Verifying the Interplay of Authorization Policies and Workflow in Service-Oriented Architectures

Michele Barletta Silvio Ranise Luca Viganò
Department of Computer Science, University of Verona, Italy
{michele.barletta | silvio.ranise | luca.vigano}@univr.it

***Abstract*—A widespread design approach in distributed applications based on the service-oriented paradigm, such as web-services, consists of clearly separating the enforcement of authorization policies and the workflow of the applications, so that the interplay between the policy level and the workflow level is abstracted away. While such an approach is attractive because it is quite simple and permits one to reason about crucial properties of the policies under consideration, it does not provide the right level of abstraction to specify and reason about the way the workflow may interfere with the policies, and vice versa.**

In this paper, we present a two-level formal verification framework to overcome these problems and formally reason about the interplay of authorization policies and workflow in service-oriented architectures. This allows us to define and investigate some verification problems for SO applications and give sufficient conditions for their decidability.

I. Introduction

A widespread design approach in distributed applications based on the Service-Oriented paradigm (SO), such as web-services, consists of clearly separating the Workflow (WF) from the Policy Management (PM). The former orchestrates complex processing of data performed by the various principals using a set of resources made available in the application, while the latter aims to regulate access decisions to the shared resources, based on policy statements made by the involved principals. This separation of concerns is beneficial in several respects for the design, maintenance, and verification of the resulting applications such as reusing policies across applications.

One of the key problems in obtaining a correct design of SO applications is to be able to foresee all the—sometimes subtle—ways in which their WF and PM levels interact. To understand the difficulty underlying this endevor, let us first consider the WF level. In this respect, SO applications can be seen as distributed systems whose transitions can be interleaved in many possible ways. This already creates a first difficult problem: to understand the behaviors of an SO application and then to establish if it meets certain properties. An additional burden, typical to SO applications, is the presence of the PM level, which is supposed to constrain the allowed behaviors of the application so as to meet certain crucial security requirements. Declarative policy languages (such as Datalog and other languages built on top of it, like Binder [11], SecPal [6] and DKAL [13]), usually based on a (fragment of) first-order logic, are used to design the PM level of SO applications in a more flexible, reusable, and verification-friendly way. The high flexibility and expressiveness of such languages may grant access to a resource to someone who, in the intention of the policy designer, is not allowed to do so.

To further complicate the situation, there are the subtle ways in which the WF and the PM levels may interact so as to give rise to behaviors that are unintended and may breach some crucial security requirements of an SO application. As a concrete example of this point, consider a system for virtual Program Committee meetings. A policy governing access to the reviews of a paper may be the following: a reviewer assigned to a paper is required to submit his own review before being able to read those of the others. So, in order to resolve an access request to the reviews of a paper, the system should be able not only to know the identities and the roles of the various members of the Program Committee but also to maintain and consult the information about which reviewers have already submitted their reviews. Indeed, information of the first kind must be derived from the WF level.

Given all the difficulties to obtain correct designs for SO applications, formal methods have been advocated to help in this task. Unfortunately, most (see, e.g., [12], [20]) of the specification and verification techniques (with some notable exceptions, e.g., [17]) have concentrated on one level at a time and abstracted away the possible interplays between the WF and the PM level. The *first contribution* of this paper is a framework capable of formalizing both the WF and the PM level as well as their interface so as to enable a more precise analysis of the possible behaviors of SO applications. In particular, we use a temporal extension of first-order logic, similarly to what has been proposed in [14] for the specification and verification of reactive systems. The motivations for this choice are three-fold. First, workflows can be easily specified by using first-order formulae to describe sets of states and transitions of SO applications. Second, a simple extension of this well-known framework allows us to easily specify policy-relevant facts and statements. Third, we hope to adapt to the case of SO applications the cornucopia of specification and verification techniques developed for reactive systems. As a first step in this direction, the *second contribution* of this paper is to define and investigate some verification problems for SO applications and give sufficient conditions for their decidability. In particular, we show how executability and some security properties (which can be expressed as invariants) can be automatically verified within the proposed framework.

We proceed as follows. In Section II, we summarize the key points of a restricted combination of first-order and temporal

logic, which provides a formal basis for our approach. In Section III, we present our formal two-level specification framework for SO applications, which we apply, in Section IV, on a number of interesting verification problems for SO applications. In Section V, we discuss related and future work, and draw conclusions. Due to lack of space, proofs are given in the accompanying technical report [5], which also contains the detailed formalization of a case study that illustrates our framework at work, as well as a number of useful pragmatical observations.

II. A RESTRICTED COMBINATION OF FIRST-ORDER LOGIC AND TEMPORAL LOGIC

As a formal basis for our approach we use a standard [14] minimal extension of *Linear Time Logic* (*LTL*) with a *many-sorted version of First-Order Logic with equality* (*FOL*=). We recall now some useful definitions and properties of FOL= where, for brevity, we do not explicitly consider sorts although all notions can be easily adapted to the many-sorted version. We assume the usual first-order syntactic notions of *signature*, *term*, *literal*, *formula*, *quantifier-free formula*, *substitution* and *grounding* substitution, *sort* and so on, and call *sentence* a formula that does not contain free variables. Also the semantic notions of *structure*, *satisfiability*, *validity*, and *logical consequence* are the standard ones.

Let Σ be a FOL= signature. An *expression* is a term, an atom, a literal, or a formula. A $\Sigma(\underline{x})$*-expression* is an expression built out of the symbols in Σ where at most the variables in the sequence $\underline{x}$ of variables may occur free, and we write $E(\underline{x})$ to emphasize that E is a $\Sigma(\underline{x})$-expression. Similarly, for a finite sequence $\underline{r}$ of predicate symbols in Σ, we write $\phi(\underline{r})$ to denote a Σ-formula where at most the predicate symbols in $\underline{r}$ may occur. We juxtapose sequences to denote their concatenation, e.g. $\underline{xy}$, and abuse notation and write $\emptyset$ to denote the empty sequence besides the empty set. If σ is a substitution and $\underline{t}$ is a (finite) sequence of expressions, then $\underline{t}\sigma$ is the sequence of expressions obtained from $\underline{t}$ by applying the substitution σ to each element of $\underline{t}$.

