Scalable Computing: Practice and Experience
Volume 12, Number 1, pp. 1–20. http://www.scpe.org
ISSN 1895-1767

WORKFLOW AND ACCESS CONTROL RELOADED: A DECLARATIVE SPECIFICATION FRAMEWORK FOR THE AUTOMATED ANALYSIS OF WEB SERVICES

MICHELE BARLETTA*, ALBERTO CALVI†, SILVIO RANISE‡, LUCA VIGANÒ§, AND LUCA ZANETTI¶

Abstract. Web services supporting business and administrative transactions between several parties over the Internet are more and more widespread. Their development involves several security issues ranging from authentication to the management of the access to shared resources according to given business and legal models. The capability of validating designs against fast evolving requirements is of paramount importance for the adaptation of business and administrative models to changing regulations and rapidly evolving market needs. We present formal specification and analysis techniques that allow us to validate the designs of security-sensitive web services specified in the Business Process Execution Language and extensions of the Role-Based Access Control model. We also present a prototype tool, called WSSMT, mechanizing our approach and describe our experience in using it on two industrial case studies, on in the e-business and one in the e-government area.

Key words: Service-oriented architectures, Web services, Business Process Models, Workflow, Access Control, Policy Management, Formal Analysis

AMS subject classifications. 68Q60, 68Q85, 68N30, 68N01, 68M99

1. Introduction. Web services are becoming more and more widespread in many fields like e-commerce, e-health, and e-governance, where services support business and administrative transactions between several parties over the Internet. Their development involves several security issues ranging from authentication to the management of the access to shared resources according to given business and legal models. The capability of validating designs against fast evolving requirements is of paramount importance for the adaptation of business and administrative models to changing regulations and rapidly evolving market needs. So, techniques for the specification and automated analysis of dependable web services to be used in security-sensitive applications are crucial in the development of these systems.

To design practically usable specification and analysis techniques, it is important to have a closer look at the definition of web service. A web 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. Several (executable) specification languages are available: the data part can be described by, e.g., the Web Service Definition Language WSDL [10], the control part by, e.g., the Business Process Execution Language BPEL [1], and the identifiers by Uniform Resource Identifiers.

An additional source of security problems is the fact that many deployed services work 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, for instance, 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 from the WF to the PM and vice versa.

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

*Dipartimento di Informatica, Università di Verona, Italy, michele.barletta@univr.it
†Dipartimento di Informatica, Università di Verona, Italy, alberto.calvi@univr.it
‡FBK-Irst, Trento, Italy, ranise@fbk.eu
§Dipartimento di Informatica, Università di Verona, Italy, luca.vigano@univr.it
¶Dipartimento di Informatica, Sistemistica e Telematica, Università di Genova & FBK-Irst, Trento, Italy, luca.zanetti@unige.it

several respects for the design and maintenance of services, and also for their validation. So, in order to design usable specification and analysis techniques, it is desirable to reflect the two level structure of a service design. In this paper, we explain how this can be achieved by a suitable instantiation of the formal framework in [7]. In particular, we show how the WF level of web services described by BPEL processes can be mapped to Petri nets and how this, in turn, can be translated to a class of transition systems involving integer variables by using well-known results (see, e.g., [20]). For the PM level, we explain how RBAC4BPEL, a variant of the Role-Based Access Control (RBAC [19]) model, can be easily expressed as a certain symbolic transition system by using a fragment of first-order logic (FOL). Then, we explain how to combine the two (symbolic) transition systems to obtain one system, abstractly describing the whole web service. This allows us to introduce a technique for the exploration of the state space that is based on a symbolic execution procedure and exploits the capability of solving logical problems in certain fragments of FOL.

We also discuss how to mechanize the symbolic execution procedure by describing the architecture of a prototype tool, called WSSMT, which we have developed to allow designers of web services 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. 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].

We proceed as follows. In Section 2, we first provide some background, in particular, on BPEL, Petri nets, and RBAC4BPEL, and then introduce a running example. In Section 3, we introduce our formal two-level specification framework, in particular, we discuss two-level transition systems and their symbolic execution. In Section 4, we explain how security-sensitive services specified by (a sub-set of) BPEL and RBAC4BPEL can be specified as two-level transition systems. In Section 5, we present our tool WSSMT and then, in Section 6, we report on the application of WSSMT to the specification and analysis of the two industrial-strength case studies we consider here. In Section 7, we draw some conclusions. Parts of the material included in this paper have appeared in preliminary form in [5, 9].

2. Background and running example. We describe the Business Process Execution Language (BPEL) [1] by using an example (a Purchase Ordering process, PO for short, taken from [16]) and explain how Petri nets can be used as an abstract semantics modelling the control flow of a security-sensitive service ignoring data and security issues. For a complete description of the semantics of BPEL and its relationship to Petri nets, the reader is pointed to, e.g., [25].

2.1. BPEL and Petri nets. In Fig. 2.1, we show a high-level BPEL specification of the WF level of the PO process. The `<process>` element wraps around the entire description of the PO process. The `<sequence>` element states that the activities contained in its scope must be executed sequentially. The `<flow>` element specifies concurrent threads of activities. The `<invoke>` element represents the invocation of an activity that is provided by an available web service. Finally, the `<receive>` element represents the invocation of an activity that is provided by the BPEL process being described. Indeed, BPEL provides a variety of constructs (e.g., to represent variables) that are ignored here for simplicity; the interested reader is pointed to [1]. In the case of the PO process, it is easy to see that the constraints on the execution described above are all satisfied by the nesting of control elements in Fig. 2.1. For example, because of the semantics of `<sequence>`, *crtPO* will be executed first while *apprPay* will be the activity finishing the PO process.

Fig. 2.1 shows also, on the right, a Petri net that can be seen as a (control-flow) abstraction of the BPEL process on the left of the figure. In order to sketch the mapping from BPEL processes to Petri nets, we first recall the basic notions concerning the latter.

A *Petri net* is a triple $\langle P, T, F\rangle$, where P is a finite set of *places*, T is a finite set of *transitions*, and F (*flow relation*) is a set of arcs such that $P \cap T = \emptyset$ and $F \subseteq (P \times T) \cup (T \times P)$. Graphically, the Petri net $\langle P, T, F\rangle$ can be depicted as a directed bipartite graph with two types of nodes, places and transitions, represented by circles and rectangles, respectively; the nodes are connected via directed arcs according to F (where arcs between two nodes of the same type are not allowed). A place p is called an *input* (resp., *output*) place of a transition t iff there exists a directed arc from p to t (resp., from t to p). The set of input (resp., output) places of a transition

```
<process name="PO"/>
  <sequence>
    <receive operation="crtPO" ... > </receive>
    <invoke  operation="apprPO" ... > </invoke>
    <flow>
      <sequence>
        <invoke operation="signGRN" ... > </invoke>
        <invoke operation="ctrSignGRN" ... > </invoke>
      </sequence>
      <invoke operation="ctrPay" ... > </invoke>
    </flow>
    <invoke operation="apprPay" ... > </invoke>
  </sequence>
</process>
```

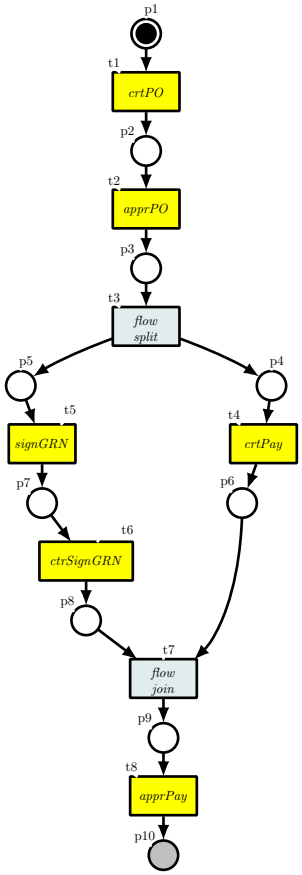

FIG. 2.1. *The PO process: BPEL (left) and corresponding Petri net (right). Legend: yellow transitions specify normal tasks, azure ones specify flow transitions (splits and joins) and grey places represents final places.*

t is denoted by $\bullet t$ (resp., $t\bullet$); $\bullet p$ and $p\bullet$ are defined similarly. A *path* in a Petri net $\langle P, T, F\rangle$ is a finite sequence $e_0, ..., e_n$ of elements from $P \cup T$ such that $e_{i+1} \in e_i\bullet$ for each $i = 0, ..., n-1$; a path $e_0, ..., e_n$ in the net is a *cycle* if no element occurs more than once in it and $e_0 \in e_n\bullet$ for some $n \geq 1$. A Petri net is *acyclic* if none of its paths is a cycle. A *marking* of a Petri net $\langle P, T, F\rangle$ is a mapping from the set P of places to the set of non-negative integers; graphically, it is depicted as a distribution of black dots in the circles of the graph representing the net. A transition t is *enabled* in a marking m iff each of its input places p is such that $m(p) \geq 1$, i.e., p contains at least one token. An enabled transition t in a marking m may *fire* by generating a new marking m', in symbols $m \xrightarrow{t} m'$, where $m'(p) = m(p)$ if $p \notin (\bullet t \cup t\bullet)$, $m'(p) = m(p) - 1$ if $p \in \bullet t$, and $m'(p) = m(p) + 1$ if $p \in t\bullet$, i.e. t consumes one token from each input place of t and produces one token in each of its output places. A marking m is *reachable* from m_0, in symbols $m_0 \rightarrow^* m$, iff there exists a sequence $m_1, ..., m_n$ of markings such that $m_i \xrightarrow{t} m_{i+1}$ for $i = 0, ..., n-1$ and $m_n = m$, for some $n \geq 0$. (For $n = 0$, we have that $m_0 = m$.) Given a Petri net $\langle P, T, F\rangle$ and a marking m, an instance of the *reachable problem* for Petri nets consists of checking whether $m_0 \rightarrow^* m$ or not. A *workflow (WF) net* [28] is a Petri net $\langle P, T, F\rangle$ such that (i) there exist two special places $i, o \in P$ with $\bullet i = \emptyset$ and $o\bullet = \emptyset$; and (ii) for each transition $t \in T$, there exists a path π in the net beginning with i and ending with o in which t occurs.

