Consolidating the Access Control of Composite Applications and Workflows

Martin Wimmer[1], Alfons Kemper[1], Maarten Rits[2], and Volkmar Lotz[2]

[1] Technische Universität München, 85748 Garching b. München, Germany
[2] SAP Research, Font de l'Orme, 06250 Mougins, France
{wimmerma,kemper}@in.tum.de, {maarten.rits,volkmar.lotz}@sap.com

Abstract. The need for enterprise application integration projects leads to complex composite applications. For the sake of security and efficiency, consolidated access control policies for composite applications should be provided. Such a policy is based on the policies of the corresponding autonomous sub-applications and has the following properties: On the one hand, it needs to be as restrictive as possible to block requests which do not comply with the integrated sub-applications' policies. Thereby, unsuccessful executions of requests are prevented at an early stage. On the other hand, the composite policy must grant all necessary privileges in order to make the intended functionality available to legitimate users. In this paper, we present our formal model and respective algorithmic solutions for consolidating the access control of composite applications. The generated policies conform to the presented requirements of the least privileges paradigm and, thus, allow to revise and optimize the access control of composite applications. We demonstrate this by means of Web service workflows that constitute the state of the art for the realization of business processes.

1 Introduction

Composite applications are applications that rely on sub-applications (also called sub-activities) to integrate their functionality. There are numerous examples for such applications including quite simple Web applications as well as large scale enterprise resource planning systems (ERP) that rely on database backends. Also, business processes that are realized as Web service workflows constitute complex composite applications. In general, sub-applications are self-contained software components, like Web services that autonomously enforce their own security policy. When integrating autonomous sub-activities into workflows, security dependencies must be considered. As an example consider the e-health workflow illustrated in Figure 1 that will be executed when a patient is transferred to the cardiology department of a hospital. Depending on the diagnostic findings, either an in-patient treatment is applied or an electrocardiogram (ECG) is made in order to acquire further insight. Each of the individual sub-activities that are depicted in the figure are autonomously enforcing their security policies. In case, these policies are not consolidated, reliable workflow execution might be hindered. Administrative employees, for instance, are allowed to query the medical records of patients, but are not permitted to perform any medical treatment.

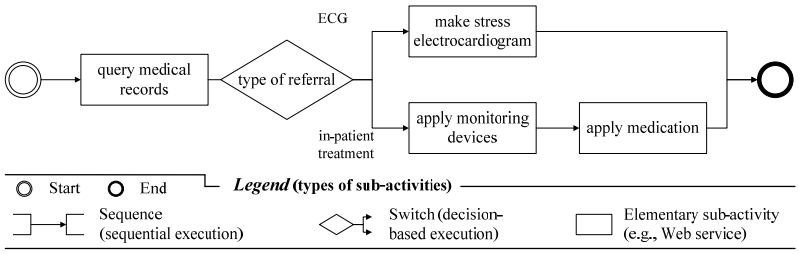

Fig. 1. Example of an e-health workflow

Thus, requests might be authorized by some sub-activities but rejected by others, which results in performance drawbacks due to unsuccessful workflow executions and can require transaction rollbacks or costly compensating actions.

In order to perform an early-filtering of the requests to avoid unsuccessful executions, a consolidated view onto the access control of workflows or general composite applications is needed. Thereby, we can identify two different perspectives onto the security configuration of a composite application. From the security officer's point of view, access control should be defined as tight as possible to avoid security vulnerabilities. Too restrictive policies on the other hand can hinder legitimate users to execute the application which contradicts the process-centered viewpoint of the application developer. Therefore, a consolidated policy is required that is tailored to the functionality of the composite application. The consolidation process derives the following information: (1) what are the least required privileges for the execution of the composite application and (2) who is granted these privileges. The first aspect allows to meet the security officer's requirements by defining access rules and role profiles that are restricted to the functionality of the composite application. The second supports the application developer in detecting unintended configurations. For instance, if only highly privileged users (e.g., administrators) are authorized to perform the workflow, this might be an indication for the design of the application itself having to be revised.

In [1] we presented a security engineering approach for optimizing the access control of Web service compositions by determining the maximum set of authorized subjects. As we will show in this paper, in order to treat generic composite applications, privilege relaxation tests are required in addition. Our contributions are a formal model and corresponding algorithmic solutions for the consolidation of the access control of generic composite applications. The consolidation is performed from the *single-user / single-role*-perspective, meaning that a user can execute the application by the activation of one task specific role. This complies with most business processes, which are typically representing job specific functions and are thus designed for specific groups of employees.