Following [14], we use a tuple $\underline{x}$ of variables, called *WF state variables*, to represent the values of application variables at a given instant of time, and use a FOL formula $\varphi(\underline{x})$ to represent sets of states. WF state variables take values in the domain of a first-order structure, which formalizes the data structures, the values of the control locations, and those of the auxiliary variables of the WF of a certain SO application. Formally, let Σ be a signature (containing, e.g., the operators of certain data structures or the names of some control locations) and $\mathcal{M}$ be a Σ-structure; $\mathcal{M}, v \models \varphi(\underline{x})$ means that the state formula $\varphi(\underline{x})$ is true in $\mathcal{M}$ for the valuation v mapping the variables in $\underline{x}$ to elements of the domain of $\mathcal{M}$. As shown in [14], this is enough for the specification of virtually any reactive system and hence also for the WF level. However, the state of SO applications should also support the PM level whose relevant part is represented by tables where certain facts are recorded (e.g., "is-reviewer-of" for the example in the introduction). Following the relational model of databases, we formalize tables as predicates and we add to the WF state variables a set $\underline{p}$ of fresh predicate symbols (i.e. $\underline{p} \cap \Sigma = \emptyset$), called *PM state variables*. Any Σ-formula $\varphi(\underline{x}, \underline{p})$ is an *SO state formula*. For a Σ-structure $\mathcal{M} = (I, D)$, a valuation v mapping the WF state variables in $\underline{x}$ to elements of the domain D, and a relational valuation b mapping the PM state variables in $\underline{p}$ (such that $\underline{p} \cap \Sigma = \emptyset$) to the powerset of D, we write

$$\mathcal{M}, v, b \models \varphi(\underline{x}, \underline{p})$$

to denote that $\mathcal{M}_b, v \models \varphi(\underline{x}, \underline{p})$ where $\mathcal{M}_b = (I', D')$ is the $(\Sigma \cup \underline{p})$-structure obtained from $\mathcal{M}$ by taking $D' = D$, $I'|_\Sigma = I$, and $I'(p) = b(p)$ for each $p \in \underline{p}$. The tuple $(\mathcal{M}, v, b)$ (or simply v, b, when $\mathcal{M}$ is clear from context) is an *SO state*.

Let Σ be a signature and $\mathcal{M}$ be a Σ-structure. We formalize an SO application by a tuple $(\underline{x}, \underline{p}, \iota, Tr)$, called an *SO transition system*, where $\underline{x}$ are the WF state variables, $\underline{p}$ are the PM state variables, ι is a $\Sigma(\underline{x}, \underline{p})$-formula, and Tr is a finite set of $\Sigma(\underline{x}, \underline{p}, \underline{x}', \underline{p}')$-formulae, called *transitions*, which relate a set of SO states (identified by the "values" of $\underline{x}, \underline{p}$) to that of a set of SO "next" states (identified by the "values" of $\underline{x}', \underline{p}'$).[1] If $\underline{p} = \emptyset$ and $\underline{x} \neq \emptyset$, then our notion of SO transition system reduces to that of transition system in [14]; in the rest of this paper, we assume that $\underline{p} \neq \emptyset$.

A *run* of an SO transition system $(\underline{x}, \underline{p}, \iota, Tr)$ is an infinite sequence of SO states $v_0, b_0, ..., v_i, b_i, ...$ such that $\mathcal{M}, v_0, b_0 \models \iota(\underline{x}, \underline{p})$ and for every $i \geq 0$, there exists a transition $\tau(\underline{x}, \underline{p}, \underline{x}', \underline{p}') \in Tr$ such that $\mathcal{M}, v_i, b_i, v_{i+1}, b_{i+1} \models \tau(\underline{x}, \underline{p}, \underline{x}', \underline{p}')$ where v_i, b_i (respectively, v_{i+1}, b_{i+1}) map state variables and predicates in $\underline{x}, \underline{p}$ (respectively, $\underline{x}', \underline{p}'$).

To specify properties of SO transition systems, we use an extension of Linear Time Logic. Formally, let $\underline{x}, \underline{p}$ be SO state variables and Σ be a signature; the set $LTL(\Sigma, \underline{x}, \underline{p})$ of *LTL (state-based) formulae for* Σ *and* $\underline{x}, \underline{p}$ is inductively defined as follows: state formulae are in $LTL(\Sigma, \underline{x}, \underline{p})$ and if φ is a state formula then $\Box\varphi$ is in $LTL(\Sigma, \underline{x}, \underline{p})$.[2] Note that we prohibit alternation of FOL quantifiers and temporal operators: this makes the logic less expressive but it helps to derive decidability results for the satisfiability problem, which is a necessary condition to develop (semi-)automatic verification methods for SO applications.

Let $\mathcal{M}$ be a Σ-structure. A *model* of an $LTL(\Sigma, \underline{x}, \underline{p})$-formula is an infinite sequence $v_0, b_0, ..., v_i, b_i, ...$ of SO states such that each v_i, b_i map all the SO state variables in $\underline{x}, \underline{p}$, for $i \geq 0$. We then say that an $LTL(\Sigma, \underline{x}, \underline{p})$-formula $\varphi(\underline{x}, \underline{p})$ is *true in a model* $v_0, b_0, ..., v_i, b_i, ...$, and write

$$\mathcal{M}, v_0, b_0, ..., v_i, b_i, ... \models \varphi(\underline{x}, \underline{p}),$$

iff

- $\mathcal{M}, v_0, b_0 \models \varphi$ whenever $\varphi(\underline{x}, \underline{p})$ is a state formula;

[1] We ignore fairness assumptions as, for simplicity in this paper, we are only concerned with security properties that can be encoded as a sub-class of safety properties.

[2] The minimalist temporal logic defined here suffices for the purposes of specifying the sub-class of invariant properties that are relevant for this paper. However, the proposed framework may be easily extended to support other temporal operators such as "sometimes in the future," "next", or "until."

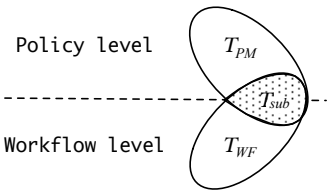

Fig. 1. FOL formalization of the WF and PM levels of SO applications

- $\mathcal{M}, v_0, b_0, ..., v_i, b_i, ... \models \Box\varphi(\underline{x}, \underline{p})$ iff $\mathcal{M}, v_k, b_k \models \varphi(\underline{x}, \underline{p})$, for every $k \geq 0$.

Let $\mathcal{S} = (\underline{x}, \underline{p}, \iota, Tr)$ be an SO transition system, $\mathcal{M}$ be a Σ-structure, and $\varphi(\underline{x}, \underline{p})$ be an $LTL(\Sigma, \underline{x}, \underline{p})$-formula. Then, $\mathcal{S} \models \varphi(\underline{x}, \underline{p})$ iff $\mathcal{M}, v_0, b_0, ..., v_i, b_i, ... \models \varphi(\underline{x}, \underline{p})$ for every run $v_0, b_0, ..., v_i, b_i, ...$ of $\mathcal{S}$. A state formula $\psi(\underline{x}, \underline{p})$ is an *invariant for the SO transition system* $\mathcal{S}$ if $\mathcal{S} \models \Box\psi(\underline{x}, \underline{p})$.