The idea underlying the Petri net semantics of BPEL is simple. Activities are mapped to transitions (the rectangles in Fig. 2.1) and their execution is modeled by the flow of tokens from input places to output places. When two BPEL operations are enclosed in a `<sequence>` element (e.g., *crtPO* and *apprPO*), two transitions are created (as in Fig. 2.1) with one input place (resp., p_1 and p_2) and one output place each (resp., p_2 and p_3), and the input place of the second is identified with the output place of the first one (p_2). When two BPEL operations are in a `<flow>` element (e.g., *crtPay* and the sequence of operations *signGRN* and *ctrSignGRN*),

$$
\begin{array}{rl}
U & := \{u_1, u_2, u_3, u_4, u_5\} \\
R & := \{\mathit{manager}, \mathit{finAdmin}, \mathit{finClerk}, \mathit{poAdmin}, \mathit{poClerk}\} \\
P & := \{p_1, ..., p_5\} \\
\mathit{ua} & := \{(u_1, \mathit{manager}), (u_2, \mathit{finAdmin}), (u_3, \mathit{finClerk}), (u_4, \mathit{poAdmin}), \quad (u_5, \mathit{poClerk})\} \\
\mathit{pa} & := \{(\mathit{finClerk}, p_4), (\mathit{finAdmin}, p_5), (\mathit{poClerk}, p_3), (\mathit{poAdmin}, p_1)\} \\
\succeq & \text{ least partial order s.t. } \mathit{manager} \succeq \mathit{finAdmin},\ \mathit{manager} \succeq \mathit{poAdmin}, \\
& \qquad\qquad \mathit{finAdmin} \succeq \mathit{finClerk}, \text{ and } \mathit{poAdmin} \succeq \mathit{poClerk}.
\end{array}
$$

FIG. 2.2. *The PM level of the PO process.*

four transitions are created: one to represent the split of the flow, one to represent its synchronization (join), and one for each activity that can be executed concurrently with the appropriate places to connect them (in Fig. 2.1, when a token is in place p_3, the *flow split* transition is enabled and its execution yields one token in place p_4 and one in p_5, which enables both transitions *signGRN* and *crtPay* that can be executed concurrently; the two independent threads of activities get synchronized again by the execution of *flow join*, which is enabled when a token is in p_6 and a token is in p_8). It is not difficult to see that the Petri net of Fig. 2.1 is an acyclic WF net where p_1 is the special input place i, p_{10} is the special output place o, and each transition occurs in a path from p_1 to p_{10}.

2.2. RBAC4BPEL. In [16], Paci, Bertino and Crampton use RBAC4BPEL, an extension of the RBAC framework [19] adapted to work smoothly with BPEL, to specify the PM level of applications like the PO process. The components of RBAC4BPEL are: (i) a set U of users, (ii) a set R of roles, (iii) a set P of permissions, (iv) a role hierarchy $\succeq$ (i.e. a partial-order relation on R), (v) a user-role assignment relation ua, (vi) a role-permission assignment relation pa, and (vii) a set A of activities and a class of authorization constraints (such as separation-of-duty) to prevent some user to acquire permissions in certain executions of the application (see below for details). Note that components (i)–(vi) are standard in RBAC whereas (vii) has been added to obtain a better integration between the PM and the WF levels.

First, we describe components (i)–(vi) and some related notions. A user $u \in U$ is assigned a role $r \in R$ if $(u, r) \in ua$ and permissions are associated with roles when $(p, r) \in pa$. In RBAC4BPEL, a user $u \in U$ has a permission p if there exists a role $r \in R$ such that $(u, r) \in ua$ and $(p, r) \in pa$. (We will see that each permission is associated to a right on a certain activity in A of a BPEL process, e.g., its execution.) The role hierarchy $\succeq \subseteq R \times R$ is assumed to be a partial order (i.e., a reflexive, antisymmetric, and transitive relation) reflecting the rights associated to roles. More precisely, a user u is an *explicit* member of role $r \in R$ if $(u, r) \in ua$ and it is an *implicit* member of role $r \in R$ if there exists a role $r' \in R$ such that $(r', r) \in \succeq$ (abbreviated as $r' \succeq r$), $r' \neq r$, and $(u, r') \in ua$. Thus, $\succeq$ induces a permission inheritance relation as follows: a user $u \in U$ *can get* permission p if there exists a role $r \in R$ such that u is a member (either implicit or explicit) of r and $(p, r) \in pa$. For simplicity, we abstract away the definition of a role in terms of a set of attributes as done in [16].

Fig. 2.2 shows the sets U, R, P and the relations ua, pa, and $\succeq$ for the PM level of the PO process. Although $(\mathit{manager}, p_i) \notin pa$ for any $i = 1, ..., 5$, we have that user u_1, which is explicitly assigned to role *manager* in ua, can get any permission p_i for $i = 2, ..., 5$ as $\mathit{manager} \succeq r$ for any role $r \in R \setminus \{\mathit{manager}\}$, hence u_1 can be implicitly assigned to each role and then get the permission p_i.

In RBAC4BPEL, each permission in P is associated with the right to handle a certain transition of T, uniquely identified by a label in A, for a Petri net $\langle P, T, F\rangle$. In many cases, this is particularly simple since only the right to execute a transition is considered as it is the case in the services considered in this paper. We bind permissions p_i to different tasks as follows: p_1 is the permission for executing apprPO, p_2 for *signGRN*, p_3 for *ctrSignGRN*, p_4 for *crtPay*, and p_5 for $\mathit{apprPay}$. We are now in the position to describe component (vii) of RBAC4BPEL. Note that there are no permissions associated to *flow split* and *flow join* as these are performed by the BPEL engine and thus no particular authorization restriction must be enforced.

A *role* (resp., *user*) *authorization constraint* is a tuple $\langle D, (t_1, t_2), \rho\rangle$ if $D \subseteq R$ (resp., $D \subseteq U$) is the domain of the constraint, $\rho \subseteq R \times R$ (resp., $\rho \subseteq U \times U$), and t_1, t_2 are in A. An authorization constraint $\langle D, (t_1, t_2), \rho\rangle$ is *satisfied* if $(x, y) \in \rho$ when $x, y \in D$, x performs t_1, and y performs t_2. In other words, authorization constraints place further restrictions (besides those of the standard RBAC components) on the roles or users who can perform certain actions once others have been already executed by users belonging to certain roles.

Constraints of this kind allow one to specify *separation-of-duty (SoD)* by $\langle D, (t_1, t_2), \neq \rangle$, *binding-of-duty (BoD)* by $\langle D, (t_1, t_2), = \rangle$, or any other restrictions that can be specified by a binary relation over roles or users.

For the PO process, (vii) of RBAC4BPEL is instantiated as:

$$\langle U, (apprPO, signGRN), \neq \rangle,\ \langle U, (apprPO, ctrSignGRN), \neq \rangle,$$
$$\langle U, (signGRN, ctrSignGRN), \neq \rangle,\ \langle R, (crtPay, apprPay), \prec \rangle,$$

where $\prec := \{(r_1, r_2) \mid r_1, r_2 \in R,\ r_2 \succeq r_1,\ r_1 \neq r_2\}$ (recall that the sets U and R are defined in Fig. 2.2).

3. A formal two-level specification framework. Let us now consider the structure of the PO process introduced in Section 2. We can regard it as structured in two levels: the *WF level* dealing with the control of the flow (and the manipulation of data) and the *PM level* describing access control rules (and trust relationships). 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 relation of RBAC system, while the latter describes the possible executions of the system, e.g., how a certain integer variable storing the number of clients being served for the WF level or how a tuple is added to or deleted from a relation in a database for the PM level.

All the four components of our framework (static/dynamic parts of the WF/PM levels) are symbolically represented by formulae of many-sorted FOL with equality (see, e.g., [12] for definitions of the basic notions that we use in the following of the paper), 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. The work in [6] has introduced 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, which is described in Section 5.

In this section, we briefly recall the main notions of the framework in [6] for specifying security-sensitive services composed of the WF and PM levels. Formally, these two levels are specified by a particular class of symbolic guarded assignment systems, called *two-level transition systems*, where first-order formulae are used to represent sets of states (i.e. the static part) and the actions (i.e. the dynamic part) of the system. More precisely, for this class of transition systems, the state variables are updated by applying a function to the actual values of the variables provided that a guard (expressed as a condition again on the values of the state variables) is satisfied. The state space of such transition systems can be explored by using a symbolic execution based on satisfiability solving of a class of FOL formulae. In the rest of this section, we recall the notion of two-level transition systems (Section 3.1) and then describe the symbolic execution technique (Section 3.2).

3.1. Two-level transition systems. A *two-level transition system Tr* is a tuple

$$\langle \underline{x}, \underline{p}, In(\underline{x}, \underline{p}), \{\tau_i(\underline{x}, \underline{p}, \underline{x}', \underline{p}') \mid i = 1, ..., n\} \rangle,$$

where $\underline{x}$ is a tuple of *WF state* variables; $\underline{p}$ is a tuple of *PM state* variables; the *initial condition* $In(\underline{x}, \underline{p})$ is a FOL formula whose only free variables are in $\underline{x}$ and where PM state variables in $\underline{p}$ may occur as predicate symbols; and for $i = 1, ..., n$ and $n \geq 1$ the *transition* $\tau_i(\underline{x}, \underline{p}, \underline{x}', \underline{p}')$ is a FOL formula whose only free variables are in $\underline{x}, \underline{x}'$ and where PM state variables in $\underline{p}, \underline{p}'$ may occur as predicate symbols (as it is customary, unprimed variables in τ_i refer to the values of the state before the execution of the transition while those primed refer to the values of the state afterward).

We assume there exists a so-called first-order *underlying structure* $\langle D, I \rangle$ *of the transition system Tr*, where D is the domain of values and I is the mapping from the signature to functions and relations over D, and in which the state variables and the symbols of the signature used to write the formulae In and τ_i for $i = 1, ..., n$ are mapped. A *state* of *Tr* is a pair $v := (v_{\underline{x}}, v_{\underline{p}})$ of mappings: $v_{\underline{x}}$ from the WF state variables to D and $v_{\underline{p}}$ from the PM state variables to relations over D.