The remainder of this contribution is structured as follows: Section 2 introduces the syntax and semantics of our policy algebra, which constitutes the basis of the policy consolidation approach presented in Section 3. In Section 4 elemen-

tary algorithms of the policy consolidation process are described. Section 5 gives an overview over related work and Section 6 concludes the paper.

2 Policy Model

First, we introduce the policy algebra which constitutes the basis for the formal specification of the proposed policy consolidation technique. Policies are described in an attribute based way and are not restricted to identity based description. For instance, subjects can be specified through characterizing properties like role-membership, age, profession skills and so on. The policy model allows to express discretionary access control (DAC) rules and supports role based access control (RBAC) models which are suitable security concepts for almost all commercial applications. The formal syntax and semantics of our policy model are based on those introduced by Bonatti et al. [2]. We adapted and extended this model where necessary, e.g., by introducing additional operators.

2.1 Notation

Predicates A predicate defines an attribute comparison of the form (*attribute-identifier* $\circ$ *constant*). Depending on the attribute's domain, the comparison operator $\circ$ is in $\{<, \leq, =, \geq, >\}$ for totally ordered sets and in $\{\sqsubset, \sqsubseteq, =, \sqsupseteq, \sqsupset\}$ for partially ordered finite sets.

Subjects, objects, actions, and conditions Let *Attr* be the set of distinguished attribute identifiers. *Attr* is subdivided into disjoint sets of subject, object, action, and environment attribute identifiers (denoted as *S-Attr*, *O-Attr*, *A-Attr*, and *E-Attr* respectively). A set of subjects S is represented by a disjunction of predicate conjunctions over *S-Attr*. That is, $S = ((s_{1,1} \wedge \ldots \wedge s_{1,l}) \vee \ldots \vee (s_{k,1} \wedge \ldots \wedge s_{k,l}))$, with each $s_{i,d}$ being a predicate conjunction that applies to one attribute. The cardinality of *S-Attr* is denoted by l. The elements of *S-Attr* are also called dimensions of a subject specification. Representations of objects O and actions A are defined in a similar way. S, O, and A are inequality-free. A condition c is a boolean formula defined over attributes of *E-Attr* that can include user defined functions with Boolean codomain (e.g., *isWeekday(date)*).

Rules and policies A rule R is a quadruple (S, O, A, c), consisting of specifications of subjects S, objects O, and actions A. A rule assigns a set of permissions specified by (O, A) to a set of subjects. The scope of the assignment is restricted through c. Individual rules $R_1, \ldots, R_n$ can be aggregated in a policy $P = \{R_1, \ldots, R_n\}$.

Evaluation context An evaluation context $e \in \mathcal{E}$ is a partial mapping of the attributes defined in *Attr*. If $D_1, \ldots, D_m$ are the domains of the attributes in *Attr*, then $\mathcal{E}$ is defined as $D_1^{\perp} \times \ldots \times D_m^{\perp}$, with $D_j^{\perp} = D_j \cup \{\perp\}$ and $\perp$ representing an unspecified attribute value.

2.2 Semantics

Evaluation of rules An evaluation context e is evaluated against the individual components of rules. A subject specification S applies to e, iff S maps to *true* w.r.t. the attribute values of e. That is, $[\![S]\!]_e := S(e) = (true|false)$. The seman-

tics of O, A and c are defined analogously. The applicability of a rule R w.r.t. e is defined as $[\![R]\!]_e := [\![S]\!]_e \wedge [\![O]\!]_e \wedge [\![A]\!]_e \wedge [\![c]\!]_e$.

Evaluation of policies The semantics of a policy P depend on the employed *policy evaluation algorithm* (abbrev. *pe-alg*). We define the evaluation algorithms *pe-all* and *pe-any*, with $[\![P]\!]_e^{\textit{pe-all}} := \bigwedge_{R \in P} [\![R]\!]_e$ and $[\![P]\!]_e^{\textit{pe-any}} := \bigvee_{R \in P} [\![R]\!]_e$. *pe-all* can be applied to realize a static policy enforcement, in cases when access control can be performed once for a composite application and all its sub-activities before the execution. *pe-any* is useful for gradually performing access control, when runtime information needs to be considered. In order to characterize unrestricted specifications (i.e., tautologies) we use the symbol Υ, with $\forall e \in \mathcal{E} : [\![\Upsilon]\!]_e^{\textit{pe-alg}} = \textit{true}$.

