* level (PM)—describing trust relationships and access control rules. Each level is further structured in a *static* and a *dynamic* part; the former specifies the data structures manipulated by the service for the WF level or the relational structure used for the PM level, e.g., the user-role assignment table of a Role-based Access Control system [4], while the latter describes the possible executions of the service, e.g., how a certain integer variable storing the number of clients being served for the WF level or how a tuple is added/deleted to a relation in a database for the PM level. All the four components of our framework (static/dynamic part of the WF/PM) are symbolically represented by formulae of many-sorted first-order logic with equality[1], which is a well-studied logic that comes equipped both with a rich catalogue of decidable fragments (i.e., classes of formulae for which there exist algorithms capable of solving their satisfiability problems) and with several well-engineered automated theorem provers to support mechanical reasoning in the logic or its fragments. Our idea was to develop a declarative framework that permits one to specify the static and dynamic parts of the WF and PM levels, and then to reduce interesting verification problems to satisfiability problems in decidable fragments. This paves the way to building push-button validation techniques for security-sensitive service applications as demonstrated by our prototype tool *WSSMT*.

The notion of *theory*—i.e, a set of formulae built over a fixed set of constant, function, and predicate symbols—is used to formalize the static part of both the WF and the PM levels. The *satisfiability problem modulo a theory* T (see, e.g., [6]) consists of establishing whether there exists a model of T (i.e., a first-order structure that makes true all the formulae in T) that makes true a given first-order formula. A set Ax of *axioms* for a theory T are such that a model of Ax is also a model for T (several interesting theories admit finite sets of axioms).

[1]We assume the usual first-order notions of sort, term, literal, formula, (grounding) substitution, structure, satisfiability, and validity; see, e.g., [5].

A. WF and PM levels: static part

To formalize the static part of our framework, we consider three theories: T_{WF}, T_{PM}, and their intersection $T_{sub} = T_{WF} \cap T_{PM}$, called the *substrate theory*. Intuitively, as illustrated in Fig. 1, T_{WF} specifies the WF level, T_{PM} the PM level, and T_{sub} formalizes the interface between the two levels so as to ensure that they "agree" on certain notions. For example, T_{sub} uniquely identifies the principals involved in the service and possibly (an abstraction of) the structure of the resources that the SO application can access or make available.

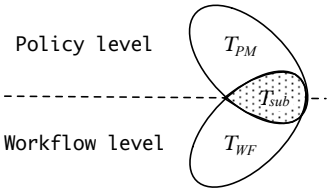

Fig. 1. Theories for the static part of the WF and PM levels

There is a well-established catalogue of (decidable) theories describing the most frequent data structures used in software systems, such as numbers, lists, and arrays. For an overview of the theories and the related decision procedures for their satisfiability problems, the reader is pointed to, e.g., [6]. Such theories can be used off-the-shelf also for the static part of the WF level. To illustrate, we consider the problem of formalizing the substrate theory (which can be seen as part of the WF level) for use-case scenarios of services, where typical executions are described (e.g., by using message sequence charts) involving only a known and finite number n of principals. In this case, we use a so-called enumerated datatype theory in which we have *(i)* a set $C := \{c_1, ..., c_n\}$ of finitely many constant symbols, corresponding to the unique identifiers of the principals, and *(ii)* the axioms

$$\forall x.\ \bigvee_{c_i \in C} x = c_i, \qquad \text{and} \qquad \bigwedge_{c_i, c_j \in C \text{ and } i<j} c_i \neq c_j$$

respectively stating that there are at most n identifiers and that there are at least n distinct identifiers. Another example of interesting substrate theory is the empty theory, which corresponds to assuming that there exists a finite but unknown number of identifiers for principals as we work in many-sorted first-order logic with equality (so that the equality sign is always interpreted as an equivalence relation). The empty theory is particularly useful when one considers invariant verification and wants to check that certain properties hold regardless of the number of principals involved in the computations.

Concerning the PM level, theories can be easily derived from access control policies expressed as logic programs (see, e.g., [7]). A *Horn theory* is a theory whose axioms are formulae of the form

$$\forall \underline{x}, \underline{y}.\ p(\underline{x}) \ \Leftarrow\ q_1(\underline{x}, \underline{y}) \wedge \cdots \wedge q_n(\underline{x}, \underline{y})$$

where $\underline{x}, \underline{y}$ are disjoint tuples of variables, and $p(\underline{x}), q_i(\underline{x}, \underline{y})$ are atoms (i.e., predicate symbols applied to a tuple of terms

or equalities) where, respectively, at most the variables in $\underline{x}$ and $\underline{x}, \underline{y}$ may occur (for $i = 1, ..., n$). Formulae written in this way are called *rules* in the logic programming literature, while $q_1, ..., q_n$ and p are called the *body* and the *head* of the rule, respectively.

We can then simply formalize in our framework the static part of a security-sensitive service as the *background theory* T_{SS}, obtained by taking the (set theoretic, since the theories as simply sets of formulae) union of T_{WF} and T_{PM}.