III. A formal two-level specification framework for SO applications

Recall that one of the main goals of this paper is to provide a natural specification framework for SO applications whose architecture is organized in two levels. Indeed, it is possible to model a large class of SO applications by using the notion of SO transition system introduced in the previous section. However, a good framework should provide an adequate support to restrict the design space for a two-level SO application and allow the designer to easily specify the WF level, the PM level, and their interface in isolation according to a divide-and-conquer strategy. In our framework, this consists of identifying suitable first-order structures formalizing both the data structures at the WF level and the tables at the PM level. Unfortunately, working with first-order structures for specification is quite difficult since there is no obvious way to mechanically represent and reason about them. Fortunately, first-order theories are sets of FOL= sentences that can be used as reasonably precise approximations of first-order structures and for which there is automated reasoning support. Hence, we decided to use theories to describe the WF, the PM levels, and their interface, as illustrated in Fig. 1: if T_{WF} and T_{PM} are the theories formalizing the WF and the PM levels, respectively, then their intersection $T_{sub} = T_{WF} \cap T_{PM}$, called the *substrate theory*, formalizes their interface.

Intuitively, the theory T_{sub} ensures that the WF and PM levels "agree" on certain notions. For example, T_{sub} univocally identifies the principals involved in the SO application and possibly (an abstraction of) the structure of the resources that the SO application can access or make available. As we will see, the use of theories allows us to easily import declarative policy specifications expressed in logical languages based on (extensions of) Datalog in our framework.

A similar approach can also be used to restrict the formulae characterizing transitions. Intuitively, the transitions that can be specified by formulae in the identified class are such that the updates on the values of both the WF and the PM state variables are functional (if we regard relations as first-order objects). Since the identities of the principals involved in the SO application being specified play a crucial role in enabling or disabling the possibility to execute a certain transition, the functional updates will depend not only on the values of the actual SO state but also on the existence of certain principals.

Before being able to formalize these intuitions, we recall the concept of FOL= theory and some standard related notions.

A. First-order theories for SO applications

A Σ-*theory* T is a set of first-order Σ-sentences, called the *axioms* of T. A Σ-structure $\mathcal{M}$ is a *model* of the Σ-theory T iff all the sentences in T are true in $\mathcal{M}$. A Σ-theory T is *consistent* if it admits at least one model. The T-*satisfiability problem* for a quantifier-free Σ-formula $\varphi(\underline{x}, \underline{p})$ (such that $\underline{p} \cap \Sigma = \emptyset$) consists of checking whether there exists a model $\mathcal{M}$ of T and mappings v and b such that $\mathcal{M}, v, b \models \varphi(\underline{x}, \underline{p})$. By transformation in disjunctive normal form (i.e. as disjunctions of conjunctions of literals), the T-satisfiability problem for quantifier-free formulae can be reduced to the T-satisfiability problem for (quantifier-free) conjunctions of literals.

For specifying SO applications, we usually need to introduce a finite set of unique identifiers to name the various principals. Formally, this can be done by using a theory of the following kind. An *enumerated data-type theory* $EDT(C, S)$ is axiomatized by the sentences

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

for S a sort symbol in the given signature (omitted for simplicity) and $C = \{c_1, ..., c_n\}$, c_i of sort S for $i = 1, \ldots, n$ and $n \geq 1$. It is easy to see that the $EDT(C, S)$-satisfiability problem is decidable. Enumerated data-type theories will be sub-theories of the theory formalizing the interface between the WF and PM levels.

For the WF level, we can reuse all the theories available in the literature formalizing data structures and the decision procedures for their satisfiability problem (see, e.g., [18] for an overview). Enumerated data-type theories are also useful for the WF level as they can formalize the (finitely many) control locations of an application. We now give a concrete example of a theory capable of formalizing a simple message passing network that is relevant for SO applications (see [5] for a more detailed case study).

Example 1 (Message passing): A net can be seen abstractly as a set of messages: sending a message amounts to adding the message to the set while receiving a message consists of checking if it is a member of the net; hence, messages are never deleted, only added to the set representing the net. This view is simple but still allows one to model interesting facts such as the reception of messages in any order (since a set does not require an ordering on its elements) or duplication of messages (as a message is never removed from the net).

To model the simple fragment of set theory necessary to formalize this idea in FOL, we use a theory $MsgPass[Msg]$, parametrized over the sort Msg of messages which contains $SetOfMsg$ as the sort for sets of messages, the constant mty

of sort $SetOfMsg$ denoting the empty set, the binary function symbol ins of sort $Msg \times SetOfMsg \rightarrow SetOfMsg$ denoting the operation of adding a message to a set of messages, and the binary predicate symbol mem of sort $Msg \times SetOfMsg$ for checking if a message is in a set of messages. The axioms of $MsgPass[Msg]$ are the following three sentences:

$$\forall E.\, \neg \mathsf{mem}(E, \mathsf{mty}) \qquad \forall E.\, \mathsf{mem}(E, \mathsf{ins}(E, S))$$
$$\forall E, E'.\, E \neq E' \rightarrow (\mathsf{mem}(E, \mathsf{ins}(E', S)) \leftrightarrow \mathsf{mem}(E, S))$$

where E, E' are variables of sort Msg and S is a variable of sort $SetOfMsg$. It is easy to describe the states of a variable net: just introduce a logical variable net and use suitable formulae from the theory $MsgPass[Msg]$. For example, the formula $\exists m_1, m_2 : Msg, net' : SetOfMsg.\, m_1 \neq m_2 \wedge net = \mathsf{ins}(m_1, \mathsf{ins}(m_2, net'))$ constrains net to contain at least two messages (plus possibly others). Note that the only free variable in the formulae describing sets of states is net. The $MsgPass[Msg]$-satisfiability problem is decidable [1]. ■

For the PM level, we recall the class of Bernays-Schönfikel-Ramsey (BSR) sentences [8], which has been used, among other applications, to model relational databases. A *BSR-theory* is a set of sentences of the form

$$\exists \underline{x}\ \forall \underline{y}.\, \varphi(\underline{x}, \underline{y}),$$

where $\underline{x}$ and $\underline{y}$ are tuples of variables and φ is a quantifier-free formula containing only predicate and constant symbols (or, equivalently, no function symbols of arity greater than or equal to 1). The decidability of the satisfiability problem for any BSR-theory is a well-known decidability result [8].