Policy Combining Operators

Conjunction Let S and S' be two subject specifications. The conjunction of S and S' is denoted as $S \wedge S'$ with $[\![S \wedge S']\!]_e = [\![S]\!]_e \wedge [\![S']\!]_e$. The conjunction operator is analogously defined on objects, actions, conditions, and rules.

Subtraction The subtraction of two subject specifications S and S' is defined as $S - S'$ with $[\![S - S']\!]_e = [\![S]\!]_e \wedge \neg([\![S']\!]_e)$. Analogously, subtraction is also defined on objects, actions, conditions and rules.

Projection Let $R = (S, O, A, c)$ be a rule. The projection on the subjects part of R is defined as $\Pi_{\mathcal{S}}(R) = S$. Analogously, $\Pi_{\mathcal{O}}(R) = O$, $\Pi_{\mathcal{A}}(R) = A$, $\Pi_{\mathcal{C}}(R) = c$, and $\Pi_{\mathcal{O},\mathcal{A}}(R) = (O, A)$.

Let $P = \{R_1, \ldots, R_n\}$ be a policy. $\Pi_{\mathcal{S}}(P)$ is defined as $\Pi_{\mathcal{S}}(P) = \{\Pi_{\mathcal{S}}(R_1), \ldots, \Pi_{\mathcal{S}}(R_n)\}$. Other projection operators on policies are defined in a similar way. We use the abbreviation $\mathcal{S}(P) = \bigwedge_{1 \leq i \leq n} \Pi_{\mathcal{S}}(R_i)$ to denote those subjects that are granted all privileges defined in P.

Privilege, rule, and policy relaxation A privilege (O', A') relaxes a privilege (O, A), denoted as $(O, A) \sqsubseteq (O', A')$, iff it applies to more (or the same) actions on more (or the same) objects. That is, $([\![(O, A)]\!]_e = \textit{true})$ implies $([\![(O', A')]\!]_e = \textit{true})$ for any evaluation context e. Analogously, a rule R' relaxes a rule R, $R \sqsubseteq R'$, iff it grants more or the same privileges to more or the same users under the same or less restrictive conditions. That is, $\forall e \in \mathcal{E}$ with $([\![R]\!]_e = \textit{true}) \Rightarrow ([\![R']\!]_e = \textit{true})$. In the same way, $P \sqsubseteq^{\textit{pe-alg}} P'$, iff $\forall e \in \mathcal{E} : ([\![P]\!]_e^{\textit{pe-alg}} = \textit{true}) \Rightarrow ([\![P']\!]_e^{\textit{pe-alg}} = \textit{true})$.

Reduced policies In order to efficiently consolidate the policy of composite applications we are focussing on reduced policies as motivated in Section 3.1: Let the applied policy evaluation algorithm be *pe-all*. P is called *reduced*, iff

(1) $\forall R, R' \in P, R \neq R' : \nexists e \in \mathcal{E} : ([\![\Pi_{\mathcal{O},\mathcal{A}}(R) \wedge \Pi_{\mathcal{O},\mathcal{A}}(R')]\!]_e = \textit{true})$ and

(2) $\forall R \in P : \mathcal{S}(P) = \Pi_{\mathcal{S}}(R)$

A policy fulfilling (2) but not (1) can be reduced as follows: Let $R_a, R_b \in P$, $R_a \neq R_b : \exists e \in \mathcal{E} : ([\![\Pi_{\mathcal{O},\mathcal{A}}(R_a) \wedge \Pi_{\mathcal{O},\mathcal{A}}(R_b)]\!]_e = \textit{true})$. Substitute the two rules R_a, R_b through the three combined rules $R_{a-b}, R_{a \wedge b}, R_{b-a}$ with $R_{a-b} =$

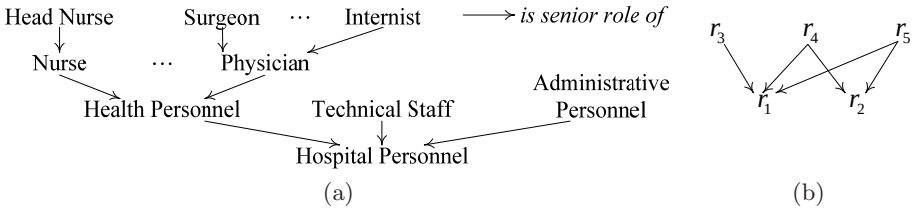

Fig. 2. Example for limited (a) and general (b) role hierarchies

$(\mathcal{S}(P), \Pi_{\mathcal{O},\mathcal{A}}(R_a) - \Pi_{\mathcal{O},\mathcal{A}}(R_b), \Pi_{\mathcal{C}}(R_a))$, $R_{a \wedge b} = (R_a \wedge R_b)$, and $R_{b-a} = (\mathcal{S}(P), \Pi_{\mathcal{O},\mathcal{A}}(R_b) - \Pi_{\mathcal{O},\mathcal{A}}(R_a), \Pi_{\mathcal{C}}(R_b))$.