B. WF and PM levels: dynamic part

To formalize the dynamic part of a service, we consider an extension of the well-established notion of *guarded assignment*, where state variables are updated by applying a function of the actual values of the variables provided that a guard is satisfied (expressed as a condition again on the values of the state variables). In the following, let T_{SS} be the background theory for the security-sensitive service SS under consideration. Its state variables are partitioned in two disjoint tuples: $\underline{x}$ for the WF level and $\underline{p}$ for the PM level; each variable in $\underline{x}$ is associated to a sort of the background theory T_{SS} and each variable in $\underline{p}$ is a predicate symbol not mentioned in T_{SS}. Using a symbolic representation of state spaces (see, e.g., [8]), a set s of states of the service SS is represented by a *state formula* $\varphi_s(\underline{x}, \underline{p})$ in T_{SS}, whose only free variables are in $\underline{x}$ or $\underline{p}$. We consider state formulae belonging to particular fragments of first-order-logic to describe initial sets of states:

$$\forall \underline{i}.\ (\varphi_I^{WF}(\underline{i}, \underline{x}) \wedge \forall \underline{z}.\ \bigwedge_{p \in \underline{p}} p(\underline{z}) \Leftrightarrow \varphi_{I,p}^{PM}(\underline{i}, \underline{p}, \underline{z}))\,, \qquad (1)$$

where $\underline{i}, \underline{z}$ are tuples of variables ranging over identifiers, φ_I^{WF} is a quantifier-free formula of T_{WF}, and $\varphi_{I,p}^{PM}$ is a quantifier-free formula of T_{PM} for each $p \in \underline{p}$.

To describe transitions, we consider a particular class of formulae, called *transition formulae*, corresponding to guarded assignments over unprimed and primed variables $\underline{x}$ and $\underline{p}$ (as usual, unprimed variables refer to the actual values of the state variables, while primed variables refer to their values after the execution of a transition):

$$\exists \underline{i}, \underline{d}. \left(\begin{array}{l} G(\underline{i}, \underline{d}) \ \wedge \ \bigwedge_{x \in \underline{x}} x' = f_x(\underline{x}, \underline{i}, \underline{d}) \ \wedge \\ \forall \underline{z}.\ \bigwedge_{p \in \underline{p}} p'(\underline{z}) \Leftrightarrow \varphi_p(\underline{i}, \underline{p}, \underline{z}), \end{array} \right) \qquad (2)$$

where $\underline{i}, \underline{z}$ are tuples of variables ranging over identifiers, and $\underline{d}$ is a tuple of variables ranging over some sorts of T_{WF}. Furthermore, G is a quantifier-free formula called the *guard* of the transition (representing its enabling condition), f_x is a term of T_{SS} representing the *updates of the WF level*, i.e., functions of the identifiers $\underline{i}$ of the principal involved in the transition, the actual values $\underline{x}$ of the state variables of the WF level, and some arbitrary values $\underline{d}$ (in this way, we can also express non-deterministic updates). Finally, φ_p is a quantifier-free formula in T_{PM} whose only free variables are in $\underline{i}, \underline{z}$, or $\underline{p}$ representing the *updates of the PM level*. For a simple illustration of the application of our framework to a case study, the reader is pointed to section V-A.

The restrictions imposed on the shapes of state and transition formulae (plus some additional assumptions on the WF, PM, and substrate theories) are crucial to derive decidability results for the satisfiability problems of the proof obligations arising in the validation of security-sensitive services. This is described in the next section.

III. Verification problems for validation

We now introduce two verification problems in our framework in order to support validation of security-sensitive services. The first is *symbolic execution*, which is an extension of standard execution where many possible behaviors of the service are simultaneously considered. This is achieved by using state formulae to describe sets of valuations of the state variables. For each possible valuation of the variables, there is a concrete system state that is being indirectly simulated. This technique is particularly useful for the design of security-sensitive services for which it is standard practice to identify several scenarios (e.g., by using message sequence charts involving some interaction between the principals involved in the operation of the service) as execution paths that the system should support. Given the high degree of non-determinism and the subtle interplay between the WF and the PM levels, it is often far from being obvious that the service just designed allows one or many of the chosen scenarios. Symbolic execution allows the designer to consider several scenarios and thus perform a preliminary sanity check that the service supports some intended scenarios of execution.

The second verification problem is *invariant verification*, which is the activity of checking that a certain property holds in every state of the service that can be reached by performing a (finite) sequence of transitions. After checking that the service under consideration is capable of achieving certain goals (by using symbolic simulation), the second step is to assess that certain properties are preserved by any transition of the service, in spite of the complexity of the computations taking place at the WF and PM levels. Invariants can encode several interesting security properties; in fact, safety can be characterized as "something bad never happens" (see, e.g., [8]). In our framework, we further restrict invariants to be encoded by state formulae; in general, safety properties can be expressed by a class of temporal logic formulae (see again [8] for an exhaustive discussion on this point). The idea is to check that the state formula φ is an invariant, by using induction over the computations of the service as follows (this is known in the literature as the INV rule [8]): (*base*) the formula describing the set of initial state implies φ, and (*step*) if φ holds in a given state, then φ holds in the state obtained by executing τ, for each transition τ of the service. If φ satisfies both (*base*) and (*step*), we say that it is an *inductive invariant*. Unfortunately, the class of state formulae encoding invariant properties properly contains inductive invariants. Thus, it is frequently the case that although the state formula φ is an invariant, it is not inductive and the INV rule fails. The problem is then to guess an appropriate inductive invariant θ such that θ implies φ. Indeed, guessing requires some

ingenuity and several methods have been proposed in the literature that can be adapted to our context. Such methods are out of the scope of this paper and we point the interested reader to [9] for an overview.