The following sub-class of BSR-theories can be used to specify policies as shown in, e.g., [15]. A *Datalog-theory* is a BSR-theory whose sentences are of the form

$$\forall \underline{x}, \underline{y}.\, q_1(\underline{x}, \underline{y}) \wedge \cdots \wedge q_n(\underline{x}, \underline{y}) \ \rightarrow\ p(\underline{x}) \qquad (1)$$

where p, q_i, for $i = 1, ..., n$, are predicate symbols, and $\underline{x}, \underline{y}$ are disjoint tuples of variables such that the length of $\underline{x}$ is equal to the arity of p. Usually, sentences of the form (1) are written as

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

where $\leftarrow$ can be read as the reverse of the implication connective (sometimes also the universal quantifiers will be dropped). Formulae written in this way are called *rules* in the literature, while their hypotheses and $p(\underline{x})$ are called the *body* and the *head* of the rule, respectively.

We conclude by recalling some notions that are relevant for the combination of theories that provide us with the formal tools to separately specify the WF and PM levels of an SO application and then modularly combine them. Let T_1 and T_2 be two theories; we say that they *share* the theory $T_0 = T_1 \cap T_2$ if $T_0 \neq \emptyset$ and their *combination* $T_1 \cup T_2$ is *non-disjoint*. Otherwise (i.e. when $T_0 = \emptyset$), we say that the combination $T_1 \cup T_2$ is *disjoint*. For verification, it is important to combine decision procedures for each theory T_1 and T_2 so as to obtain a decision procedure for their combination. This is crucial to derive decidability results for verification problems of SO applications as we will reduce them to satisfiability problems in the combination of the theories formalizing the WF and the PM levels. A class of theories that will be relevant in this task (see Lemma 2 below) is the following. A theory T is *stably infinite* if a T-satisfiable quantifier-free formula is satisfiable in a model of T whose domain has infinite cardinality. Examples of stably infinite theories are $MsgPass[Msg]$ of Example 1, any BSR theory (see, e.g., [19]), and many theories formalizing data structures, such as arrays or sets. Enumerated data-type theories are not stably infinite as they admit only models whose domains have finite cardinality.

B. Two-level SO transition systems

We are now ready to define an instance of the framework of Section II to formally specify SO applications designed according to the two-level architectures considered in this paper. This framework relies on the following assumptions.

Framework assumption 1: As depicted in Fig. 1, we assume that the WF and PM levels are formalized by a Σ_{WF}-theory T_{WF} and a $(\Sigma_{PM} \cup \underline{p})$-theory T_{PM}, which share a Σ_{sub}-theory T_{sub}, called the *substrate*. Formally, $\Sigma_{sub} \subseteq \Sigma_{WF}$ and $\Sigma_{sub} \subseteq \Sigma_{PM}$, and $T_{sub} \subseteq T_{WF}$ and $T_{sub} \subseteq T_{PM}$. ■

The shared theory T_{sub} plays the role of *interface* between the two levels. A minimal requirement on the interface is to provide some knowledge about the identities of the principals involved in the SO application. This is formalized as follows.

Framework assumption 2: Σ_{sub} contains the sort symbol Id. ■

This last assumption is crucial for many aspects of SO applications related to PM, such as integrity (of messages or certificates), authenticity (of certificates), and proof-of-compliance (of credentials).

Using the notion of combination of theories introduced at the end of Section III-A, we are now able to define the concept of background theory for an SO application that is obtained by modularly combining the theories formalizing the WF and the PM levels. Let T_{sub}, T_{WF}, and T_{PM} be consistent theories satisfying Framework assumptions 1 and 2. The *background Σ_{SOA}-theory* T_{SOA} is the union of the theories T_{WF} and T_{PM}, i.e. $\Sigma_{SOA} := \Sigma_{WF} \cup \Sigma_{PM}$ and $T_{SOA} := T_{WF} \cup T_{PM}$. Note that, by Robinson consistency theorem (see, e.g., [10]), T_{SOA} is consistent since both T_{WF} and T_{PM} are assumed to be so. We will sometimes refer to T_{WF} as the *WF background theory* and to T_{PM} as the *PM background theory*.

We introduce a particular class of SO transition systems (defined in Section II) by using background theories obtained by combining theories for the WF and PM levels satisfying the two framework assumptions above. A technical problem in doing this is the following. SO transition systems (in particular their states and runs) are defined with respect to a certain first-order structure. Instead, we want to use theories that, in general, identify classes of first-order structures and not just one particular structure. However, since the verification problems for SO applications considered below will be reduced

to satisfiability problems, the following notion tells us that—under suitable conditions—we can use theories in place of structures. A Σ-theory T is *adequate* for a Σ-structure $\mathcal{M}$ if $\mathcal{M}, v \models \varphi(\underline{x})$, for some valuation v mapping the variables in $\underline{x}$ to elements of the domain of $\mathcal{M}$, is equivalent to the T-satisfiability of $\varphi(\underline{x})$, for any quantifier-free formula $\varphi(\underline{x})$. For example, it is possible to see that enumerated data-type theories are adequate for any of their models (as they are all isomorphic) or that the theory $MsgPass[Msg]$ is adequate to the structure containing finite sets of messages.

As notation, let us write $\forall \underline{z}.\, \underline{p}(\underline{z}) \leftrightarrow \underline{\iota_{PM}}(\underline{i}, \underline{p}, \underline{z})$ (respectively, $\forall \underline{z}.\, \underline{p}'(\underline{z}) \leftrightarrow \underline{\varphi}(\underline{i}, \underline{p}, \underline{z})$) to abbreviate the finite conjunction, for $j = 1, ..., n$, of formulae of the form $\forall \underline{z}_j.\, p_j(\underline{z}_j) \leftrightarrow \iota_j(\underline{i}, \underline{p}, \underline{z}_j)$ (respectively, $\forall \underline{z}_j.\, p_j(\underline{z}_j) \leftrightarrow \varphi_j(\underline{i}, \underline{p}, \underline{z}_j)$) when $\underline{p} = p_1, \ldots, p_n$, $\underline{\iota_{PM}} = \iota_1, \ldots, \iota_n$ (respectively, $\underline{\varphi} = \varphi_1, \ldots, \varphi_n$), $\underline{z}_j \subseteq \underline{z}$, and the length of $\underline{z}_j$ is equal to the arity of p_j.