2.3 Role Based Access Control

Policy administration can easily become unmanageable if privileges are independently assigned to each user. Scalability is provided through role based access control (RBAC, see [3] and [4]). Using RBAC, privileges required for performing a certain task are grouped by roles. Users acquire these privileges via the indirection of being granted those roles. Roles can be organized in a hierarchy that defines a partial order. Senior roles, which are at higher levels in the hierarchy, inherit all privileges that are granted to their junior roles. To give an example, the role *Internist* in Figure 2(a) is senior to *Physician*, denoted as *Internist* $\sqsupseteq$ *Physician*. Accordingly, *Physician* is called junior role of *Internist*.

A role r' is an *immediate descendant* of a role r if $r' \sqsubseteq r$ and there is no r'' with $r \neq r''$ and $r'' \neq r'$ such that $r' \sqsubseteq r'' \sqsubseteq r$. A role hierarchy is called *limited* if each role has at most one immediate descendant. This is the case for the example hierarchy illustrated in Figure 2(a). In contrast to this, *general role hierarchies* like the one shown in Figure 2(b) support the concept of multiple (access right) inheritance.

3 Policy Consolidation

3.1 Problem Specification

Let $\text{APP}_1, \ldots, \text{APP}_N$ be $N \geq 1$ (autonomous) sub-activities of the composite application APP_0 and P_i be the policy that applies to APP_i (for $1 \leq i \leq N$). We equate the permission to execute the i^{th} sub-activity with the set of privileges that apply to the accesses performed by APP_i (which itself can be a composite application, too). These are defined by $\Pi_{\mathcal{O},\mathcal{A}}(P_i)$. In order to enforce all of these access rights, we use *pe-all* as evaluation algorithm. We assume P_i to be a reduced policy. Thus, as defined in Section 2.2, P_i has the following two characteristics: First, the privileges defined in P_i are disjoint. This property can be assured through the preprocessing described in Section 2.2. Second, its rules apply to the same set of subjects. In some cases it might be required, that the privileges $\Pi_{\mathcal{O},\mathcal{A}}(P_i)$ are granted to different groups of users under distinguished conditions. In order to efficiently process the set of constraints, the policy is decomposed into policies conforming to Section 2.2 that are evaluated individually.

Let P_0 be the reduced policy for APP_0. In many cases, there might be no predefined policy for APP_0, i.e., P_0 is equivalent to Υ. Nevertheless, policy proto-

types can be pre-defined specifying intended configurations. The objective of the policy consolidation process is to evaluate P_0 against the policies of the underlying applications. Its result is an optimized policy P^{opt} that fulfills the following two criteria:

LP Least privilege criterion: Each privilege defined in P^{opt} must also be defined in at least one policy P_i with $1 \leq i \leq N$. The privileges defined in P^{opt} must be sufficient to perform APP_0 and its respective sub-activities.

MS Maximum set of subjects criterion: Each subject that is authorized based on the original policy configurations $(P_i)_{0 \leq i \leq N}$ must also be authorized by P^{opt}. Each subject that is defined in P^{opt} must also be defined in at least one policy P_i with $1 \leq i \leq N$ and in P_0.

3.2 Workflow Dependencies

Sub-activities of a composite application can for example be executed in sequence or in parallel. Also, iterations (i.e., loops) are possible. From an access control point of view it is of importance that **all** sub-activities will be performed. We represent this fact through the SEQUENCE pattern. Furthermore, conditional and event based executions can be defined. From the access control perspective this denotes that only **one** sub-activity will be invoked, which we represent through the so-called SWITCH template. SEQUENCE and SWITCH templates can be nested to model complex workflows. Apart from these kinds of control flow dependencies further interdependencies influencing access control can exist:

a) *Data-flow dependencies* are given, if an output parameter x of a sub-activity APP_i is input to APP_j and the value of x determines the result of the evaluation of the policy P_j.
b) *External dependencies* are dependencies by parameters external to the system, like time. For example, P_i and P_j might define time constraints that restrict the execution of APP_i and APP_j to disjoint time frames. That is, the conjunction of conditions defined in P_i and P_j respectively constitute a contradiction. Nevertheless, the control-flow can be consistent due to the execution order (e.g., think of delays during long-running transactions).