A *run* of *Tr* is a (possibly infinite) sequence of states $v^0, v^1, ..., v^n, ...$ such that (i) v^0 satisfies In, in symbols $v^0 \models In$, and (ii) for every pair v^i, v^{i+1} in the sequence, there exists $j \in \{1, ..., n\}$ such that v^i, v^{i+1} satisfies τ_j, in symbols $v^i, v^{i+1} \models \tau_j$, where the domain of v^i is $\underline{x}, \underline{p}$ and that of v^{i+1} is $\underline{x}', \underline{p}'$. Given a formula $G(\underline{x}, \underline{p})$,

called the *goal*, an instance of the *goal reachability problem* for Tr consists of answering the following question: does there exist a natural number $\ell \geq 0$ such that the formula

$$In(\underline{x}_0, \underline{p}_0) \wedge \bigwedge_{i=o}^{\ell-1} \tau(\underline{x}_i, \underline{p}_i, \underline{x}_{i+1}, \underline{p}_{i+1}) \wedge G(\underline{x}_\ell, \underline{p}_\ell) \tag{3.1}$$

is satisfiable in the underlying structure of Tr, where $\underline{x}_i, \underline{p}_i$ are renamed copies of the state variables in $\underline{x}, \underline{p}$? (When $\ell = 0$, (3.1) is simply $In(\underline{x}_0, \underline{p}_0) \wedge G(\underline{x}_0, \underline{p}_0)$). The interest of the goal reachability problem lies in the fact that many verification problems for two-level transition systems, such as invariant checking, can be reduced to it.

3.2. Symbolic execution of two-level transition systems. If we were able to check automatically the satisfiability of (3.1), an idea to solve the goal reachability problem for two-level transition systems would be to generate instances of (3.1) for increasing values of ℓ. However, this would only give us a semi-decision procedure for the reachability problem. In fact, this method terminates only when the goal is reachable from the initial state, i.e., when the instance of (3.1) for a certain value of ℓ is unsatisfiable in the underlying structure of the transition system Tr. But, when the goal is not reachable, the check will never detect the unsatisfiability and we will be bound to generating an infinite sequence of instances of (3.1) for increasing values of ℓ. That is, the decidability of the satisfiability of (3.1) in the underlying structure of Tr is only a necessary condition for ensuring the decidability of the goal reachability problem.

We can formalize this method as follows. The *post-image* of a formula $K(\underline{x}, \underline{p})$ with respect to a transition τ_i is

$$Post(K, \tau_i) := \exists \underline{x}', \underline{p}'.(K(\underline{x}', \underline{p}') \wedge \tau_i(\underline{x}', \underline{p}', \underline{x}, \underline{p})).$$

For the class of transition systems that we consider below, we are always able to find FOL formulae that are equivalent to $Post(K, \tau_i)$. Thus, the use of the second-order quantifier over the predicate symbols in $\underline{p}'$ should not worry the reader (see Section 4.2 for details).

Now, define the following sequence of formulae by recursion: $FR^0(K, \tau) := K$ and $FR^{i+1}(K, \tau) := Post^i(FR^i, \tau) \vee FR^i(K, \tau)$, for $i \geq 0$ and $\tau := \bigvee_{k=1}^n \tau_i$. The formula $FR^\ell(K, In)$ describes the set of states of the transition system Tr that are *forward reachable in* $\ell \geq 0$ *steps*.

A *fix-point* is the least value of ℓ such that $FR^{\ell+1}(\tau, In) \Rightarrow FR^\ell(\tau, In)$ is true in the structure underlying Tr. Note also that $FR^\ell(\tau, In) \Rightarrow FR^{\ell+1}(\tau, In)$ by construction and hence if $FR^{\ell+1}(\tau, In) \Rightarrow FR^\ell(\tau, In)$ is valid, then also $FR^\ell(\tau, In) \Leftrightarrow FR^{\ell+1}(\tau, In)$ is so and $FR^\ell(\tau, In) \Leftrightarrow FR^{\ell'}(\tau, In)$ for each $\ell' \geq \ell$.

Using the sequence of formulae $FR^0(\tau, In), FR^1(\tau, In), ...$ it is possible to check if the goal property G will be reached by checking whether $FR^\ell(\tau, In) \wedge G$ is satisfiable in the structure underlying Tr for some $\ell \geq 0$. In case of satisfiability, we say that G is *reachable*. Otherwise, if $FR^\ell(\tau, In)$ is a fix-point, the unsatisfiability of $FR^\ell(\tau, In) \wedge G$ implies that G is *unreachable*.

Finally, if $FR^\ell(\tau, In)$ is not a fix-point and $FR^\ell(\tau, In) \wedge G$ is unsatisfiable, then we must increase the value of ℓ by 1 so as to compute the set of forward reachable states in $\ell + 1$ steps and perform the reachability checks again. Unfortunately, also this process is not guaranteed to terminate for arbitrary two-level transition systems. Fortunately, we are able to characterize a set of transition systems, corresponding to a relevant class of applications specified in BPEL and RBAC4BPEL, for which we can pre-compute an upper bound on ℓ; this paves the way to solving automatically the goal reachability problem for these systems.

To this end, we consider three sufficient conditions to automate the solution of the goal reachability problem. First, the class $\mathcal{C}$ of formulae used to describe sets of states must be closed under post-image computation. Second, the satisfiability (in the structure underlying the transition system) of $\mathcal{C}$ must be decidable. Third, it must be possible to pre-compute a bound on the length of the sequence $FR^0, FR^1, ..., FR^\ell$ of formulae. Below, we show that these conditions are satisfied by a class of two-level transition systems to which applications specified in BPEL and RBAC4BPEL can be mapped. For ease of exposition, we first consider the WF and PM levels in isolation and then show how the results for each level can be modularly lifted when considering the two levels together. Before doing this, we introduce the notion of symbolic execution tree. The purpose of this is two-fold. First, it is crucial for the technical development of our decidability result. Second, it is the starting point for the implementation of our techniques as discussed in Section 5.

The *symbolic execution tree of the two-level transition system* Tr is a labeled tree defined as follows: (i) the root node is labeled by the formula In, (ii) a node n labeled by the formula K has $d \leq n$ sons $n_1, ..., n_d$ labeled

by the formulae $Post(\tau_1, K), ..., Post(\tau_d, K)$ such that $Post(\tau_j, K)$ is satisfiable in the model underlying Tr and the edge from n to n_j is labeled by τ_j for $j = 1, ..., d$, (iii) a node n labeled by K has no son, in which case n is a *final node*, if $Post(\tau_j, K)$ is unsatisfiable in the underlying model of the VAS, for each $j = 1, ..., n$. A symbolic execution tree is 0-*complete* if it consists of the root node labeled by the formula In, it is $(d+1)$-*complete* for $d \geq 0$ if its depth is $d+1$ and for each node n labeled by a formula K_n at depth d, if $Post(\tau_j, K_n)$ is satisfiable, then there exists a node n' at depth $d+1$ labeled by $Post(\tau_j, K_n)$. In other words, a symbolic execution tree is d-complete when all non-empty sets of forward states reachable in one step represented by formulae labeling nodes at depth $d-1$ have been generated. It is easy to see that the formula $FR^{\ell}(K, In)$, describing the set of states of the transition system Tr forward reachable in $\ell \geq 0$ steps, is equivalent to the disjunction of the formulae labeling the nodes of an ℓ-complete symbolic execution tree. This will be proved for the classes of two-level transition systems that we consider below.

4. Mapping BPEL and RBAC4BPEL to two-level transition systems. We now explain how security-sensitive services specified by (a sub-set of) BPEL and RBAC4BPEL (such as the PO process described in Section 2) can be specified as two-level transition systems. To simplify the task of the specifier as well as the technical development, we first describe how WF nets can be seen as a certain class of transition systems (Section 4.1), then we show how an RBAC4BPEL system can be mapped to a certain transition system (Section 4.2), and finally we explain how these specifications can be combined to obtain a complete specification (Section 4.3). We illustrate the notions by using the PO process as the running example. Although we do not show it for lack of space, transforming WF nets and RBAC4BPEL can be made automatic. For more details about this point, see also Section 6.2 where we briefly discuss how we have modified an available tool for generating Petri nets from BPEL files to derive the transition systems described in the following subsection.

4.1. WF nets and terminating forward reachability. We consider *Vector Addition System (VAS)*, a particular class $\langle \underline{x}, In(\underline{x}), \{\tau_i(\underline{x}, \underline{x}') \mid i = 1, ..., n\}\rangle$, of two-level transition systems such that (i) $\underline{p} = \emptyset$; (ii) their underlying structure is that of integers; (iii) each WF state variable in $\underline{x} = x_1, ..., x_m$ ranges over the set of non-negative integers; (iv) the initial condition $In(\underline{x})$ is a formula of the form $x_1 \bowtie c_1 \wedge \cdots \wedge x_m \bowtie c_m$, where c_j is a natural number for $j = 1, ..., m$ and $\bowtie \in \{=, \neq, >, \geq\}$; and (v) each transition τ_i, for $i = 1, ..., n$, is a formula of the form

$$\bigwedge_{i \in P} x_i \geq 0 \wedge \bigwedge_{j \in U^+} x'_j = x_j + 1 \wedge \bigwedge_{k \in U^-} x'_k = x_k - 1 \wedge \bigwedge_{l \in U^=} x'_l = x_l \,,$$

where $P, U^+, U^-, U^=$ are subsets of $\{1, ..., n\}$ such that $U^+, U^-, U^=$ form a partition of $\{1, ..., n\}$.