Definition 1 (Two-level SO transition system): Let T_{sub}, T_{WF}, and T_{PM} be consistent theories satisfying Framework assumptions 1 and 2 and $\mathcal{M}$ be a Σ_{SOA}-structure. A *two-level SO transition system (with background theory T_{SOA} adequate for $\mathcal{M}$)* is an SO transition system $(\underline{x}, \underline{p}, \iota, Tr)$ such that (a) $\underline{p} \cap \Sigma_{SOA} = \emptyset$; (b) ι is a state Σ_{SOA}-formula of the form:

$$\forall \underline{i}.\, (\iota_{WF}(\underline{i}, \underline{x}) \wedge \forall \underline{z}.\, \underline{p}(\underline{z}) \leftrightarrow \underline{\iota_{PM}}(\underline{i}, \underline{p}, \underline{z}))\,, \tag{2}$$

where $\underline{i}$ is a finite sequence of variables of sort Id, ι_{WF} is a quantifier-free $\Sigma_{WF}(\underline{i}, \underline{x})$-formula, and ι_{PM} is a quantifier-free $\Sigma_{PM}(\underline{z}, \underline{i}, \underline{p})$-formula; and (c) Tr is a finite state of transition formula of the form

$$\exists \underline{i}, \underline{d}.\, (G(\underline{i}, \underline{d}) \wedge \underline{x}' = \underline{f}(\underline{x}, \underline{i}, \underline{d}) \wedge \forall \underline{z}.\, \underline{p}'(\underline{z}) \leftrightarrow \underline{\varphi}(\underline{i}, \underline{p}, \underline{z}))\,, \tag{3}$$

called *guarded assignment transition*, where $\underline{i}$ is a tuple of variables of sort Id; $\underline{d}, \underline{z}$ are sets of variables of a sort dependent on the WF and PM levels of the application; G is a quantifier-free formula, called the *guard* of the transition; $\underline{f}$ is a tuple of $\Sigma_{WF}(\underline{x}, \underline{i})$-terms, called the *WF updates* of the transition, whose sorts are pairwise equal to those of the state variables in $\underline{x}$; and $\underline{\varphi}$ is a tuple of quantifier-free $\Sigma_{PM}(\underline{i}, \underline{p}, \underline{z})$-formulae, called the *PM updates* of the transition. ■

If $\underline{x} = \emptyset$ (recall that we have assumed that $\underline{p} \neq \emptyset$ for SO transition systems, cf. Section II), then we say that the SO application is *(purely) relational*. Intuitively, the form (2) for the initial state formula is inspired by the observation that usually the principals at the beginning of the computation have some common (or no) knowledge about the facts that are relevant to the PM level. Note that $\underline{f}$ and $\underline{\varphi}$ may not contain the state variables in $\underline{x}'$ and the state predicates in $\underline{p}'$, i.e. updates are not recursive.

Below, for simplicity, we will no more mention the Σ_{SOA}-structure $\mathcal{M}$ and implicitly assume that T_{SOA} is adequate for $\mathcal{M}$. To help intuition, we illustrate the notion of two-level SO transition system by means of a simple example (extracted from a larger one in [5]).

Example 2: Consider a situation where the clerks of an office may send messages over a network. The messages may contain, among many other things, certificates about their identities, roles, or capability to access certain resources in the organization they belong to. Certificates about the identities and roles are issued by a trusted certification authority while those about the access to a certain resource are issued by heads (who are clerks with this special right). In order to comply with the policies of accessing resources, each clerk maintains a table about his/her identity, role, and access capability as well as about other clerks. We describe a two-level SO transition system to formalize this situation.

First of all, we specify the WF background theory:

$$\begin{aligned} T_{sub} &:= EDT(\{\texttt{Ed}, \texttt{Helen}, \texttt{RegOffCA}, \texttt{Res}\}, Id) \cup \\ &\quad\; EDT(\{\texttt{employee}, \texttt{head}\}, Role) \\ T_{WF} &:= T_{sub} \cup MsgPass[Msg] \cup Msg \cup Cert \end{aligned}$$

where Ed and Helen are two clerks, RegOffCA is the trusted certification authority, Res is a shared resource (e.g., a repository), employee and head are the possible roles of clerks, and $MsgPass[Msg]$ is the theory for message passing introduced in Example 1. In particular, Msg is a theory to describe the structure of messages as follows: a message contains a field identifying the sender, a field identifying the receiver, and a field carrying their contents. Formally, this is done by introducing two new sort symbols $Body$ and the ternary function msg of sort $Id \times Body \times Id \rightarrow Msg$. Finally, $Cert$ is a theory to provide functionalities to analyze the body of messages and extract some relevant information: the predicate cert_of_role of sort $Body \times Id \times Role$ is capable of recognizing that the body of a message contains a certificate that its second argument is the identifier of a clerk whose role is that of its third argument. For example, if cert_Ed_empl is a constant of sort $Body$ representing the certificate that the employee Ed has the role employee, then the message sent by RegOffCA to Helen containing the certificate cert_Ed_empl is encoded by the following term: $\mathsf{msg}(\texttt{RegOffCA}, \mathsf{cert\_Ed\_empl}, \texttt{Helen})$ and we will also have that, e.g., $\mathsf{cert\_of\_role}(\mathsf{cert\_Ed\_empl}, \texttt{Ed}, \texttt{employee})$ holds.

The state of the two-level SO transition system specifying the situation above should contain a WF state variable net of sort $SetOfMsg$ (containing the set of messages exchanged during a run of the transition system) and a PM state variable hasrole of arity $Id \times Id \times Role$ (storing the join of the tables of each clerk about their roles). The initial state of the system should specify that no message has been exchanged over the net and that no role is known to the various clerks. This can be formalized by a state formula as follows:

$$net = \mathsf{mty} \quad \wedge \quad \forall z_1, z_2, r.\mathsf{hasrole}(z_1, z_2, r) \;\leftrightarrow\; \mathit{false}\,,$$

which is a formula of the form (2) by taking $\underline{i} = \emptyset$, $\underline{z} = \{z_1, z_2, r\}$, $net \in \underline{x}$, and $\mathsf{hasrole} \in \underline{p}$.

As an example of interplay between the WF and PM levels, Fig. 2 shows the guarded assignment transition of the form (3) that formalizes what happens when a message containing a certificate about the role of an employee (say, Ed) is sent to another employee (say, Helen) by the certification authority (RegOffCA). Note that the content of the state variable net is left unchanged by the transition, whose only effect is to

$$\exists i_1, i_2, c. \left(\begin{array}{l} \mathsf{mem}(\mathsf{msg}(\mathtt{RegOffCA}, c, i_1), net) \ \wedge \ \mathsf{cert\_of\_role}(c, i_2, \mathtt{employee}) \ \wedge \ net' = net \ \wedge \\ \forall z_1, z_2, d. \, \mathsf{hasrole}'(z_1, z_2, d) \leftrightarrow \big(\ \mathit{if} \ (z_1 = i_1 \wedge z_2 = i_2 \wedge d = \mathtt{employee}) \ \mathit{then} \ \mathit{true} \ \mathit{else} \ \mathsf{hasrole}(z_1, z_2, d) \ \big) \end{array} \right)$$

where i_1, i_2, z_1, z_2 are variables of sort Id, c is a variable of sort $Body$, and d is a variable of sort $Role$

Fig. 2. A formalization of the interplay between WF and PM levels by a guarded assignment transition (cf. Example 2)

update the access table (represented by the predicate hasrole) with the entry corresponding to the content of the received (role) certificate. For example, upon reception of the message containing the certificate cert_Ed_empl, the following fact $\mathsf{hasrole}'(\mathtt{Helen}, \mathtt{Ed}, \mathtt{employee})$ must hold in the next state, while for all the other triples, $\mathsf{hasrole}'$ has the same Boolean value of hasrole.