We first describe the consolidation of access control policies for the two patterns SEQUENCE and SWITCH before we return to discuss the influence of interdependencies a) and b).

3.3 Analysis of Sequence Patterns

For a SEQUENCE pattern to be consistent from the access control perspective, the following two conditions must be met: First, the access rights defined in P_0 must include those privileges defined in the policies $(P_i)_{1 \leq i \leq N}$. Second, there must be at least one subject that is granted these privileges. Otherwise, the access specifications are conflicting, preventing the execution of APP_0. Formally:

$$\forall 1 \leq i \leq N : \forall R \in P_i : \exists R' \in P_0 : \Pi_{\mathcal{O},\mathcal{A}}(R) \sqsubseteq \Pi_{\mathcal{O},\mathcal{A}}(R') \tag{1}$$

$$\exists e \in \mathcal{E} : [\![S_{\text{all}}]\!]_e = \mathit{true} \;\text{ for }\; S_{\text{all}} = \bigwedge\nolimits_{0 \leq i \leq N} \mathcal{S}(P_i) \tag{2}$$

The consolidated policy $P^{\text{opt}}_{(\text{all})}$ is defined as:

$$P^{\text{opt}}_{(\text{all})} = \{(S_{\text{all}}, \Pi_{\mathcal{O},\mathcal{A}}(R), (\Pi_{\mathcal{C}}(R) \wedge \Pi_{\mathcal{C}}(R'))) \mid \forall i \in \{1,\ldots,N\} : R \in P_i, R' \in P_0 : \Pi_{\mathcal{O},\mathcal{A}}(R) \sqsubseteq \Pi_{\mathcal{O},\mathcal{A}}(R')\} \tag{3}$$

The applied evaluation algorithm is *pe-all*. If the policies $(P_i)_{1\leq i\leq N}$ fulfill LP, then LP can also be inferred for $P^{\text{opt}}_{(\text{all})}$, as the privileges in $P^{\text{opt}}_{(\text{all})}$ are restricted to those defined in $(P_i)_{1\leq i\leq N}$ and its rules are constrained through conjunctions of the respective conditions defined in these policies and P_0. Sub-activities can perform similar accesses on the same objects, like scans of the same tables of a database. Thus, $P^{\text{opt}}_{(\text{all})}$ – which aggregates the privileges defined in $(P_i)_{1\leq i\leq N}$ – might contain redundancies that can be eliminated according to Section 2.2.

3.4 Analysis of Switch Patterns

The access control configurations for SWITCH patterns can be defined from two different perspectives. The *full-authorization* approach enforces each subject defined in the consolidated policy to be authorized for any of the $(\text{APP}_i)_{1\leq i\leq N}$, irrespective which sub-activity will actually be executed. As a consequence, the consolidated policy corresponds to $P^{\text{opt}}_{(\text{all})}$ defined in the previous paragraph.

On the opposite side, *partial-authorization* distinguishes the different execution paths. Subjects might be authorized to execute APP_0 in case a particular APP_i is invoked next, but will be blocked in any other case. Thus, in order to efficiently evaluate a SWITCH pattern the distinguished execution branches have to be analyzed separately. Consequently, up to N security configurations have to be considered. In order to specify the optimized policy for the i^{th} branch, the policies P_0 and P_i are consolidated and the following must be true:

$$\forall R \in P_i : \exists R' \in P_0 : \Pi_{\mathcal{O},\mathcal{A}}(R) \sqsubseteq \Pi_{\mathcal{O},\mathcal{A}}(R') \tag{4}$$

$$\exists e \in \mathcal{E} : [\![S^{(i)}]\!]_e = \mathit{true} \;\text{ for }\; S^{(i)} = \mathcal{S}(P_0) \wedge \mathcal{S}(P_i) \tag{5}$$

The consolidated policy for the i^{th} branch (using *pe-all*) is defined as:

$$P^{\text{opt}}_{(i)} = \{(S^{(i)}, \Pi_{\mathcal{O},\mathcal{A}}(R), (\Pi_{\mathcal{C}}(R) \wedge \Pi_{\mathcal{C}}(R'))) \mid R \in P_i, R' \in P_0 : \Pi_{\mathcal{O},\mathcal{A}}(R) \sqsubseteq \Pi_{\mathcal{O},\mathcal{A}}(R')\} \tag{6}$$