Formally, we consider that the state variables $\underline{x}$ and $\underline{p}$ of the service SS to be validated have been identified together with its background theory T_{SS}.

A. Symbolic execution of security-sensitive services and applications

In any scenario provided with the service, there is only a known and finite number of principals. Thus, the substrate theory $T_{sub} \subseteq T_{SS}$ is assumed to be an enumerated datatype theory (recall its definition in Section II). Notice that the assumption to have a fixed set of identifiers for principals allows us to replace existential quantifiers over identifiers in transition formulae (2) with (finite) disjunctions. So, for symbolic execution, a transition formula is of the form

$$\begin{array}{l}\exists \underline{d}.(\bigvee_{\underline{i} \in C}(G(\underline{i},\underline{d}) \ \wedge \ \bigwedge_{x \in \underline{x}} x' = f_x(\underline{x},\underline{i},\underline{d}) \ \wedge \\ \quad \forall \underline{z}.\bigwedge_{p \in \underline{p}} p'(\underline{z}) \Leftrightarrow \varphi_p(\underline{i},\underline{p},\underline{z}))),\end{array} \tag{3}$$

where C is the finite set of principal identifiers. Now, since the principal that executes a transition in a scenario is precisely identified, instead of the disjunction (3), we only consider the disjunct that mentions the principals involved in a transition at a given step of a scenario. Let us consider two state formulae φ and ψ of the form (1), where the universal quantifier is replaced by a finite conjunction because T_{sub} is an enumerated datatype theory by assumption. An instance of a transition of the service, namely a disjunct τ of (3), *leads the service from state φ to state ψ* if the formula $(\varphi \wedge \tau) \Rightarrow \psi$ is a logical consequence of T_{SS} or, equivalently by refutation, $\varphi \wedge \tau \wedge \neg\psi$ is unsatisfiable modulo T_{SS}. Given a sequence of state formulae $\varphi_0, ..., \varphi_{n+1}$ of the form above and a sequence of instances of transitions $\tau_1, ..., \tau_n$, i.e., disjuncts of the form (3), the *symbolic execution problem* is to establish if τ_i leads the service from state φ_i to state φ_{i+1}, for each $i = 0, ...n$.

This problem can be reduced to n satisfiability problems modulo the theory T_{SS} of formulae in a certain form. Under suitable assumptions on the component theories T_{WF} and T_{PM}, it is possible to show that each one of such satisfiability modulo theory problems is decidable and conclude that also the symbolic execution problem is so. We omit here the technical details and point the interested reader to [10].

B. Invariant verification of security-sensitive services and applications

For invariant verification, there is a finite but unknown number n of principals and we want to check that a given property φ is an inductive invariant regardless of n. Thus, the substrate theory $T_{sub} \subseteq T_{SS}$ is assumed to be empty as this allows us to consider all finite sets of principal identifiers (recall the discussion in Section II about this point). We also assume that the initial state formula I as well as φ are of the form (1), and that transition formulae $\tau_1, ..., \tau_n$ are of the form (2).

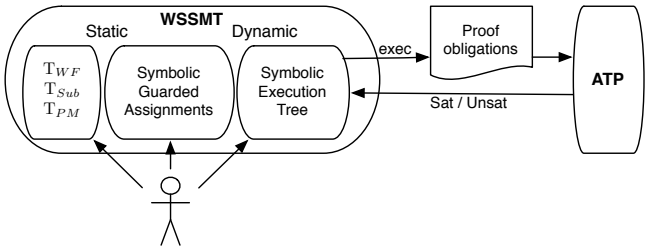

Fig. 2. High level architecture view of WSSMT

The inductive argument sketched above for inductive invariant verification can be encoded by the following two proof obligations:

(*base*): $I \Rightarrow \varphi$ is a logical consequence of T_{SS} or, equivalently, $I \Rightarrow \varphi$ is unsatisfiable modulo T_{SS} and

(*step*): $(\varphi(\underline{x},\underline{p}) \wedge \tau_i(\underline{x},\underline{p},\underline{x}',\underline{p}')) \Rightarrow \varphi(\underline{x}',\underline{p}')$ is a logical consequence of T_{SS} or, equivalently, $\varphi(\underline{x},\underline{p}) \wedge \tau_i(\underline{x},\underline{p},\underline{x}',\underline{p}') \wedge \neg\varphi(\underline{x}',\underline{p}')$ is unsatisfiable modulo T_{SS}.

The two satisfiability problems above are decidable under suitable assumptions (see again [10] for details) and hence also the invariant verification problem is so.

IV. WSSMT: A MECHANIZATION OF THE SPECIFICATION AND VERIFICATION PROCESS

There are two ways to mechanize the formal framework introduced in Section II together with the two verification problems defined in Section III: the implementation of an *ad hoc* tool or the re-use of existing tools via a suitable front-end. Since the verification problems are reduced to satisfiability problems modulo theories, it is highly desirable to exploit the cornucopia of well-engineered and scalable Automated Theorem Proving (ATP) systems such as resolution-based provers and Satisfiability Modulo Theories (SMT) solvers. We chose the second option and implemented a tool called *WSSMT*, acronym of "Web-Service (validation by) Satisfiability Modulo Theories." A detailed description of the tool and its implementation can be found in [11], here we only sketch its main functionalities and architecture, which is depicted in Fig. 2.