It is well-known that Petri nets and VASs are equivalent in the sense that analysis problems for the former can be transformed to problems of the latter whose solutions can be mapped back to solutions for the original problem and vice versa (see, e.g., [20]). We briefly describe the correspondence by considering the Petri net in Fig. 2.1. We associate an integer variable x_i to each place p_i for $i = 1, ..., 10$ whose value will be the number of tokens in the place. The state is given by the value of the integer variables representing the marking of the net, i.e., a mapping from the set of places to non-negative integers. Formulae can be used to represent sets of states (or, equivalently, of markings). So, for example, the formula $x_1 = 1 \wedge \bigwedge_{i=2}^{10} x_i = 0$ represents the marking where one token is in place p_1 and all the other places are empty (which is the one depicted in Fig. 2.1 where the token is represented by a solid circle inside that represents the place p_1 while all the other places do not contain any solid circle). The transition *crtPO* is represented by the formula

$$x_1 \geq 1 \wedge x'_1 = x_1 - 1 \wedge x'_2 = x_2 + 1 \wedge \bigwedge_{i=3}^{10} x'_i = x_i$$

saying that it is enabled when there is at least one token in p_1 ($x_1 \geq 1$) and the result of its execution is that a token is consumed at place p_1 ($x'_1 = x_1 - 1$), the tokens in p_2 are incremented by one ($x'_2 = x_2 + 1$), while the tokens in all the other places are unaffected ($x'_i = x_i$ for $i = 3, ..., 10$). The other transitions of the Petri net in Fig. 2.1 are translated in a similar way. In general, it is always possible to associate a state of a VAS to a marking of a Petri net and vice versa. This implies that solving the reachability problem for a VAS is equivalent to solving the reachability problem of the associated Petri net.

Now, we show that the three sufficient conditions (see Section 3.2) to mechanize the solution of the goal reachability problem are satisfied by VASs when using forward reachability. First, the class of formulae is closed under post-image computation:

FACT 1. *$Post(K, \tau_i)$ is equivalent to $K[x_j - 1, x_k + 1, x_l] \wedge \bigwedge_{i \in P} x_i \geq 0$, where $K[x_j - 1, x_k + 1, x_l]$ denotes the formula obtained by replacing x'_j with $x_j - 1$ for $j \in U^+$, x'_k with $x_k - 1$ for $k \in U^-$, and x'_l with x_l for $j \in U^=$.* ■

As a corollary, it is immediate to derive that if K is a formula of Linear Arithmetic (LA) [8] — roughly, a formula where multiplication between variables is forbidden — then also $Post(K, \tau_i)$ is equivalent to an effectively computable formula of LA. Second, the satisfiability of the class of formulae of LA is decidable by well-known results [8]. Third, it is possible to pre-compute a bound on the length of the sequence $FR^0, FR^1, ..., FR^\ell$ of formulae. Using the notion of symbolic execution tree introduced above, once specialized to VASs, we can then prove:

LEMMA 4.1. *Let $PN := \langle P, T, F \rangle$ be an acyclic workflow net and Π be the set of all its paths. Then, the set of forward reachable states of $\langle \underline{x}, In(\underline{x}), \{\tau_i(\underline{x}, \underline{x}') \mid i = 1, ..., n\} \rangle$, the VAS associated to PN, is identified by the formula $FR^\ell(\tau, In)$ for $\ell = max_{\pi \in \Pi}\{len(\pi|_T)\}$, where $\pi|_T$ is the sequence obtained from π by forgetting each of its elements in P and $len(\pi|_T)$ is the length of the sequence $\pi|_T$.* ■

This result is proved in [9] by using the notion of symbolic execution tree introduced above, specialized to VASs.

4.2. RBAC4BPEL and terminating forward reachability. Preliminarily, let $Enum(\{v_1, ..., v_n\}, S)$ be the following set of FOL formulae axiomatizing the enumerated datatype with values $v_1, ..., v_n$ for a given $n \geq 1$ over a type S: $v_i \neq v_j$ for each pair (i, j) of numbers in $\{1, ..., n\}$ such that $i \neq j$ and $\forall x. (x = v_1 \vee \cdots \vee x = v_n)$, where x is a variable of type S. The formulae in $Enum(\{v_1, ..., v_n\}, S)$ fix the number of elements of any interpretation to be $v_1, ..., v_n$; it is easy to see that the class of structures satisfying these formulae are closed under isomorphism. We consider RBAC4BPEL, a particular class $\langle \underline{p}, In(\underline{p}), \{\tau_i(\underline{p}, \underline{p}') \mid i = 1, ..., n\} \rangle$ of two-level transition systems such that (i) $\underline{x} = \emptyset$; (ii) the initial condition $In(\underline{p})$ is of the form $\forall \underline{w}. \varphi(\underline{w})$, where φ is a quantifier-free formula where at most the variables in $\underline{w}$ may occur free; and (iii) the underlying structure is one in the (isomorphic) class of many-sorted structures axiomatized by the following sentences:

$$
\begin{array}{l}
Enum(U, User)\,, \\
Enum(R, Role)\,, \\
Enum(P, Permission)\,, \\
Enum(A, Action)\,, \\
\forall u, r.\, (ua(u, r) \Leftrightarrow \bigvee_{c_u \in U_{ua}, c_r \in R_{ua}} (u = c_u \wedge r = c_r))\,, \\
\forall r, p.\, (pa(r, p) \Leftrightarrow \bigvee_{c_r \in R_{pa}, c_p \in P_{pa}} (r = c_r \wedge p = c_p))\,, \\
c_r \succeq c'_r \text{ for } c_r, c'_r \in R\,, \\
\forall r.\, (r \succeq r)\,, \\
\forall r_1, r_2, r_3.\, (r_1 \succeq r_2 \wedge r_2 \succeq r_3 \Rightarrow r_1 \succeq r_3)\,, \\
\forall r_1, r_2.\, (r_1 \succeq r_2 \wedge r_2 \succeq r_1 \Rightarrow r_1 = r_2)\,,
\end{array}
$$

where U, R and P are finite sets of constants denoting users, roles, and permissions, respectively, A is a finite set of actions, u is a variable of type *User*, r and its subscripted versions are variables of type *Role*, p is a variable of type *Permission*, $U_{ua} \subseteq U$, $R_{ua} \subseteq R$, $R_{pa} \subseteq R$, and $P_{pa} \subseteq P$; (d) $\underline{p} = xcd$ is a predicate symbol of type $User \times Action$ abbreviating *executed*; and (e) each τ_i is of the form

$$\exists \underline{u}.\, (\xi(\underline{u}, xcd) \wedge \forall x, y.(xcd'(x, y) \Leftrightarrow ((x = u_j \wedge y = p) \vee xcd(x, y)))),$$

where $\underline{u}$ is a tuple of existentially quantified variables of type *User*, u_j is the variable at position j in $\underline{u}$, and $\xi(\underline{u}, xcd)$ is a quantifier-free formula (called the *guard* of the transition) where no function symbol of arity greater than 0 may occur (the part of τ_i specifying xcd' is called the *update*).

To explain how a RBAC4BPEL system can be specified by the formulae above let us consider again the example described in Section 2. To constrain the sets of users, roles, and permissions to contain exactly the elements specified in Fig. 2.2, it is sufficient to use the following sets of formulae:

$$
\begin{array}{l}
Enum(\{u_1, u_2, u_3, u_4\}, User)\,, \\
Enum(\{manager, finAdmin, finClerk, poAdmin, poClerk\}, Role)\,, \\
Enum(\{p_1, p_2, p_3, p_4, p_5\}, Permission)\,.
\end{array}
$$

It is also easy to see that the formulae

$$\forall u, r.\,(ua(u,r) \Leftrightarrow \left(\begin{array}{l}(u = u_1 \wedge r = \mathit{manager}) \vee (u = u_2 \wedge r = \mathit{finAdmin}) \vee \\ (u = u_3 \wedge r = \mathit{finClerk}) \vee (u = u_4 \wedge r = \mathit{poAdmin}) \vee \\ (u = u_5 \wedge r = \mathit{poClerk})\end{array}\right)$$

and

$$\forall r, p.\,(pa(r,p) \Leftrightarrow \left(\begin{array}{l}(r = \mathit{finClerk} \wedge p = p_4) \vee (r = \mathit{finAdmin} \wedge p = p_5) \vee \\ (r = \mathit{poClerk} \wedge p = p_3) \vee (r = \mathit{poAdmin} \wedge p = p_1)\end{array}\right)$$

are satisfied by the interpretations of *ua* and *pa* in Fig. 2.2 and that $\mathit{manager} \succeq \mathit{finAdmin}$, $\mathit{manager} \succeq \mathit{poAdmin}$, $\mathit{finAdmin} \succeq \mathit{finClerk}$, and $\mathit{poAdmin} \succeq \mathit{poClerk}$ with the three formulae above for reflexivity, transitivity and antisymmetry make the interpretation of $\succeq$ the partial order considered in Fig. 2.2. The state variable *xcd* allows us to formalize component (vii) of the RBAC4BPEL system about the authorization constraints. The idea is to use *xcd* to store the pair user u and action a when u has performed a so that the authorization constraints can be formally expressed by a transition involving suitable pre-conditions on these variables. We illustrate the details on the first authorization constraint considered in Section 4.2, i.e., $\langle U, (\mathit{apprPO}, \mathit{signGRN}), \neq\rangle$. The corresponding transition can be formalized as follows:

$$\exists x_1, x_2.\,(xcd(x_1, \mathit{apprPO}) \wedge x_1 \neq x_2 \wedge \forall x, y.\,(xcd'(x,y) \Leftrightarrow ((x = x_2 \wedge y = \mathit{signGRN}) \vee xcd(x,y)))\,.$$

The guard of the transition prescribes that the user x_2 is not the same user x_1 that has previously performed the action *apprPO* and the update stores in *xcd* the new pair $(x_2, \mathit{signGRN})$. The following two constraints at the end of Section 4.2, namely $\langle U, (\mathit{apprPO}, \mathit{ctrSignGRN}), \neq\rangle$ and $\langle U, (\mathit{signGRN}, \mathit{ctrSignGRN}), \neq\rangle$, are formalized in a similar way. The encoding of the last constraint, i.e., $\langle R, (\mathit{crtPay}, \mathit{apprPay}), \prec\rangle$, is more complex and requires also the use of the user-role relation *ua* to represent the constraint on the role hierarchy:

$$\exists x_1, x_2, r_1, r_2.\,(xcd(x_1, \mathit{crtPay}) \wedge ua(x_1, r_1) \wedge ua(x_2, r_2) \wedge r_2 \succeq r_1 \wedge r_1 \neq r_2 \wedge \\ \forall x, y.\,(xcd'(x,y) \Leftrightarrow ((x = x_2 \wedge y = \mathit{apprPay}) \vee xcd(x,y)))\,.$$

The reader should now be convinced that every RBAC4BPEL specification can be translated into a RBAC4BPEL system.