3.5 The Benefits of Policy Consolidation

The policy consolidation technique performs a static analysis of the policy of a composite application APP_0 by comparing it with the security configuration of its underlying sub-activities $\text{APP}_1, \ldots, \text{APP}_N$. Its result is an optimized policy P^{opt} for APP_0 and all sub-activities ($P^{\text{opt}} = P^{\text{opt}}_{(\text{all})}$) or specific branches $\text{APP}_0 \rightarrow \text{APP}_i$ ($P^{\text{opt}} = P^{\text{opt}}_{(i)}$), respectively. In case no external or dataflow dependencies exist, the access control costs can be reduced significantly. As each execution which is granted based on P^{opt} will also be granted by the sub-activities, it is sufficient to enforce access control solely at APP_0, thus, saving redundant enforcements through the sub-activities. In case interdependencies 3.2.a) or 3.2.b) have to

be considered at runtime, no single point of access control can be established. Nevertheless, the static analysis allows to receive a consolidated view onto the set of authorized users (MS) and the least required privileges (LP) enabling the following optimizations:

Evaluation of MS: $\mathcal{S}(P^{\text{opt}})$ specifies those subjects that are authorized to execute the workflow (branch) or general composite application, respectively. It allows application developers to check more easily whether the policy complies with the intended security specifications, e.g., detecting over-privileged users or conflicts. Furthermore, in case role based access control is employed, *least required roles* can be inferred. In this regard, a least required role is a role that grants process execution without demanding for further intermediary role activations. This "one role will do"-approach is especially relevant for business processes that are typically defined for specific job functions. Least required roles are identified through the predicate reduction introduced in Alg. 1 and are unique for limited role hierarchies but not necessarily for general role hierarchies. For example, the infima of the role hierarchy shown in Figure 2(b) are r_1 and r_2. The respective least required roles are least common senior roles, i.e., r_4 and r_5 in the example.

Evaluation of LP: $\Pi_{\mathcal{O},\mathcal{A}}(P^{\text{opt}})$ represents the aggregated set of privileges tailored to the access requirements of the composite application. In the meaning of a reverse security engineering, this information allows to generate *task specific roles* which are appropriate for the application. They are called *task specific* as they group exactly those rights that are required for the composite application's functionality (while *least required* roles can be more generic).

3.6 Case Study: Web Service Workflows

Sub-activities of the intra-organizational workflow illustrated in Figure 1 on the one hand represent practical activities that require human interaction like a medication. On the other hand, they stand for information processing tasks, like an update of the stock of pharmaceuticals in the database. In the following we concentrate on the technical aspects of the workflow and assume the sub-activities to be realized as Web services with the following access rules:

- Health personnel with permanent employment and administrative personnel are allowed to access the medical records of patients. The subject specification S_{MR} applying to the sub-activity *query medical records* is defined as

$$S_{\text{MR}} = ((\textit{role} \sqsupseteq \text{Health Pers.} \wedge \textit{employment} = \text{permanent}) \vee (\textit{role} \sqsupseteq \text{Admin. Pers.}))$$

- Nurses of the cardiology and internists are allowed to update medical records, e.g., by inserting ECG results. Users allowed to execute *make stress electrocardiogram* are in
  $S_{\text{ECG}} = ((\textit{role} \sqsupseteq \text{Nurse} \wedge \textit{field-of-activity} = \text{cardiology}) \vee (\textit{role} \sqsupseteq \text{Internist}))$
- Internists are allowed to perform the sub-activity *apply monitoring devices*: $S_{\text{App}} = (\textit{role} \sqsupseteq \text{Internist})$.
- The sub-activity *apply medication* can be performed by nurses and physicians:
  $S_{\text{Med}} = ((\textit{role} \sqsupseteq \text{Nurse}) \vee (\textit{role} \sqsupseteq \text{Physician}))$.