So far, we have specified the WF level of the SO application. As anticipated above, we can define T_{PM} to contain T_{sub} and a finite set of Datalog rules that declaratively formalize the access policy statements of the SO application. Since this way of formalizing policies has been well studied in the literature (see, e.g., [11]), as a simple example, we only give the following Datalog rule

$$\forall i. \, \mathsf{can\_access}(i, \mathtt{Res}) \quad \leftarrow \quad \mathsf{hasrole}(\mathtt{Res}, i, \mathtt{head}),$$

where the variable i is of sort Id and $\mathsf{can\_access} \in \Sigma_{PM}$. It says that the clerk i can access the shared resource Res if the latter knows (by retrieving the right entry in the table represented by hasrole) that i has the role of head. ■

In [5], we discuss in detail a generalization of the above example inspired by an industrial application.

IV. Some verification problems for SO applications

Let $\mathcal{A} = (\underline{x}, \underline{p}, \iota, Tr)$ be a two-level SO transition system with background theory T_{SOA} for an SO application; for brevity, we will sometimes refer to $\mathcal{A}$ simply as SO application. We define and investigate some verification problems for SO applications and give sufficient conditions for their decidability. In [5], we also discuss pragmatical aspects of how to implement the decision procedures.

A. Executability of SO applications

Symbolic execution is a form of execution where many possible behaviors of a system are considered simultaneously. This is achieved by using symbolic variables to represent many possible states and executions. For each possible valuation of these variables, there is a concrete system state that is being indirectly simulated. This technique is particularly useful for the design of SO applications when usually several scenarios are identified as typical execution paths that the application 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 SO application just designed allows one or many of the chosen scenarios. A valuable contribution of the proposed framework is that symbolic execution of SO applications can be done by using existing techniques for automated deduction.

In any scenario, there is only a known and finite number of principals. So, for the verification of the executability of two-level SO transition systems, we can assume that:

Verification assumption 1: $T_{sub} \supseteq EDT(\underline{c}, Id)$. ■

Since there are only finitely many principals, universal quantifiers in initial state formulae do not add to expressiveness as $\forall \underline{i}. \, \iota(\underline{i}, \underline{x}, \underline{p})$ is logically T_{SOA}-equivalent to a quantifier-free formula of the form $\bigwedge_\sigma \iota(\underline{i}\sigma, \underline{x}, \underline{p})$, where σ ranges over all possible grounding substitutions mapping the variables in $\underline{i}$ to the constants in $\underline{c}$. Thus, we can further assume that:

Verification assumption 2: Initial state formulae as well as any other state formula used to describe a state of a scenario are quantifier-free. ■

The key notion for symbolic execution in our framework is the following. Let $\varphi(\underline{p}, \underline{x})$ and $\psi(\underline{p}, \underline{x})$ two quantifier-free state Σ_{SOA}-formulae; and let $\tau(\underline{p}, \underline{x}, \underline{p}', \underline{x}') \in Tr$ be a transition formula of the form (3). We write $\{\varphi\} \ \tau \ \{\psi\}$ (in analogy with Hoare triples) to abbreviate the following formula:

$$\forall \underline{x}, \underline{x}'. \, \varphi(\underline{p}, \underline{x}) \wedge \tau(\underline{p}, \underline{x}, \underline{p}', \underline{x}') \rightarrow \psi(\underline{p}', \underline{x}'), \tag{4}$$

whose validity modulo T_{SOA} implies that *the transition τ leads $\mathcal{A}$ from a state satisfying φ to one satisfying ψ.*

Definition 2: Let $\mathcal{A} = (\underline{x}, \underline{p}, \iota, Tr)$ be a two-level SO transition system with background theory T_{SOA}, let $\tau_1, ..., \tau_n$ be a sequence of transition formulae such that $\tau_i \in Tr$, and let $\varphi_0, ..., \varphi_n$ be a sequence of quantifier-free state formulae. The *(symbolic) execution problem* consists of checking whether τ_i leads $\mathcal{A}$ from a state satisfying φ_i to a state satisfying φ_{i+1}, or, equivalently, to checking

$$T_{SOA} \quad \models \quad \{\varphi_i\} \ \tau \ \{\varphi_{i+1}\}$$

for each $i = 0, ..., n-1$. ■

Property 1: Let φ and ψ be two quantifier-free state formulae and τ be a transition. Then, it is possible to effectively compute a quantifier-free formula ϕ that is logically equivalent to the negation of $\{\varphi\} \ \tau \ \{\psi\}$ and such that $T_{SOA} \models \{\varphi\} \ \tau \ \{\psi\}$ iff ϕ is T_{SOA}-unsatisfiable. ■

That is, for quantifier-free φ and ψ, the negation of $\{\varphi\} \ \tau \ \{\psi\}$ "is" still quantifier-free.

If we are able to check the T_{SOA}-satisfiability of quantifier-free formulae, then we are also able to solve the symbolic execution problem for the two-level SO transition system with T_{SOA} as background theory. We now identify sufficient conditions on the component theories of T_{SOA} (i.e. T_{sub}, T_{WF}, and T_{PM}) for the decidability of the T_{SOA}-satisfiability problem.

Lemma 1: Let T_{sub} be an enumerated data-type theory, and $T_{WF} \supseteq T_{sub}$ and $T_{PM} \supseteq T_{sub}$ be consistent theories with decidable satisfiability problems. The T_{SOA}-satisfiability problem is decidable for $T_{SOA} = T_{WF} \cup T_{PM}$. ■

Pragmatical aspects of how to implement the decision procedures are discussed in [5].

We are now in the position to state the main result of this section, which follows from the properties and lemmas above.

Theorem 1: Let T_{sub} be an enumerated data-type theory, and $T_{WF} \supseteq T_{sub}$ and $T_{PM} \supseteq T_{sub}$ be consistent theories with decidable satisfiability problems. Then, the symbolic execution problem for two-level SO transition systems with background theory $T_{SOA} = T_{WF} \cup T_{PM}$ is decidable. ■

Note that the use of an enumerated data-type theory as T_{sub} does not imply that only two-level SO transition systems with finite state space can be verified by our method. In fact, both T_{WF} and T_{PM} can have models with infinite cardinalities (this is the case, for example, of the theory $MsgPass$). So, symbolic execution is decidable even if the state space of the two-level SO transition systems is infinite provided that there exist decision procedures for the theories characterizing the WF and the PM levels and it is possible to find a common sub-theory, used for synchronization by the two levels, whose models are finite.