We show that the three sufficient conditions to mechanize the solution of the goal reachability problem (see Section 3.2) are satisfied by RBAC4BPEL systems when using forward reachability. First, the class of formulae is closed under post-image computation.

Fact 2. *$\mathit{Post}(K, \tau_i)$ is equivalent to*

$$(\exists \underline{u}.(K(xcd) \wedge xcd(u_j, t) \wedge \xi(\underline{u}, xcd))) \vee (\exists \underline{u}.(K[\lambda x, y.(\neg(x = u_j \wedge y = t) \wedge xcd(x,y))] \wedge \\ \xi[\underline{u}, \lambda x, y.(\neg(x = u_j \wedge y = t) \wedge xcd(x,y))]))\,,$$

where $K[\lambda x, y.(\neg(x = u_j \wedge y = t) \wedge xcd(x,y))]$ is the formula obtained from K by substituting each occurrence of xcd' with the λ-expression in the square brackets and then performing the β-reduction and similarly for $\xi[\underline{u}, \lambda x, y.(\neg(x = u_j \wedge y = t) \wedge xcd(x,y))]$. ■

As anticipated above when introducing the definition of post-image for two-level transition systems, we can eliminate the second-order quantifier over the predicate symbol *xcd*. Now, recall that a formula is in the *Bernays-Schönfinkel-Ramsey (BSR)* class if it has the form $\exists \underline{z} \forall \underline{w}.\,\phi(\underline{z}, \underline{w})$, for ϕ a quantifier-free formula and $\underline{z} \cap \underline{w} = \emptyset$ (see, e.g., [17]). As a corollary of Fact 2, it is immediate to see that if K is a BSR formula, then also $\mathit{Post}(\tau_i, K)$ is equivalent, by trivial logical manipulations, to a formula in the BSR class. Since $\mathit{In}(xcd)$ is a formula in the BSR class, then all the formulae in the sequence $FR^0, FR^1, ...$ will also be BSR formulae. The second requirement is also fulfilled since the satisfiability of the BSR class is well-known to be decidable [17] and the formulae used to axiomatize the structures underlying the RBAC4BPEL transition systems are also in BSR. Third, it is possible to pre-compute a bound on the length of the sequence $FR^0, FR^1, ..., FR^\ell$ of formulae, although the existential prefix grows after each computation of the post-image when considering the formulae describing the set of forward reachable states. This is so because we consider only a finite and known set of

users so that the length of the existentially quantified prefix is bounded by $n_u^k \times n$, where k is the maximal length of the existential prefixes of the transitions in the RBAC4BPEL system, n_u is the number of users, and n is the number of transitions.

PROPERTY 1. *Let $\langle \underline{p}, In(\underline{p}), \{\tau_i(\underline{p}, \underline{p}') \mid i = 1, ..., n\}\rangle$ be a RBAC4BPEL system, k be the maximal length of the existential prefixes of $\tau_1, ..., \tau_n$, and n_u be the cardinality of the set of users. Then, its symbolic execution tree is ℓ-complete for every ℓ such that $\ell \geq n_u^k \times n$.* ■

The key idea of the proof is the observation that *xcd* is interpreted as a subset of the Cartesian product between the set of users and the set of actions whose cardinalities are bounded.

4.3. Combining VASs and RBAC4BPEL systems. We are now ready to fully specify applications that feature both the WF and the PM level. To do this, we consider *VAS+RBAC4BPEL* systems, two-level transition systems of the form

$$\langle \underline{x}, \underline{p}, In_V(\underline{x}) \wedge In_R(\underline{p}), \{\tau_i^V(\underline{x}, \underline{x}') \wedge \tau_i^R(\underline{p}, \underline{p}') \mid i = 1, ..., n\}\rangle,$$

where $\underline{x} = x_1, ..., x_n$ for some $n \geq 1$; $\underline{p} = xcd$, $In_V(\underline{x})$ is the initial condition of a VAS; $In_R(\underline{p})$ is the initial condition of a RBAC4BPEL system; $\tau_i^V(\underline{x}, \underline{x}')$ is a transition of a VAS; and $\tau_i^R(\underline{p}, \underline{p}')$ is a transition formula of a RBAC4BPEL system for $i = 1, ..., n$. Note that for some transition, the guard ξ of $\tau_i^R(\underline{p}, \underline{p}')$ may be tautological since the operation involves no access-control policy restriction (e.g., the *flow split* and *flow join* of the Petri net in Fig. 2.1). It is natural to associate a VAS and an RBAC4BPEL system to a VAS+RBAC4BPEL system by projection, i.e., the associated VAS is $\langle \underline{x}, In_V(\underline{x}), \{\tau_i^V(\underline{x}, \underline{x}') \mid i = 1, ..., n\}\rangle$ and the associated RBAC4BPEL system is $\langle \underline{p}, In_R(\underline{p}), \{\tau_i^R(\underline{p}, \underline{p}') \mid i = 1, ..., n\}\rangle$. The structure underlying the VAS+RBAC4BPEL system is such that its reduct to the signature of the VAS is identical to the structure underlying the associated VAS, and its reduct to the signature of the RBAC4BPEL system is identical to the structure underlying the associated RBAC4BPEL system.

We now show how it is possible to modularly compute the post-image of a VAS+RBAC4BPEL system by combining the post-images of the associated VAS and RBAC4BPEL system.

FACT 3. *Let $K(\underline{x}, xcd) := K_V(\underline{x}) \wedge K_R(xcd)$. Then, $Post(K, \tau_i)$ is equivalent to*

$$K_V[x_j + 1, x_k - 1, x_l] \wedge \bigwedge_{i \in P} x_i \geq 0 \wedge ((\exists \underline{u}.(K_R(xcd) \wedge xcd(u_j, t) \wedge \xi(\underline{u}, xcd))) \vee (\exists \underline{u}.(K_R[\lambda x, y.(\neg(x = u_j \wedge y = t) \wedge xcd(x, y))] \wedge \xi[\underline{u}, \lambda x, y.(\neg(x = u_j \wedge y = t) \wedge xcd(x, y))]))),$$

where the same notational conventions of Facts 1 and 2 have been adopted. In other words, the post-image of a VAS+RBAC4BPEL system is obtained as the conjunction of the post-images of the associated VAS, denoted with $Post_V(K, \tau_i) := Post(K_V, \tau_i^V)$, and the associated RBAC4BPEL system, denoted with $Post_R(K, \tau_i) := Post(K_R, \tau_i^R)$. Thus, we abbreviate the above formula as $Post_V(K, \tau_i) \wedge Post_R(K, \tau_i)$. ■

The proof of this fact is obtained by simple manipulations minimizing the scope of applicability of $\exists \underline{x}$ and $\exists xcd$, respectively, and then realizing that the proofs of Facts 1 and 2 can be re-used verbatim. Because of the modularity of post-image computation, it is possible to modularly define the set of forward reachable states and the symbolic execution trees for VAS+RBAC4BPEL systems in the obvious way. By modularity, we can easily show the following property.

PROPERTY 2. *Let $PN := \langle P, T, F\rangle$ be a an acyclic WF net, $\langle \underline{x}, In_V(\underline{x}), \{\tau_i^V(\underline{x}, \underline{x}') \mid i = 1, ..., n\}\rangle$ be its associated VAS, and $\langle \underline{p}, In_R(\underline{p}), \{\tau_i^R(\underline{p}, \underline{p}') \mid i = 1, ..., n\}\rangle$ be the RBAC4BPEL system with n_u users and k be the maximal length of the existential prefixes of $\tau_1^R, ..., \tau_n^R$. Then, the symbolic reachability tree of the VAS+RBAC4BPEL system whose associated VAS and RBAC systems are those specified above is ℓ-complete for every $\ell \geq min(max_{\pi \in \Pi}\{len(\pi|_T)\}, n_u^k \times |T|)$.* ■

The key observation in the proof of this property is that in order to take a transition, the preconditions of the associated VAS and of the associated RBAC4BPEL system must be satisfied. Because of the modularity of the post-image, the duality between the set of forward reachable states and the formulae labeling the symbolic execution tree can be lifted to VAS+RBAC4BPEL. We are now ready to state the decidability of the VAS+RBAC4BPEL system; see [9, extended version] for the proof.

THEOREM 4.2. *Let $PN := \langle P, T, F\rangle$ be a an acyclic WF net and let $\langle \underline{x}, In_V(\underline{x}), \{\tau_i^V(\underline{x}, \underline{x}') \mid i = 1, ..., n\}\rangle$ be its associated VAS. Further, let $\langle \underline{p}, In_R(\underline{p}), \{\tau_i^R(\underline{p}, \underline{p}') \mid i = 1, ..., n\}\rangle$ be a RBAC4BPEL system with a bounded*

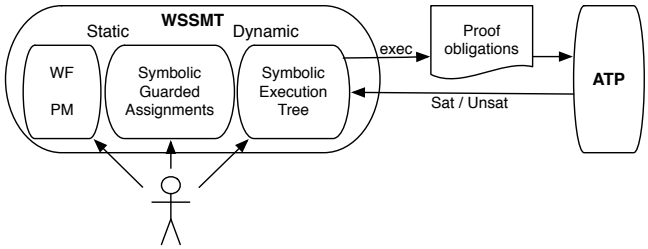

FIG. 4.1. *High level architecture view of WSSMT. All components of the architecture are described in Section 5.*

number of users. Then, the symbolic reachability problem of the VAS+RBAC4BPEL system (whose associated VAS and RBAC4BPEL systems are those specified above) is decidable. ■

To illustrate the kind of formulae arising in the application of Theorem 4.2, we consider the example specified in Fig. 2.1. In this case, we can restrict to consider three paths (projected over the transitions) in the WF net: *crtPO*, *apprPO*, *flow split*, *signGRN*, *ctrSignGRN*, *crtPay*, *flow join*, *apprPay*; *crtPO*, *apprPO*, *flow split*, *signGRN*, *crtPay*, *ctrSignGRN*, *flow join*, *apprPay*; and *crtPO*, *apprPO*, *flow split*, *crtPay*, *signGRN*, *ctrSignGRN*, *flow join*, *apprPay*; each one of length eight. It is easy to see that only the first path is to be considered as the other two produce states that are equivalent since it does not matter at what time *crtPay* is executed with respect to *signGRN* and *ctrSignGRN* (it is possible to mechanize also this check but we leave out the details for lack of space). So, for example, it is possible to check the so-called soundness of workflows [22], i.e., to check whether it is possible to terminate without "garbage" left. In terms of a WF net, this means that no tokens are left in places other than the special final place o of the net. This can be checked by computing the post-images of the initial state of the VAS+RBAC4BPEL system of our motivating example along the lines of Facts 1, 2, and 3 and put this in conjunction with the formula characterizing the "no-garbage" condition, i.e.,

$$x_{10} \geq 1 \wedge \bigwedge_{i=1}^{9} x_i = 0\,.$$

Thanks to the closure under post-image computation of the VAS and the RBAC4BPEL systems, as well as the modularity of the post-image computation for the VAS+RBAC 4BPEL system, the resulting proof obligation is decidable as it can be put in the form $\varphi_V \wedge \varphi_R$ where φ_V is a formula of LA (whose satisfiability is decidable) and φ_R is a BSR formula (whose satisfiability is again decidable), and thus the satisfiability of their conjunction is also decidable.