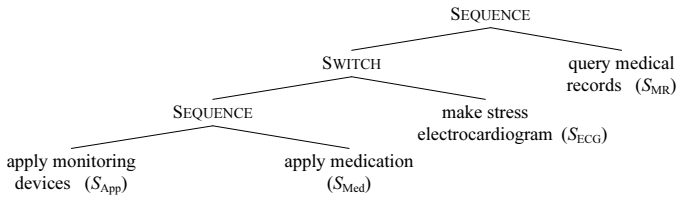

Fig. 3. Tree representation of the e-health business process

As motivated in Section 3.2, the workflow can be modeled as a composition of SEQUENCE and SWITCH patterns. This allows the authorization dependencies of the workflow to be represented as a tree as shown in Figure 3. Through a bottom-up analysis, the consolidated access control configuration for the workflow can be inferred. Typically, no access control policies are defined for the control structures – i.e, $\mathcal{P}_0 \equiv \Upsilon$ for SEQUENCE and SWITCH nodes. Hence, the privileges defined for the individual Web services, are iteratively aggregated without demanding for relaxation tests. We thus focus on determining the set of authorized users which proceeds as follows: First, S_{App} and S_{Med} are intersected as both sub-activities are linked in sequence. It holds $S_{\text{App}} \wedge S_{\text{Med}} = (\mathit{role} \sqsupseteq \text{Internist})$. Next, the SWITCH node is evaluated. The subjects that are granted full-authorization are defined by $S = (\mathit{role} \sqsupseteq \text{Internist})$. In contrast to this, nurses are only granted partial authorization for the *ECG*-branch: $S' = (\mathit{role} \sqsupseteq \text{Nurse} \wedge \mathit{field\text{-}of\text{-}activity} = \text{cardiology})$. Finally, S and S' have to be intersected with S_{MR}. We receive:

$$\begin{aligned} S_{\text{all}} &= (S_{\text{MR}} \wedge S) = (\mathit{role} \sqsupseteq \text{Internist} \wedge \mathit{employment} = \text{permanent}) \\ S^{(ECG)} &= (S_{\text{MR}} \wedge S') = (\mathit{role} \sqsupseteq \text{Nurse} \wedge \mathit{field\text{-}of\text{-}activity} = \text{cardiology} \\ &\qquad \wedge \mathit{employment} = \text{permanent}) \end{aligned}$$

Thus, the workflow is executable for nurses and internists, whereby nurses are only granted partial authorization. This allows the following optimization of access control at the workflow layer: Internists that fulfill the specification of S_{all} need only be authorized at the workflow layer. For nurses, access control has to be performed twice: On top of the workflow layer and when entering the SWITCH part. All other subjects, like those granted the *Administrative Personnel* role, can be blocked right from the beginning, as they will never succeed in reaching an end state of the workflow. The optimization capabilities can be realized to the maximum extent possible, if the access control of the sub-activities can be controlled by the composite application, for instance by building up a security context between the workflow execution system and the autonomous services, e.g., employing WS-SecureConversation [5]. In any case, policy enforcement at the workflow layer helps to reduce unnecessary service executions, transaction rollbacks and compensating actions.

4 Algorithmic Solutions

For an implementation of the described policy consolidation technique, algorithmic solutions for the evaluation of predicate conjunctions and subtractions, and the validation of privilege relaxation are required.

4.1 Implementing the Conjunction Operator

Equations (2) and (5) introduce S_{all} and $S^{(i)}$ as conjunctions of subject specifications. The conjunction operator is semantically equivalent to the set theoretical intersection operator. That is, S_{all} and $S^{(i)}$ can be interpreted as the intersection of subject sets. Let S and S' be two subject specifications. According to the policy model, S and S' are represented via disjunctions of predicate conjunctions over attributes in *S-Attr*:

$$S = s_1 \vee \ldots \vee s_k = (s_{1,1} \wedge \ldots \wedge s_{1,l}) \vee \ldots \vee (s_{k,1} \wedge \ldots \wedge s_{k,l}) \quad \text{and}$$
$$S' = s'_1 \vee \ldots \vee s'_{k'} = (s'_{1,1} \wedge \ldots \wedge s'_{1,l}) \vee \ldots \vee (s'_{k',1} \wedge \ldots \wedge s'_{k',l})$$