B. Invariant verification of SO applications

Recall that we fixed a two-level SO transition system $\mathcal{A} = (\underline{x}, \underline{p}, \iota, Tr)$ with background theory T_{SOA}. We now consider the problem of verifying that $\mathcal{A}$ satisfies a certain security property ϕ, in symbols $\mathcal{A} \models \phi$. Since many interesting security properties can be expressed as invariance properties (e.g., for the verification of security protocols or web services), which are a sub-class of safety properties, we assume below that ϕ is a state formula of the form

$$\forall \underline{i} : Id.\, \varphi(\underline{i}, \underline{x}, \underline{p}). \tag{5}$$

Two remarks are in order. First, we are considering a sub-class of invariance properties since, in general, φ can be a past-formula (see, e.g., [14]). Second, we cannot assume that a finite and known number of principals is fixed so that (5) is equivalent to a quantifier-free formula and thus the verification techniques in Section IV-A still apply. Rather, we want to verify that for a fixed but unknown number of principals, $\mathcal{A}$ satisfies the invariance property ϕ, i.e. we want to solve a *parameterized* invariance verification problem. For this reason, in the rest of this section, we assume that:

Verification assumption 3: T_{sub} is the theory of an equivalence relation. ■

In this way, we are able to distinguish between the identifiers of the various principals. Now, in order to show that formulae of the form (5) are invariant of $\mathcal{A}$, we can use the well-known INV rule of Manna and Pnueli [14]:

$$\begin{array}{llll} (I_1) & T_{SOA} & \models & \forall \underline{i}.\iota(\underline{i}) \rightarrow \psi(\underline{i}) \\ (I_2) & T_{SOA} & \models & \forall \underline{i}.\psi(\underline{i}) \rightarrow \varphi(\underline{i}) \\ (I_3) & T_{SOA} & \models & \{\psi\}\ \tau\ \{\psi\} \text{ for each } \tau \in Tr \\ \hline & \mathcal{A} & \models & \Box\varphi \end{array}$$

The intuition underlying the correctness of the rule is the following. Assume there exists a formula ψ of the form (5) identifying a set of states that includes both the set of initial states (I_1) and the set of states characterized by φ (I_2), and, furthermore is an invariant of $\mathcal{A}$ (I_2), i.e. each transition of τ_i in Tr leads from a state satisfying ψ to a state satisfying again ψ. Then, also φ is an invariant of $\mathcal{A}$.

Using the INV rule, assuming that the invariant ψ has been guessed, it is possible to reduce the problem of verifying that a certain property is an invariant of the application, to several T_{SOA}-satisfiability problems. In fact, reasoning by contradiction we have that (I_1) and (I_2) hold iff the quantifier-free formulae

$$\iota(\underline{i}, \underline{p}, \underline{x}) \wedge \neg\psi(\underline{i}, \underline{p}, \underline{x}) \quad \text{and} \quad \psi(\underline{i}, \underline{p}, \underline{x}) \wedge \neg\varphi(\underline{i}, \underline{p}, \underline{x})$$

are T_{SOA}-unsatisfiable, respectively, where the variables in $\underline{i}$ are regarded as (Skolem) constants. Similarly, for a given τ, (I_3) holds iff the (universally) quantified formula

$$\forall \underline{j}.\, \psi(\underline{j}, \underline{p}, \underline{x}) \wedge \tau(\underline{p}, \underline{x}, \underline{p}', \underline{x}') \wedge \neg\psi(\underline{i}, \underline{p}', \underline{x}')$$

is T_{SOA}-unsatisfiable, where $\underline{x}, \underline{x}'$ and $\underline{i}$ are regarded again as (Skolem) constants, for each $\tau \in Tr$.

Property 2: Let $\psi_0(\underline{x}, \underline{p}) := \forall \underline{i}.\, \psi(\underline{i}, \underline{x}, \underline{p})$ be a state formula and $\tau(\underline{x}, \underline{p}, \underline{x}', \underline{p}')$ be a transition formula. Then, it is possible to effectively compute a formula of the form $\psi_1'(\underline{i}, \underline{x}, \underline{p}) \wedge \forall \underline{j}.\, \psi_2'(\underline{i}, \underline{x}, \underline{p})$ that is logically equivalent to the negation of $\{\psi_0\}\ \tau\ \{\psi_0\}$ and such that $T_{SOA} \models \{\psi_0\}\ \tau\ \{\psi_0\}$ iff $\psi_1'(\underline{i}, \underline{x}, \underline{p}) \wedge \forall \underline{j}.\psi_2'(\underline{i}, \underline{x}, \underline{p})$ is T_{SOA}-unsatisfiable. ■

To be able to check that $\mathcal{A} \models \Box\varphi$, we need to solve the T_{SOA}-satisfiability problem of (universally) quantified formulae. We now identify sufficient conditions on the background theory T_{SOA} for this problem to be decidable.

Lemma 2: Let Σ_{sub} contain only (countably many) constant symbols (i.e. T_{sub} is the theory of an equivalence relation), $T_{WF} \supseteq T_{sub}$ be a consistent and stably-infinite theory with decidable satisfiability problem such that the signature of no function symbol in Σ_{WF} is of the kind $S_1 \times \cdots S_n \rightarrow Id$ (for $S_i \in \Sigma_{WF}$ and $i = 1, ..., n$) and $T_{PM} \supseteq T_{sub}$ be a consistent BSR theory. Then, the T_{SOA}-satisfiability problem is decidable for formulae of the form

$$\forall \underline{i} : Id.\, \varphi(\underline{i}, \underline{x}, \underline{p}),$$

where $T_{SOA} = T_{WF} \cup T_{PM}$ and $\underline{x}, \underline{p}$ are (finite) sequences of variables and predicate symbols (such that $\underline{p} \cap \Sigma_{SOA} = \emptyset$), respectively. ■

As we have already said, there are many theories formalizing data structures (such as $MsgPass$) relevant for modeling the WF of SO applications, which are stably infinite. Also, it is frequently the case that the data structures formalized by these theories use identifiers to create new pieces of information (typical examples are the certificates of the identities or the roles of principals) but do not create new identifiers. In this way, the requirement that functions in T_{WF} do not create identifiers (syntactically, this is expressed by forbidding that the return type of the functions is not Id) is frequently satisfied. We point out that this requirement is a sufficient condition to avoid the creation of new identifiers and it may be weakened. However, we leave the study of more general

sufficient conditions to future work. Pragmatical aspects of invariant verification of SO applications are discussed in [5].

We conclude this section with the main technical result, which follows from the above properties and lemma.