The main goal of WSSMT is to help users writing specifications of security-sensitive services structured along the previously identified directions: WF/PM levels and static/dynamic parts. Once the (structured) specification is finished, the front-end will enable the user to create and manipulate a *symbolic execution tree*, which compactly represents several possible symbolic executions of the service under consideration. Indeed, to create such a tree, whose nodes are labelled with state formulae and edges with (instances of) transitions, the front-end must create the appropriate proof obligations (as explained in Section III) and then invoke an available ATP system, chosen by the user among those available in the back-end. Once the ATP has established the satisfiability of the proof

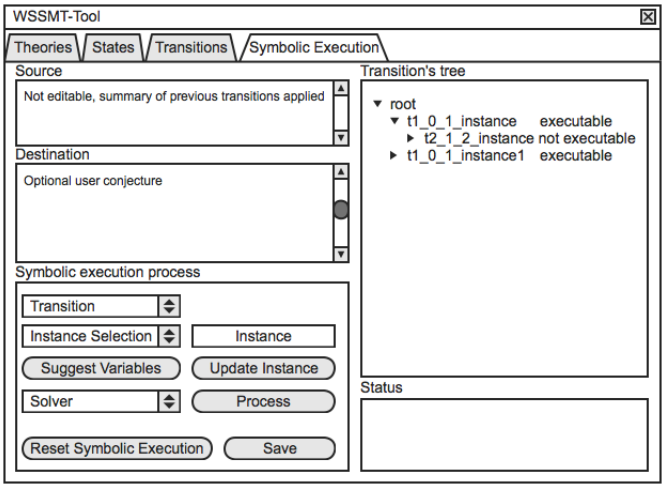

Fig. 3. Symbolic Execution tab

obligation, the front-end updates the symbolic execution tree or complains about the impossibility of executing the chosen transition. The client-server architecture of WSSMT follows these observations as shown in Fig. 2.

The front-end is organized in several tabs corresponding to the various ingredients of our specification and verification framework: *Theories*, *States*, *Transitions*, and *Symbolic Execution*. The first two tabs describe the static part of the specification and are structured in such a way to specify the WF and PM levels independently. The *Transitions* tab allows the user to enter transitions in the form (2). The *Symbolic Execution* tab, depicted in Fig. 3, is split in two parts: on the left, the user can enter the symbolic step to be checked for executability, while the right part shows the symbolic execution tree that represents one or more possible scenarios of execution. More precisely, the left part shows the state formula from which the symbolic execution step is taken (labelled *Source*) and allows the user to enter the formula to which the execution step should lead (labelled *Destination*) together with a transition chosen from the list of available transitions (combo labelled *Transition*), whose identifiers have been instantiated as explained in the combo labelled *Instance selection*. To send the resulting proof obligation to a back-end ATP system, the user should press the button *Process*.

To ease portability and modularity, WSSMT has been implemented in Java 1.5 as an Ecplise 3.5 plug-in by exploiting the SWT and JFace libraries [12], [13] for the creation of multi-platform graphical user interfaces. The concrete input language of state and transition formulae, as well as of axioms of the theories in WSSMT, is the *DFG* syntax [14]. It has been chosen because it supports many-sorted first-order logic, it is easy to extend, and several tools are available for its parsing and translation. in the distribution of the ATP system *SPASS* [15], a state-of-the-art resolution-based prover. Currently, WSSMT has been used with SPASS and the SMT solver Z3 [16]. The former was chosen because it has the same input language as the front-end so that it is trivial to generate the proof obligations to support symbolic execution. However, it would be easy to integrate any ATP system whose input language is the TPTP format [17] as there exists a translator from the DFG syntax to the TPTP format in the distribution of SPASS. This is left to future work.

Z3 was chosen because it is one of the best SMT solvers (according to the last competition for such tools [18]) and complements the reasoning capabilities of SPASS by providing support for ubiquitous theories as decidable fragments of arithmetics (while SPASS only supports reasoning in pure first-order logic). It was easy to create a translator from the DFG syntax to the SMT-LIB input language [19], which is one of the input languages of Z3. Furthermore, integration of further SMT solvers can be done seamlessly as the SMT-LIB language is their common input language. The evaluation of the advantages of having several SMT solvers available as back-ends in WSSMT is also left to future work.

The ATP systems are invoked via calls to the operating system provided by Java and their results are parsed by the front-end in order to update the symbolic execution tree accordingly (along the lines explained in Section III). Some remarks about the performances of the ATP systems to discharge typical proof obligations arising in WSSMT are given in the next section.

V. Case studies

To conclude the paper, we show the practical viability of our framework to validate designs of security-sensitive services by reporting our experience of applying WSSMT to the specification and verification of two case studies of the European project AVANTSSAR (a detailed description of these and other case studies can be found in [20]). The former is an E-government application and the latter is a digital contract signing protocol.