5. WSSMT: mechanizing the analysis of security-sensitive services. There are two ways to mechanize the symbolic execution technique introduced in Section 3.2: 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, which is an acronym of "Web Service (validation by) Satisfiability Modulo Theories". A detailed description of the tool and its implementation can be found in [5, 31], here we only sketch its main functionalities and architecture, which is depicted in Fig. 4.1.

The main goal of WSSMT is to help users write specifications of security-sensitive services structured along the previously identified directions: WF/PM levels and static/dynamic parts. Once the algebraic structure of the WF and PM levels have been identified (e.g., Linear Arithmetic in the case of a VAS) by means of suitable *first-order theories* (see, e.g., [7] for a discussion of how to use theories to describe the state space of two-level transition systems) and a two-level transition system is specified to describe the possible actions of the system, 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 labeled with state formulae and edges with (instances of) transitions, the front-end must create the appropriate proof obligations (as explained in Section 3) and then invoke an available ATP system, chosen

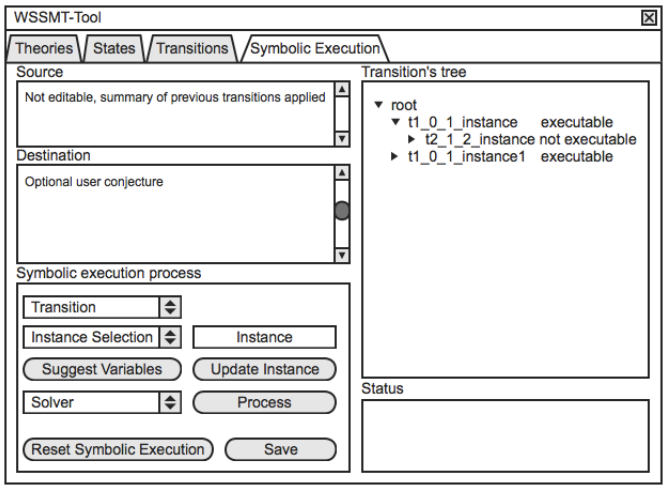

FIG. 5.1. *Symbolic Execution tab.*

by the user among those available in the back-end. Once the ATP has established the satisfiability of the proof 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. 4.1.

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 as specified in Section 4.3. The *Symbolic Execution* tab, depicted in Fig. 5.1, 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 (labeled *Source*) and allows the user to enter the formula to which the execution step should lead (labeled *Destination*) together with a transition chosen from the list of available transitions (combo labeled *Transition*), whose identifiers have been instantiated as explained in the combo labeled *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 Eclipse 3.5 plug-in by exploiting the SWT and JFace libraries [13, 27] 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 [29]. It has been chosen because it supports many-sorted FOL, it is easy to extend, and several tools are available for its parsing and translation.

Currently, WSSMT has been used with SPASS [24], a state-of-the-art resolution-based prover, and the SMT solver Z3 [30]. 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 [26] 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 [23] at the state of writing) 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 FOL). It was easy to create a translator from the DFG syntax to the SMT-LIB input language [18], 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 3). Our experience in using WSSMT for the analysis of some specifications of security-sensitive services can be found in the next section, along with remarks about the performances of the ATP systems to discharge

```
<process name="LOP"/>
  <sequence>
    <invoke> inputCustData </invoke>
    <flow>
        <invoke> prepareContract </invoke>
         <sequence>
         <invoke> intRating </invoke>
         <if>
           <condition> ∼lowRisk </condition>
           <invoke> extRating </invoke>
         </if>
         </sequence>
     </flow>
   <invoke> approve </invoke>
     <if>
       <condition> ∼productOK </condition>
       <invoke> sign </invoke>
     </if>
  </sequence>
</process>
```

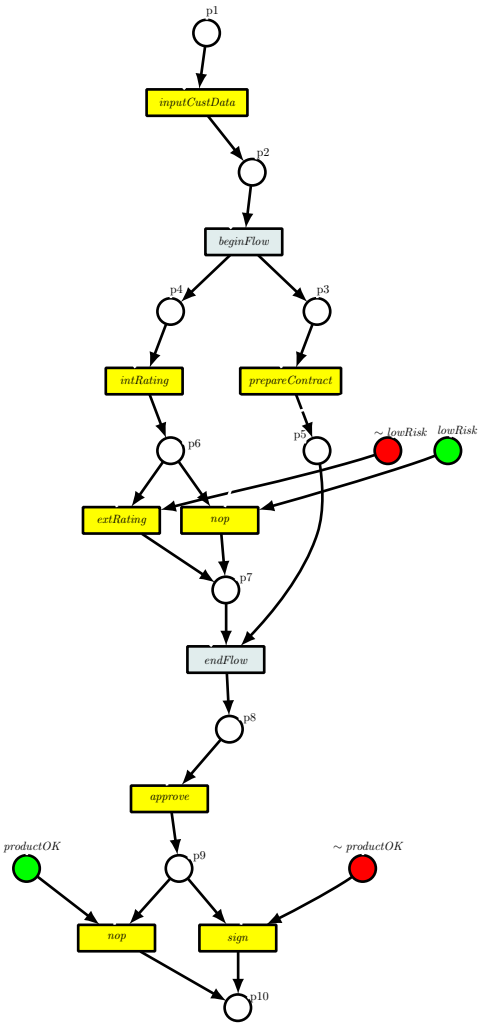

FIG. 6.1. *The LOP process: BPEL (left) and corresponding Petri net (right). Legend: see the legend in Fig. 2.1 and, in addition, consider that places filled in red (resp., green) specify Boolean variables whose value is false (resp., true).*

typical proof obligations.

6. Case studies. In this section, we report on the application of WSSMT to the specification and analysis of two industrial case studies considered in the FP7 European project AVANTSSAR (more details about these and other case studies can be found in [4]): the first one is called the Loan Origination Process and concerns a banking service in the e-business applications area (Section 6.1), whereas the second one is called the Digital Contract Signing and is a protocol in the e-government area (Section 6.2).

6.1. Loan Origination Process. The Loan Origination Process (LOP) is a bank's loan application process, which has been used as an example in many works (see, e.g., [2, 4, 11, 21]). We adapt the variant proposed in [2, 7], whose workflow is presented as a BPEL process and whose policy level is described by RBAC policies with delegation. Fig. 6.1 shows the BPEL description (left) and the corresponding Petri net translation (right) of the LOP. The Petri net has been obtained along the same lines as the one for PO.

Roughly, the LOP is composed of seven activities: the customer starts the process by providing its own data (*inputCustData*) and then the contract is prepared (*prepareContract*) and the customer's rating evaluation is run concurrently. The evaluation activity is performed by first performing an internal evaluation (*intRating*): if the risk associated with the loan is low, then the internal rating is sufficient (*nop*); otherwise, a credit

reporting agency is asked for an external rating (*extRating*). Afterwards, the bank approves the result of the loan evaluation (*approve*) and decides whether to sign the loan (*sign*) or not, entering again in the task (*nop*).

6.1.1. WF level. Similarly to the PO of Fig. 2.1, the order of the execution of the various activities of the LOP should satisfy some constraints to complete successfully. The *inputCustData* must be executed at the beginning of the process, then the flow splits in two so that parallel execution of two sets of tasks is possible. *intRating* and *prepareContract* are concurrent activities and while *prepareContract* is the only task in the right branch (see Fig.6.1), *intRating* in the left branch leads to a fork composed by *extRating* or *nop*. The choice to follow one of these branches will depend on the value of a Boolean variable (external to the flow) *lowRisk*. At this point the two parallel strings can synchronize by executing the task *endFlow*. Then, the task *approve* leads again to split the flow and the choice to follow the task *sign* or *nop* will be done depending on another external Boolean variable *productOK*. At this point, no more transitions are enabled and the execution of the workflow is terminated.

We can map the BPEL description to a WF net and this to a transition system as described in Section 4. In the following, for the sake of brevity, we sketch how to translate some of the markings of the Petri net and a few transitions.

Formally, we represent the set of places (p_i) using a map to the set of non-negative integers. To formalize the *Initial* (resp., *Final*) state of the flow where there is just one token in place p_1 and (resp., p_{10}) all other places are empty, is sufficient to write the following two formulae:

$$\mathit{Initial} \;:=\; p_1 = 1 \wedge \bigwedge_{i=2}^{10} p_i = 0 \;\text{ and }\; \mathit{Final} \;:=\; p_{10} = 1 \wedge \bigwedge_{i=1}^{9} p_i = 0\,.$$

Let us analyze *inputCustData*: the guard of the transition requires that there is one token in place p_1, its update consumes the token in p_1 and produces a token in p_2 while leaving all other places as they are. This can be formally represented by the following formula:

$$\mathit{inputCustData} := p_1 = 1 \wedge p_1' = p_1 - 1 \wedge p_2' = p_2 + 1 \wedge \bigwedge_{i=3}^{10} (p_i' = p_i)\,.$$

All the other transitions are formalized along the same lines.