Theorem 2: Let Σ_{sub} contain only (countably many) constant symbols (i.e. T_{sub} is the theory of an equivalence relation), $T_{WF} \supseteq T_{sub}$ be a consistent and stably-infinite theory with decidable satisfiability problem such that the signature of no function symbol in Σ_{WF} is of the kind $S_1 \times \cdots S_n \rightarrow Id$ (for $S_i \in \Sigma_{WF}$ and $i = 1, ..., n$), and $T_{PM} \supseteq T_{sub}$ be a consistent BSR theory. Let $\mathcal{A} = (\underline{x}, \underline{p}, \iota, Tr)$ be a two-level SO transition system with background theory $T_{SOA} = T_{WF} \cup T_{PM}$, and $\forall \underline{i}.\, \varphi(\underline{i}, \underline{x}, \underline{p})$ be a state formula. It is decidable to check whether $\mathcal{A} \models \forall \underline{i}.\, \varphi(\underline{i}, \underline{x}, \underline{p})$, provided there exists a state formula $\forall \underline{i}.\, \psi(\underline{i}, \underline{x}, \underline{p})$ such that (a) $T_{SOA} \models \forall \underline{i}.\iota(\underline{i}) \rightarrow \psi(\underline{i})$, (b) $T_{SOA} \models \forall \underline{i}.\psi(\underline{i}) \rightarrow \varphi(\underline{i})$, and (c) $T_{SOA} \models \{\psi\}\ \tau\ \{\psi\}$ for each $\tau \in Tr$. ■

Indeed, the usefulness of the theorem depends on the availability of the formula $\forall \underline{i}.\, \psi(\underline{i})$, which is called an *inductive* invariant since it is preserved under the application of the transitions of the two-level SO transition system. Since the problem of finding such a formula when $\underline{p} = \emptyset$ is undecidable (see, e.g., [14]), it is also undecidable in our case. However, several heuristics have been proposed; see, e.g., [9] for a recent proposal and pointers to the literature. An interesting line of future work is to adapt these techniques to find invariants of SO applications. Note that it is possible to dispense with the computation of the auxiliary invariant whenever $\forall \underline{i}.\, \varphi(\underline{i})$ is already inductive; in which case, conditions (a) and (b) of the theorem are trivially satisfied and we are only required to discharge proof obligation (c).

V. Related work and conclusions

We have presented a two-level formal framework that allows us to specify and verify the interplay of authorization policies and workflow in SO applications and architectures. In the previous sections, we already discussed relevant related works and also pointed out different research lines along which we are currently extending the techniques and results presented here. In particular, as we remarked, formal methods are being increasingly applied extensively to support the correct design of SO applications. These works range from extending the workflow with access control aspects (e.g., [7], [16]) to, vice versa, embedding the workflow within the access control system (e.g., [6], [12], [13], [20]), thus mainly focusing on one level at a time and abstracting away most or all of the possible interplay between the WF and PM levels. Other works (e.g. [2], [3], [4], [17]) have in contrast proposed approaches that attempt to model and analyze the interplay. We believe that our framework is abstract enough to encompass such approaches and we are currently investigating how they can be recast in our framework. In particular, we plan to use our framework to model Petri nets and access control policies as in [2] so as to perform deductive-based model checking of security-sensitive business processes, and also to formally analyze properties of RBAC by adapting the framework of [3], [4]. Finally, we also plan to extend the framework as we presented it in this paper, e.g. with interfaces more refined than the T_{sub} we considered here, so as to be able to perform modular reasoning in the assume-guarantee style.

Acknowledgments: 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 PRIN'07 project "SOFT". We thank the members of the AVANTSSAR project for useful discussions.

References

[1] A. Armando, S. Ranise, and M. Rusinowitch, "A Rewriting Approach to Satisfiability Procedures," *Info. and Comp.*, vol. 183, no. 2, pp. 140–164, 2003.

[2] A. Armando and S. E. Ponta, "Model Checking of Security-Sensitive Business Processes," 2009, submitted.

[3] P. Balbiani, Y. Chevalier, and M. El Houri, "A logical approach to dynamic role-based access control," in *AIMSA'08*, ser. LNCS 5253. Springer-Verlag, 2008.

[4] ——, "An attribute based framework to express dynamic evolution of services in a distributed environment," 2009, submitted.

[5] M. Barletta, S. Ranise, and L. Viganò, "Verifying the Interplay of Authorization Policies and Workflow in Service-Oriented Architectures (Full version)," 2009, available as arXiv:0906.4570 at http://arxiv.org/abs/0906.4570.

[6] M. Y. Becker, C. Fournet, and A. D. Gordon, "Security Policy Assertion Language (SecPAL)," http://research.microsoft.com/en-us/projects/SecPAL/.

[7] E. Bertino, J. Crampton, and F. Paci, "Access Control and Authorization Constraints for WS-BPEL," in *Proceedings of ICWS'06*. IEEE Computer Society, 2006, pp. 275–284.

[8] E. Börger, E. Grädel, and Y. Gurevich, *The Classical Decision Problem*. Springer-Verlag, 1997.

[9] A. R. Bradley and Z. Manna, "Property-Directed Incremental Invariant Generation," *Formal Aspects of Computing*, 2009, to appear.

[10] C.-C. Chang and J. H. Keisler, *Model Theory*. North-Holland, 1990.

[11] J. DeTreville, "Binder, a logic-based security language," in *IEEE Symposium on Security and Privacy*. IEEE Computer Society, 2002.

[12] D. J. Dougherty, K. Fisler, and S. Krishnamurthi, "Specifying and reasoning about dynamic access-control policies," in *IJCAR'06*, ser. LNCS 4130. Springer-Verlag, 2006, pp. 632–646.

[13] Y. Gurevich and I. Neeman, "Distributed-Knowledge Authorization Language (DKAL)," http://research.microsoft.com/~gurevich/DKAL.htm.

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

[15] J. C. M. Ninghui Li, "Understanding spki/sdsi using first-order logic," *Int. Journal of Information Security*, vol. 5, no. 1, pp. 48–64, 2006.

[16] F. Paci, E. Bertino, and J. Crampton, "An access control framework for WS-BPEL," *Int. J. Web Service Res.*, vol. 5, no. 3, pp. 20–43, 2008.

[17] 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*, vol. 2, no. 1, 2007.

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

[19] C. Tinelli and C. G. Zarba, "Combining non-stably infinite theories," *Journal of Automated Reasoning*, vol. 34, no. 3, 2005.

[20] N. Zhang, M. D. Ryan, and D. Guelev, "Evaluating access control policies through model checking," in *ISC'05*, ser. LNCS 3650. Springer-Verlag, 2005, pp. 446–460.