A. Car Registration

The first scenario considers the situation where the request to register a new car by a citizen is submitted to an on-line service provided by a Car Registration Office (CRO). For lack of space, we consider the following simplified scenario (a more realistic specification developed in our framework can be found in [10]). A citizen submits his request to an employee of the CRO, called `Ed`, who checks if the request can be accepted, according to some criteria that are abstracted away in the specification. In case the request can be accepted, `Ed` must store it in a central database. To this end, he must send a request to a Central Repository (`CRep`) that checks if `Ed` is entitled to permanently store such a request and, in this case, stores the document in the database. `Ed` can acquire the right to store documents in the central database only if the head of the CRO, called `Helen`, decides so. The certificates of the roles (being an employee or a head) of the various principals are generated by a trusted Certification Authority (`CA`).

We now proceed to formalize this scenario in the framework of Section II. The substrate is formalized as an enumerated datatype theory containing four principals: `Ed`, `Helen`, `CA`,

CRep. We model the network over which the principals exchange messages as a set N (thus, N is the only state variable of the WF level): sending a message m over the network corresponds to adding it to N, i.e. $\{m\} \cup N$, and receiving it corresponds to testing if m is a member of N, i.e. $m \in N$. A message is a tuple $\langle p, x, q \rangle$ where p is the sender, q is the receiver, and x is the payload of the message. In the following, we will use standard set-theoretic notation; it is straightforward to encode everything in first-order logic (see, e.g., [10] to see how). The PM level also contains just one state variable $knows_0$, which is a predicate with two arguments: the first is a principal and the second is a piece of information (called *infon* for brevity). Intuitively, $knows_0(p, x)$ encodes the fact that the piece of information x is part of the internal knowledge of principal p. The PM level has a rich background theory, which is inspired to the logical language DKAL [21] for authorization and trust management. The PM theory is Horn and its axioms are the following rules:

$$
\begin{array}{ll}
(IK) & \mathsf{knows}(p, x) \Leftarrow knows_0(p, x) \\
(TA) & \mathsf{knows}(p, x) \Leftarrow \mathsf{knows}(p, \text{“}q \text{ said } x\text{”}) \ \wedge \\
 & \qquad\qquad \mathsf{knows}(p, \text{“}q \text{ tdOn } x\text{”}) \\
(P1) & \mathsf{knows}(\mathtt{CRep}, \text{“}p \text{ cans”}) \ \Leftarrow \\
 & \qquad\qquad \mathsf{knows}(\mathtt{CRep}, \text{“}r \text{ ish”}) \ \wedge \\
 & \qquad\qquad \mathsf{knows}(\mathtt{CRep}, \text{“}p \text{ ise”}) \ \wedge \\
 & \qquad\qquad \mathsf{knows}(\mathtt{CRep}, \text{“}r \text{ said } p \text{ cans”}) \\
(P2) & \mathsf{knows}(p, \text{“}\mathtt{CA} \text{ tdOn } x\text{”}) \\
(P3) & \mathsf{knows}(p, \text{“}q \text{ tdOn } \mathtt{CA} \text{ said } x\text{”}) \\
(P4) & \mathsf{knows}(p, \text{“}q \text{ tdOn } r \text{ said } q \text{ cans”}) \Leftarrow \\
 & \qquad\qquad \mathsf{knows}(p, \text{“}r \text{ ish”})
\end{array}
$$

where knows is a predicate symbol of the PM theory; p, q, and r are variables ranging over principals; and x is a variable ranging over infon. Quotes creates complex infons by combining principals with the keywords said, tdOn, ise, and ish: “p said x” denotes the fact that principal p communicates the infon x, “p tdOn x” says that p is trusted on communicating the infon x (if p decides to communicate it), “p ise” (“p ish”) is the certificate that p is an employee (head, respectively), and “p cans” is the certificate that the principal p can store documents in the central database. Although quotations can be easily represented as terms of first-order logic, we prefer to use this informal notation here to enhance readability. The intuitive reading of the first two rules above is the following: (IK) says that internal knowledge is knowledge (in fact, a principal can gain knowledge also by deriving it using the rules listed above); (TA) regulates the application of the trust relationships among principals to derive further knowledge, by stipulating that a principal p can acquire the infon x if x has been communicated to p by another principal q, which is trusted on communicating x by the principal p itself. This rule is crucial for accepting/refuting the certificates presented by the various principals to support a certain request (e.g., Ed asking CRep to store a certain document in its database). The certificates and the trust relationships among the principals of the scenario described above are formalized by the last four rules above, whose intuitive meaning is the following. $(P1)$ says that CRep grants the right to store documents in its database to a principal p provided that it knows that p is an employee of the CRO and that the head q of p has decided to grant him this opportunity (notice that the fact that q is the head of the CRO should be certified). The remaining rules represent the following trust relationships: $(P2)$ says that any certificate emitted by CA is trusted, $(P3)$ says that any principal is trusted when it hands over a certificate emitted by CA, and $(P4)$ says that a principal p can trust a certificate concerning the fact that another principal can store documents in the central database if p knows that the principal emitting the certificate is the head of the CRO.