6.1.2. PM level. Each task can be executed by an agent who is entitled to perform it. For example, the task *inputCustData* can be executed by an employee of the bank who has the role of *preprocessor*, while the task *sign* can be performed by the *director* of the bank. Indeed, the employees have certificates that can be used to play certain roles that are organized hierarchically, i.e., a certain role inherits all the rights owned by roles that are lower in the hierarchy. In this way, for example, the *director* of the bank has all the rights of one of its employees, so that the *director* can, in principle, process the loan application of a customer.

We also consider delegation of tasks. For example, for certain loan applications (e.g., for loans below a certain amount), the *director* of the bank can delegate one of his supervisors to sign the loan contract after its approval. To manage these issues we use the RBAC model, whose main motivation is to map security management to an organization's structure (such as the bank in the LOP). The underlying idea is that each user's role may be based on the user's job responsibilities in the organization. The key ingredients of an RBAC model are the same as the ones introduced in Section 4.2.

Similarly to the case of RBAC4BPEL, to formalize RBAC we have introduced three sort symbols $Id, Role$, and $Task$ for the sets U, R, and T, respectively. To formalize the PM level of the LOP, we assume that the three sort symbols are endowed with the following enumerated data-type theories:

$$T_{Id} := \mathit{Enum}(\{\mathit{davide}, \mathit{maria}, \mathit{marco}, \mathit{pierPaolo}, \mathit{pierSilvio}\}, Id),$$
$$T_{Role} := \mathit{Enum}(\{\mathit{preprocessor}, \mathit{postprocessor}, \mathit{supervisor}, \mathit{manager}, \mathit{director}\}, Role),$$
$$T_{Task} := \mathit{Enum}(\{\mathit{inputCustData}, \mathit{prepareContract}, \mathit{intRating}, \mathit{extRating}, \mathit{approve}, \mathit{sign}\}, Task).$$

Note that constant symbols of sort $Task$ have exactly the same names of transition labels of the Petri net in Fig. 6.1 (right).

Since a user can be associated to several roles, s/he must choose to activate a role (with the appropriate rights) to execute a task. Formally, we introduce a dynamic predicate symbol $\mathit{activated} : Id \times Role$ such that

$activated(i, r)$ holds whenever the user i has chosen to become active in role r (if this is possible according to the relation ua). We modeled activation and deactivation of roles with the following two transitions: the first formalizes the fact that an agent is active in a certain role if it was not already playing the given role and it can be assigned to it

$$\exists i.\ \left(\begin{array}{l} ua(i,\rho) \wedge \neg activated(i,\rho) \wedge \\ \forall j, r.(activated'(j,r) \Leftrightarrow \left(\begin{array}{l} \textit{if } (j = i \wedge r = \rho) \\ \textit{then true else } activated(j,r) \end{array} \right)) \end{array} \right) \wedge \upsilon$$

and, dually, the second formula models deactivation from a role

$$\exists i.\ \left(\begin{array}{l} activated(i,\rho) \wedge \\ \forall j, r.(activated'(j,r) \Leftrightarrow \left(\begin{array}{l} \textit{if } (j = i \wedge r = \rho) \\ \textit{then false else } activated(j,r) \end{array} \right)) \end{array} \right) \wedge \upsilon\,,$$

where, for both activation and deactivation, $\rho \in \Lambda$, $\Lambda := \{preprocessor, postprocessor, supervisor, manager, director\}$, and $\upsilon := \forall i, r.\ (ua'(i,r) \Leftrightarrow ua(i,r)) \wedge \forall i, t.\ (pa'(i,t) \Leftrightarrow pa(i,t))$.

The RBAC model has been widely adopted to organize policy access for large organizations (see, e.g., [21]). However, even when a task is typically handled by an employee with a certain role, it may be the case that, under certain conditions, the employee wishes to delegate its right to execute a task to another employee belonging to a role down in the hierarchy. So, delegation is a key for the flexibility and scalability of service-oriented applications. In the LOP, roles can be delegated according to the following rules:

1. if the loan has been approved internally, then a *manager* can delegate a *supervisor* to approve a loan,
2. if the loan has been approved internally, then a *manager* can delegate a *supervisor* to sign the loan,
3. if the loan does not require an external rating, then the *director* can delegate a *manager* to sign the loan,
4. if the loan does not require an external rating, then a *supervisor* can delegate a *postprocessor* to perform an external rating of a loan.

To formalize this, we add the dynamic predicate $delegated : Id \times Role \times Task$ to the PM level state, and the two Boolean variables $intRatingOK$ and $highValue$ to the WF level state. If the user i is delegated to execute task t by a user who has role r, then $delegated(i, r, t)$ holds; $intRatingOK$ is the result of the internal (to the bank) evaluation of the loan application (since the logic determining the value of this variable is not modeled, the value of $intRatingOK$ will not be modified by any transition); and $highValue$ is set to true if the amount of the loan is greater than a certain threshold established by the bank (since the precise value of the threshold is unimportant for our model, the value of $highValue$ will not be modified by any transition). By using this and the previously introduced dynamic predicates, the transitions formalizing the role delegation rules above

can be written as follows:

$$1.\ \exists i_1, i_2.\ \left(\begin{array}{l} intRatingOK \wedge ua(i_1, manager) \wedge \\ \quad ua(i_2, supervisor) \wedge pa(manager, approve) \wedge \\ \quad \forall i, r, t.\ delegated'(i, r, t) \Leftrightarrow \\ \quad \left(\begin{array}{l} if\ (i = i_2 \wedge r = manager \wedge t = approve) \\ then\ true\ else\ delegated(i, r, t) \end{array}\right) \wedge \upsilon \end{array}\right)$$

$$2.\ \exists i_1, i_2.\ \left(\begin{array}{l} intRatingOK \wedge ua(i_1, manager) \wedge \\ \quad ua(i_2, supervisor) \wedge pa(manager, sign) \wedge \\ \quad \forall i, r, t.\ delegated'(i, r, t) \Leftrightarrow \\ \quad \left(\begin{array}{l} if\ (i = i_2 \wedge r = manager \wedge t = sign) \\ then\ true\ else\ delegated(i, r, t) \end{array}\right) \wedge \upsilon \end{array}\right)$$

$$3.\ \exists i_1, i_2.\ \left(\begin{array}{l} \neg highValue \wedge ua(i_1, director) \wedge \\ \quad ua(i_2, manager) \wedge pa(director, sign) \wedge \\ \quad \forall i, r, t.\ delegated'(i, r, t) \Leftrightarrow \\ \quad \left(\begin{array}{l} if\ (i = i_2 \wedge r = director \wedge t = sign) \\ then\ true\ else\ delegated(i, r, t) \end{array}\right) \wedge \upsilon \end{array}\right)$$

$$4.\ \exists i_1, i_2.\ \left(\begin{array}{l} \neg highValue \wedge ua(i_1, supervisor) \wedge \\ \quad ua(i_2, postprocessor) \wedge pa(supervisor, extRating) \wedge \\ \quad \forall i, r, t.\ delegated'(i, r, t) \Leftrightarrow \\ \quad \left(\begin{array}{l} if\ (i = i_2 \wedge r = supervisor \wedge t = extRating) \\ then\ true\ else\ delegated(i, r, t) \end{array}\right) \wedge \upsilon \end{array}\right),$$

where $\upsilon := \forall i, r.\ (ua'(i, r) \Leftrightarrow ua(i, r)) \wedge \forall i, r.(activated'(i, r) \Leftrightarrow activated(i, r)) \wedge \forall i, t.\ (pa'(i, t) \Leftrightarrow pa(i, t))$.

At this point, we are able to characterize when an agent i playing role r is granted the right to execute task t: either i is delegated to execute task t by a user in role r or user i is assigned to role r, i is active in that role, and there exists a role ρ such that $r \succeq \rho$ and ρ has the permission to perform task t. To formalize this, we add the predicate $granted\ :\ Id \times Role \times Tas$ as well as the axioms

$$\forall i, r, t.\ granted(i, r, t)\ \Leftrightarrow \left(\begin{array}{l} delegated(i, r, t) \vee \\ (ua(i, r) \wedge activated(i, r) \wedge (r \succeq \rho \wedge pa(\rho, t))) \end{array}\right)$$

for each $\rho \in \Lambda$. The predicate *granted* is the key to constrain the transitions describing the workflow of the application in Fig. 6.1. Formally, this is done by incorporating suitable applications of the predicate *granted* in the guards of the transitions. Before giving the transition system characterizing the interplay between the WF and the PM levels, we introduce a further dynamic predicate symbol $executed\ :\ Id \times Task$ to keep track of the fact that user i has performed a task t, i.e., $executed(i, t)$. As it will be clear below, this enables us to specify certain crucial security properties of the Loan Origination Process.

Symbolic execution and debugging. We now show how the capability of automatically checking executability of scenarios can be useful for debugging a specification. In particular, we show that the specification of the LOP given above violates a crucial security property, namely SoD: "for each agent i, there exists a task in the workflow that is never executed by i". This property is violated if we can find a sequence of transitions leading the LOP from the initial to the final state and the following formula

$$\forall i. \exists t. \left(\begin{array}{l} ((t \neq extRating \wedge t \neq sign) \Rightarrow \neg executed(i, t)) \wedge \\ (lowRisk \Rightarrow \neg executed(i, extRating)) \wedge \\ (productOK \Rightarrow \neg executed(i, sign)) \end{array}\right) \tag{6.1}$$

expressing SoD for the workflow of the LOP, is unsatisfiable. To this end, we can show that the sequence of tasks: *inputCustData*, *beginFlow*, *prepareContract*, *intRating*, *nop*, *endFlow*, *approve*, and *sign*, starting from a certain initial state, lead the transition system to a final state by generating a trace that violates (6.1) because each task can be executed by just one user. The scenario is taken from [2] and can be easily checked by using WSSMT (all proof obligations are discharged almost immediately). For a full description of the symbolic execution of the scenario showing that user *pierSilvio* can execute all tasks specified above, thereby violating SoD, the reader is once more pointed to [7].

6.2. 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 manually translating the BPEL files to Petri nets and then derive the associated VASs, as we were able to do for the PO and the LOP, we used the freely available tool BPEL2oWFN [15]. Since DCS consists of four BPEL processes (one corresponding to a signer, one for *BP*, one for *SServ*, and one for *PKI*), we used BPEL2oWFN to compose the instances of the BPEL processes for the execution of the protocol, and we used it also to compute the corresponding Petri net. We have then processed this to derive the VAS in the input syntax of WSSMT.