The possible operations of the service considered in this case study can be modeled by the following two transition formulae, which are both of the form (2):

$$\exists p, q.\, \exists x. \left(\begin{array}{l} \mathsf{knows}(p, x) \ \wedge \ N' = N \cup \{\langle p, \mathsf{said}(x), q \rangle\} \ \wedge \\ \forall z_1, z_2.(knows'_0(z_1, z_2) \Leftrightarrow knows_0(z_1, z_2)) \end{array} \right)$$

abbreviated with $\exists p, q.\, \exists x.\, k2m(p, x, q)$, which describes the situation where a principal p is willing to send a message with payload $\mathsf{said}(x)$ to another principal q provided that p knows the infon x (notice that only the WF level state variable N is updated by this action while the PM level state variable is left unchanged), and

$$\exists p, q.\, \exists m. \left(\begin{array}{l} \langle p, m, q \rangle \in N \ \wedge \ N' = N \ \wedge \\ \forall z_1, z_2.(knows'_0(z_1, z_2) \Leftrightarrow \\ \qquad \left(\begin{array}{ll} (z_1 = q \wedge z_2 = \text{“}p \text{ said } m\text{”}) & \vee \\ knows_0(z_1, z_2) & \end{array} \right)) \end{array} \right)$$

abbreviated with $\exists p, q.\, \exists m.\, m2k(p, m, q)$, which describes the dual situation with respect to the previous one, namely the fact that a principal p can augment its internal knowledge with the content of a certain message if there exists a message in the network with payload m.

The initial state can be formalized by the following state formula, which is of the form (1):

$$
\begin{array}{l}
N = \emptyset \ \wedge \\
\forall z_1, z_2.(knows_0(z_1, z_2) \Leftrightarrow ((z_1 = \mathtt{CA} \wedge z_2 = \text{“}\mathtt{Ed} \text{ ise”}) \ \vee \\
\qquad\qquad\qquad\qquad (z_1 = \mathtt{CA} \wedge z_2 = \text{“}\mathtt{Helen} \text{ ish”}) \ \vee \\
\qquad\qquad\qquad\qquad (z_1 = \mathtt{Helen} \wedge z_2 = \text{“}\mathtt{Ed} \text{ cans”}))),
\end{array}
$$

stating that Ed and Helen are employee and head of the CRO, respectively, and that Ed is granted the permission to store documents in the central database by Helen.

At this point, CA can distribute the two role certificates in its possession to Ed by (symbolically) executing the following two instances of the first transition above: $k2m(\mathtt{CA}, \text{“}\mathtt{Ed} \text{ ise”}, \mathtt{Ed})$ and $k2m(\mathtt{CA}, \text{“}\mathtt{Helen} \text{ ish”}, \mathtt{Ed})$. This is followed by Helen sending Ed the certificate of the permission to store documents in the central database, i.e. $k2m(\mathtt{Helen}, \text{“}\mathtt{Ed} \text{ cans”}, \mathtt{Ed})$. At this point, the following facts can be derived at the PM level (by using rule (IK)):

$$
\begin{array}{r}
\mathsf{knows}(\mathtt{Ed}, \text{“}\mathtt{CA} \text{ said } \mathtt{Ed} \text{ ise”}) \\
\mathsf{knows}(\mathtt{Ed}, \text{“}\mathtt{CA} \text{ said } \mathtt{Helen} \text{ ish”}) \\
\mathsf{knows}(\mathtt{Ed}, \text{“}\mathtt{Helen} \text{ said } \mathtt{Ed} \text{ cans”}).
\end{array}
$$

Finally, Ed can ask CRep to store the accepted (if the case) car registration request in the central database since it is possible to derive that knows(CRep, "Ed cans"), by using the rules in the PM theory as follows. Consider this application of $(P1)$:

$$(P1)\quad \mathsf{knows}(\mathtt{CRep}, \text{“}\mathtt{Ed\ cans}\text{”}) \Leftarrow \mathsf{knows}(\mathtt{CRep}, \text{“}\mathtt{Helen\ ish}\text{”}) \wedge \mathsf{knows}(\mathtt{CRep}, \text{“}\mathtt{Ed\ ise}\text{”}) \wedge \mathsf{knows}(\mathtt{CRep}, \text{“}\mathtt{Helen\ said\ Ed\ cans}\text{”})$$

In order to derive the conclusion, we must derive the three premises. This can be done by applying five times the rule (TA) (twice for each of the two role certificates and once for the certificate concerning the permission to store documents in the central database). In this way, we reconstruct (backward) how knowledge has been propagated between principals and we discharge the hypotheses of these rule instances by using (suitable instances of the) rules $(P1)$—$(P4)$. We omit the details for lack of space; the interested reader is pointed to [10] for a full account. Fortunately, the gory details of these derivations can be mechanized by WSSMT, which is capable of performing all the reasoning described above for the symbolical simulation of the scenario automatically in less than a second (on a standard laptop), by using SPASS as the back-end ATP system.

B. Digital Contract Signing

The Digital Contract Signing (DCS) case study concerns a protocol for secure digital contract signing between *two signers*, which are assumed to have secure access to a trusted third party, called the *Business Portal* (BP) so as to digitally sign a contract. To achieve this goal, each signer communicates the contract's conditions to BP, which creates a digital version of the contract, stores it, and coordinates the two signers so as to obtain their digitally signed copies of the contract, which will be stored for future reference. The BP relies on two trusted services: the *Security Server* SServ and the *Public Key Infrastructure* PKI. The SServ provides operations for creating a unique record in a secure database (only BP can access the SServ and thus modify the database), updating fields of existing records (i.e., to add signatures provided by signers), and sealing the signed contract in the record. The PKI is invoked in order to double check signatures against a Certificate Revocation List (CRL) so to be sure that during the execution of the protocol one signer has not misbehaved (though we have formalized this aspect of the DCS case study with a high level of abstraction). The DCS protocol is successful when both signers provide two correctly signed copies of the same contract and the BP can permanently store the signed copies of the contract.

Instead of formalizing the scenario directly in WSSMT from scratch, we have built a BPEL [2] specification of four processes (one corresponding to a signer, one for BP, one for SServ, and one for PKI) and we have composed their instances for the execution of the DCS protocol. Then, we have run a modified version of the freely available tool BPEL2oWFN [22] on the composed BPEL process so as to obtain a first abstraction of the DCS protocol in the input syntax of WSSMT. The modification of BPEL2oWFN that we did was simple since the tool is capable of computing a Petri net specifying the control-flow of the (composed) service and there is a well-known connection between Petri nets and Vector Addition Systems (VASs), which can be seen as a particular class of guarded assignment systems for the WF level. The details of this relationship are omitted here for lack of space but can be found in, e.g., [23], [24]. The reader may wonder why we took this indirect way to obtain a first abstract specification of DCS: the answer is that the (composed) service was big and thus was very difficult to write from scratch. To have an idea of the dimension of the problem, we show in Fig. 4 the (admittedly almost unreadable) Petri net generated by BPEL2oWFN consisting of 51 places and 26 transitions.

Once obtained an abstraction of the WF level, we have manually added the PM level in order to specify the access control rules for each of the four principals involved in the protocol. Since the source specification was BPEL, we have used the RBAC4BPEL framework proposed in [25], which is an extension of the Role-Based Access Control (RBAC) [4] for BPEL processes. In this framework, the notion of role is used as an indirection between users and permissions to execute certain transitions in the BPEL process (in our abstraction, corresponding to transitions of the Petri net). To realize this schema, two relations are needed: one associating users with roles and one associating roles with permissions (in our case, transitions since we consider only the right to execute an action). By taking the join of the two relations, we can compute the access control relation so as to grant or deny the right to execute a transition to a certain user. Since it is well-known how to symbolically represent relations and the join operation in first-order logic, it was not difficult for us to create a suitable theory for the PM level and augmenting the guards and the updates of the transitions in order to integrate the constraints of the access control rules. Along the same lines, we have added Separation of Duties constraints (e.g., the user signing the contract should not be the same as the one checking the validity of the signature on the contract) and Bound of Duties constraints (e.g., the users signing the contract should be same that have agreed on the conditions of the contract) authorization constraints. Again, further details can be found in [24].

Given the abstract specification of the DCS, we have used WSSMT to perform the symbolic steps corresponding to the typical scenario of execution described in [20]. Since we used a VAS for the WF level, we discharged the resulting proof obligations by invoking the Z3 SMT solver, which provides native support for arithmetics (while the resolution prover SPASS does not). Each proof obligation was discharged in few seconds on a standard laptop and augmented our confidence in the correctness of the specification.

However, the specification we created was quite abstract as it ignored the content of the messages exchanged among the various principals. This was so because we used the tool BPEL2oWFN to generate the specification of the WF level. In

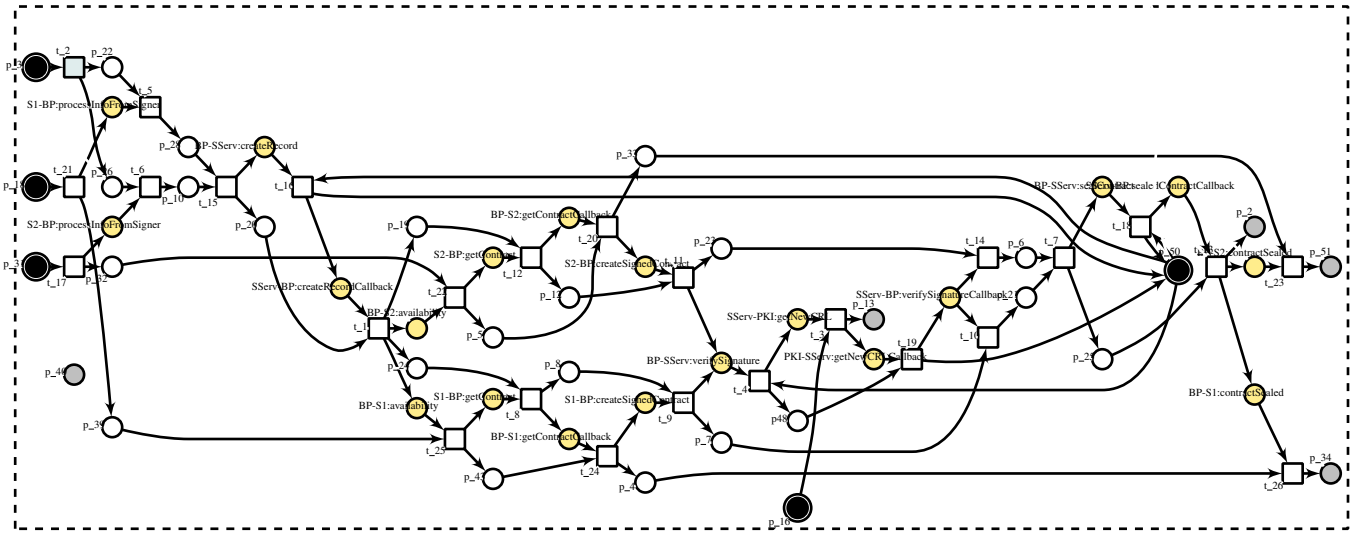

Fig. 4. The Petri net representation of the WF level of the DCS case study

fact, such a tool creates a coarse abstraction of BPEL processes where it is only taken into account if messages are sent and or received. We decided to manually refine the specification by adding FIFO queues containing messages with sender, receiver, and payload. To encode this in first-order theories, several methods are possible (see, e.g., [26]). Once obtained the refined specification, we replayed the symbolic execution corresponding to the typical scenario previously considered by using again Z3 as the back-end ATP system. Again, all the proof obligations were discharged in less that a minute on a standard laptop. Finally, we have also verified some simple inductive invariant properties encoding the fact that the number of tokens in the Petri net remains constant.