To have an idea of the dimension of the problem, we sketch in Fig. 6.2 the Petri net generated by BPEL2oWFN consisting of 51 places (orange places represent data exchange between instances of different BPEL processes) and 26 transitions, while we omit the specification of the BPEL processes because it would take too much space.

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. As for the PO process, we have used RBAC4BPEL: the formalization that we used is along the same lines as those in Section 4.2. For example, we have two relations: 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). As before, 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 FOL, 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 SoD (e.g., the user signing the contract should not be the same as the one checking the validity of the signature on the contract)—as for the LOP process—and BoD (e.g., the users signing the contract should be same that have agreed on the conditions of the contract) authorization constraints. Further details can be found in [9].

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 [4]. 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 (whereas 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 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., [14]). 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.

7. Conclusions. To conclude the paper, let us briefly summarize our contributions. We have described automated formal analysis techniques for the validation of a class of web services specified in BPEL and

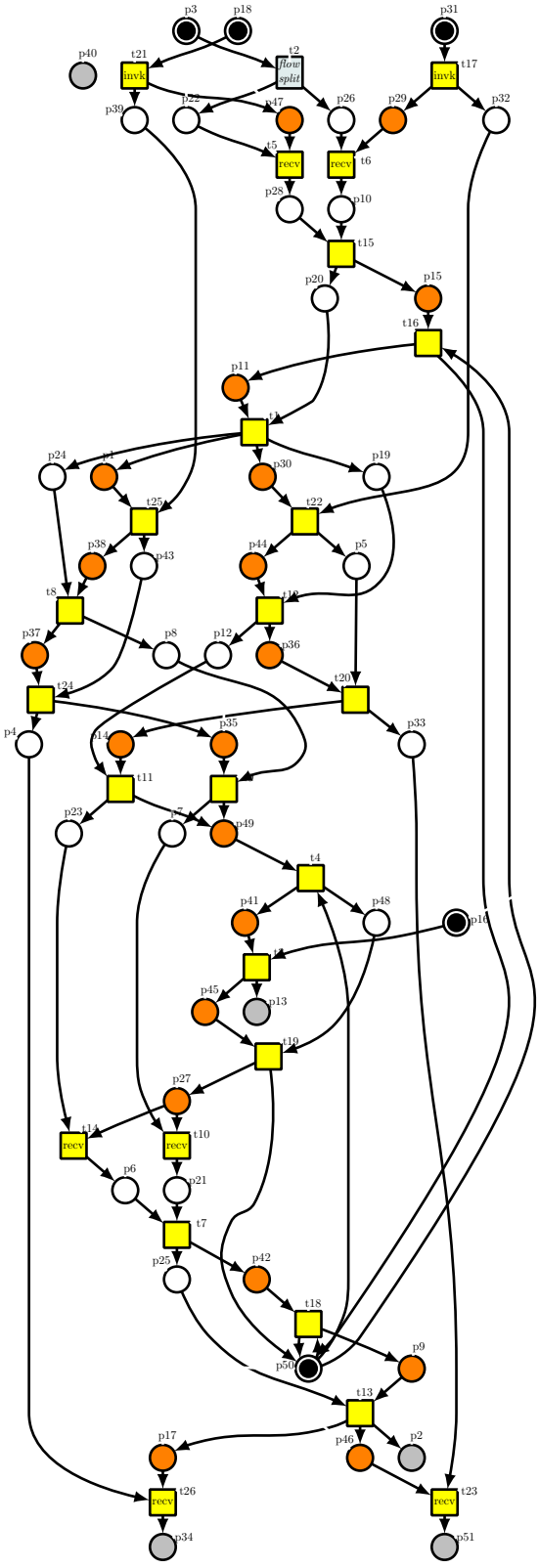

FIG. 6.2. *The Petri net representation of the WF level of the DCS case study. Legend: see Fig. 6.1 and, in addition, consider that orange places specify data exchange between instances of different BPEL processes.*

RBAC4BPEL. We also discussed our experience in using our prototype tool WSSMT on two industrial case studies, a loan origination process and a digital contract signing service, which have provided proof of concept of the flexibility of our specification and analysis framework, which allowed us to precisely capture the interplay between the workflow and the access-control level of the service. Throughout the paper, we have referred to related work and already mentioned several interesting directions for future work. In particular, besides for considering other case studies, we plan to extend our decidability results to WF nets containing restricted forms of loops and extensions of RBAC4BPEL with delegation, as well as to consider in more detail the interplay of the WF and PM levels with the data flow of the services under validation.

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] A. Alves et al Web Services Business Description Language, Version 2.0, OASIS Standard, April 2007. Available at `http://docs.oasis-open.org/wsbpel/2.0/wsbpel-v2.0.pdf`.

[2] A. Armando and S. Ponta. Model checking of security-sensitive business processes. In P. Degano and J. Guttman, editors, *Formal Aspects in Security and Trust*, volume 5983 of *Lecture Notes in Computer Science*, pages 66–80. Springer Berlin / Heidelberg, 2010. `10.1007/978-3-642-12459-4\_6`.

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

[4] AVANTSSAR. Deliverable 5.1: Problem cases and their trust and security requirements, 2008.

[5] M. Barletta, A. Calvi, S. Ranise, L. Viganò, and L. Zanetti. WSSMT: Towards the Automated Analysis of Security-Sensitive Services and Applications. In *12th International Symposium on Symbolic and Numeric Algorithms for Scientific Computing*, pages 417–424, Los Alamitos, CA, USA, 2010. IEEE Computer Society. Extended version available at arXiv: `http://arxiv.org/abs/1009.4625`.

[6] M. Barletta, S. Ranise, and L. Viganò. Verifying the Interplay of Authorization Policies and Workflow in Service-Oriented Architectures. In *IEEE CSE'09, 12th IEEE International Conference on Computational Science and Engineering*, pages 289–296. IEEE Computer Society, 2009.

[7] 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*, pages 1–33, 2010. 10.1007/s11761-010-0073-4.

[8] A. Bockmayr and V. Weispfenning. *Handbook of Automated Reasoning, volume I*, chapter Chap. 12 "Solving numerical constraints", pages 751–842. Elsevier Science B.V., 2001.

[9] A. Calvi, S. Ranise, and L. Viganò. Automated Validation of Security-sensitive Web Services specified in BPEL and RBAC. In *12th International Symposium on Symbolic and Numeric Algorithms for Scientific Computing*, pages 456–464, Los Alamitos, CA, USA, 2010. IEEE Computer Society. Extended version available at arXiv: `http://arxiv.org/abs/1009.4625`.

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

[11] A. Deutsch, L. Sui, V. Vianu, and D. Zhou. Verification of communicating data-driven web services. In *Proceedings of the twenty-fifth ACM SIGMOD-SIGACT-SIGART symposium on Principles of database systems*, PODS'06, pages 90–99, New York, NY, USA, 2006. ACM.

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

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

[14] S. Lahiri, S. Seshia, and R. Bryant. Modeling and Verification of Out-of-Order Microprocessors in UCLID. In M. Aagaard and J. OLeary, editors, *Formal Methods in Computer-Aided Design*, volume 2517 of *Lecture Notes in Computer Science*, pages 142–159. Springer Berlin / Heidelberg, 2002.

[15] N. Lohmann and C. Gierds and M. Znamirowski. BPEL2OWFN. Available at `http://www.gnu.org/software/bpel2owfn`.

[16] F. Paci, E. Bertino, and J. Crampton. An Access-Control Framework for WS-BPEL. *Int. Journal of Web Services Research*, 5(3):20–43, 2008.

[17] R. Piskac, L. de Moura, and N. Bjoerner. Deciding Effectively Propositional Logic Using DPLL and Substitution Sets. *J. of Autom. Reas.*, 44(4):401–424, 2010.

[18] S. Ranise and C. Tinelli. The satisfiability modulo theories library (smt-lib). `http://www.smt-lib.org`, 2009.

[19] R. Sandhu, E. Coyne, H. Feinstein, and C. Youmann. Role-Based Access Control Models. *IEEE Computer*, 2(29):38–47, 1996.

[20] S. Sankaranarayanan, H. Sipma, and Z. Manna. Petri Net Analysis Using Invariant Generation. In N. Dershowitz, editor, *Verification: Theory and Practice*, volume 2772 of *Lecture Notes in Computer Science*, pages 188–189. Springer Berlin / Heidelberg, 2004. 10.1007/978-3-540-39910-0_29.

[21] A. Schaad, K. Sohr, and M. Drouineaud. A Workflow-based Model-checking Approach to Inter- and Intra-analysis of Organisational Controls in Service-oriented Business Processes. Journal of Information Assurance and Security, 2(1): 55–67, 2007.

[22] H. Schlingloff, A. Martens, and K. Schmidt. Modeling and Model Checking Web Services. *ENTCS*, 126:3–26, 2005.
[23] SMT-COMP. The SMT-COMP web-site. `http://www.smtcomp.org`.
[24] SPASS. The SPASS web-site. `http://www.spass-prover.org`.
[25] C. Stahl. A Petri Net Semantics for BPEL. Technical report, Humboldt-Universität zu Berlin, Institut für Informatik, 2004.
[26] G. Sutcliffe. The TPTP web-site. `http://www.cs.miami.edu/~tptp`.
[27] SWT: The Standard Widget Toolkit. `http://www.eclipse.org/swt`.
[28] W. van der Aalst. The application of Petri nets to workflow management. *J. of circuits, systems, and computers*, 8(1):21–66, 1998.
[29] C. Weidenbach. Spass input syntax version 1.5. `www.spass-prover.org/download/binaries/spass-input-syntax15.pdf`.
[30] Z3 theorem prover. `http://research.microsoft.com/projects/z3`.
[31] L. Zanetti. *Integrazione di tecniche SMT in un sistema di verifica per applicazioni orientate ai servizi.* Master Thesis, Faculty of Mathematical, Physical and Natural Sciences, University of Verona, Italy, 2009.

Edited by: Dana Petcu and Alex Galis
Received: March 1, 2011
Accepted: March 31, 2011