ACKNOWLEDGMENT

The work presented in this paper was partially supported by the FP7-ICT-2007-1 Project no. 216471, "AVANTSSAR: Automated Validation of Trust and Security of Service-oriented Architectures" and the "Automated Security Analysis of Identity and Access Management Systems (SIAM)" project funded by Provincia Autonoma di Trento in the context of the "Team 2009 - Incoming" COFUND action of the European Commission (FP7).

REFERENCES

[1] E. Christensen, F. Curbera, G. Meredith, and S. Weerawarana, "Web Services Description Language (WSDL) 1.1," http://www.w3.org/TR/wsdl.

[2] A. Alves et al, "Web Services Business Description Language, Version 2.0, OASIS Standard, April 2007," http://docs.oasis-open-org/wsbpel/2.0/OS/wesbpel-v2.0-OS.pdf.

[3] AVANTSSAR, "The AVANTSSAR Project," http://www.avantssar.eu.

[4] R. Sandhu, E. Coyne, H. Feinstein, and C. Youmann, "Role-Based Access Control Models," *IEEE Computer*, vol. 2, pp. 38–47, 1996.

[5] H. B. Enderton, *A Mathematical Introduction to Logic*. Academic Press, 1972.

[6] R. Sebastiani, "Lazy satisfiability modulo theories," *Journal on Satisfiability, Boolean Modeling and Computation*, vol. 3, pp. 141–224, 2007.

[7] N. Li and J. C. Mitchell, "Understanding SPKI/SDSI using first-order logic," *Int. Journal of Information Security*, vol. 5, pp. 48–64, 2006.

[8] Z. Manna and A. Pnueli, *Temporal Verification of Reactive Systems: Safety*. Springer-Verlag, 1995.

[9] A. R. Bradley and Z. Manna, "Property-Directed Incremental Invariant Generation," *Formal Aspects of Computing*, vol. 20, pp. 379–405, 2008.

[10] M. Barletta, S. Ranise, and L. Viganò, "A declarative two-level framework to specify and verify workflow and authorization policies in service-oriented architectures," *Service-Oriented Computing and Applications*, to appear.

[11] L. Zanetti, *Integrazione di tecniche SMT in un sistema di verifica per applicazioni orientate ai servizi*. Master Thesis (in Italian), Faculty of Mathematical, Physical and Natural Science, University of Verona, 2009.

[12] "SWT: The Standard Widget Toolkit," http://www.eclipse.org/swt.

[13] "JFace," http://wiki.eclipse.org/index.php/JFace.

[14] C. Weidenbach, "Spass input syntax version 1.5," *www.spass-prover.org/download/binaries/spass-input-syntax15.pdf*.

[15] SPASS, "The SPASS web-site," http://www.spass-prover.org.

[16] "Z3," http://research.microsoft.com/projects/z3.

[17] G. Sutcliffe, "The TPTP web-site," http://www.cs.miami.edu/~tptp.

[18] SMT-COMP, "The SMT-COMP web-site," http://www.smtcomp.org.

[19] S. Ranise and C. Tinelli, "The satisfiability modulo theories library (smtlib)," *www.smt-lib.org*, 2009.

[20] AVANTSSAR, "Deliverable 5.1: Problem cases and their trust and security requirements," 2008, available at http://www.avantssar.eu.

[21] Y. Gurevich and I. Neeman, "DKAL: Distributed-Knowledge Authorization Language," in *Proc. of CSF*. IEEE CS Press, 2008, pp. 149–162.

[22] N. Lohmann and C. Gierds and M. Znamirowski, "BPEL2OWFN," available at http://www.gnu.org/software/bpel2owfn.

[23] S. Sankaranarayanan, H. Sipma, and Z. Manna, "Petri Net Analysis Using Invariant Generation," in *Verification: Theory and Practice*. LNCS 2772, Springer-Verlag, 2003, pp. 682–701.

[24] A. Calvi, S. Ranise, and L. Viganò, "Automated Validation of Security-sensitive Web Services specified in BPEL and RBAC," in *Proc. of WoSS 2010*. IEEE CS Press.

[25] F. Paci, E. Bertino, and J. Crampton, "An Access-Control Framework for WS-BPEL," *Int. Journal of Web Services Research*, vol. 5, no. 3, pp. 20–43, 2008.

[26] S. Lahiri, S. Seshia, and R. Bryant., "Modeling and Verification of Out-of-order Microprocessors using UCLID," in *FMCAD'02: Formal Methods in Computer-Aided Design*. LNCS 2517, Springer-Verlag, 2002, pp. 143–159.