The attributes in *S-Attr* are also called the dimensions of subject specifications. We assume all dimensions in S and S' to be specified. If a conjunction s_i is not constrained in dimension d, then the respective predicate $s_{i,d}$ represents the whole domain of d. According to Section 2.2 the intersection of S and S' is: $S \wedge S' = \bigvee_{1 \leq i \leq k, 1 \leq j \leq k'} (\bigwedge_{1 \leq d \leq l} (s_{i,d} \wedge s'_{j,d}))$. Nevertheless, conjunctions $(s_{i,d} \wedge s'_{j,d})$ can be contradictory, i.e., unsatisfiable by any evaluation context. Such terms constitute unnecessary parts of a policy and shall be omitted to keep policy specifications clear. Alg. 1 illustrates an approach for computing a condensed representation of $S \wedge S'$. We illustrate the algorithm by means of an example. Consider the following two subject descriptions (based on the example role hierarchy shown in Figure 2(a)):

$$S = (s_1) = (\mathit{role} \sqsupseteq \text{Nurse} \wedge \mathit{yop} \geq 1) \quad \text{and}$$
$$S' = (s'_1 \vee s'_2) = (\mathit{role} \sqsupseteq \text{Admin. Pers.} \wedge \mathit{yop} \geq 0) \vee$$
$$(\mathit{role} \sqsupset \text{Health Pers.} \wedge \mathit{yop} \geq 2 \wedge \mathit{yop} \leq 4)$$

S represents all subjects that are granted the *Nurse* role and that have at least one year of practice (abbrev. *yop*). S' represents administrative employees and all subjects that are granted senior roles of the *Health Personnel* role with at least two and at most four years of practice. Thus, the dimensions are *role* and *yop*. While the domain of *role* is a finite lattice (defined by the role hierarchy shown in Figure 2(a)), the domain of *yop* is $[0, +\infty[$.

The terms s_1 and s'_1 are disjoint, because they do not overlap in the *role*-dimension, i.e., $(s_{1,\text{role}} \wedge s'_{1,\text{role}})$ is a contradiction and needs not be considered in line 7. In contrast to this, s_1 and s'_2 overlap in each dimension. The conjunction $(\mathit{yop} \geq 1) \wedge (\mathit{yop} \geq 2 \wedge \mathit{yop} \leq 4)$ is reduced to $(\mathit{yop} \geq 2 \wedge \mathit{yop} \leq 4)$. The predicates $s_{1,\text{role}}$ and $s'_{2,\text{role}}$ define the two finite sets $\Phi_1 = \{\mathit{Nurse}, \mathit{Head\ Nurse}\}$ and $\Phi'_2 = \{\mathit{Nurse}, \mathit{Head\ Nurse}, \mathit{Physician}, \mathit{Internist}, \mathit{Surgeon}\}$. Thus, $(s_{1,\text{role}} \wedge s'_{2,\text{role}})$ is equivalent to $\Phi_1 \cap \Phi'_2 = \{\mathit{Nurse}, \mathit{Head\ Nurse}\}$. The intersection can be represented through the predicate $(\mathit{role} \sqsupseteq \text{Nurse})$, as *Nurse* is the infimum of $\Phi_1 \cap \Phi'_2$ according to the example role hierachy. Thus, $S_1 \wedge S_2 = (\mathit{role} \sqsupseteq \text{Nurse} \wedge \mathit{yop} \geq 2 \wedge \mathit{yop} \leq 4)$. That is, the intersection consists of those subjects that are granted the *Nurse* role and that have at least two and at most four years of practice.

Algorithm 1 ***intersect***(S, S'), with $S \equiv s_1 \vee \ldots \vee s_k$, and $S' \equiv s'_1 \vee \ldots \vee s'_{k'}$

```
1: Ψ = false
2: for all conjunctions s_i of S do
3:    for all conjunctions s'_j of S' do
4:       for all dimensions d = 1...l do
5:          ψ_d = reduce(s_{i,d} ∧ s'_{j,d})
6:       end for
7:       Ψ = Ψ ∨ (ψ_1 ∧ ... ∧ ψ_l)
8:    end for
9: end for
10: return Ψ
```

$$\Psi \equiv S \wedge S' = \bigvee_{1 \leq i \leq k, 1 \leq j \leq k'} \left(\bigwedge_{1 \leq d \leq l} \left(s_{i,d} \wedge s'_{j,d} \right) \right)$$

4.2 Checking Privilege Relaxation

Let (O, A) and (O', A') be two privileges. As objects and actions are defined on disjoint sets of attribute identifiers (*O-Attr* and *A-Attr*, see Section 2.1) and according to the definition of privilege relaxation (Section 2.2), (O', A') relaxes (O, A), if the following holds: $\forall e \in \mathcal{E} : (\llbracket O \rrbracket_e = \mathit{true} \wedge \llbracket A \rrbracket_e = \mathit{true}) \Rightarrow (\llbracket O' \rrbracket_e = \mathit{true} \wedge \llbracket A' \rrbracket_e = \mathit{true})$. Therefore, the privilege relaxation problem can be reduced to the implication problem: Let $T = (t_1 \vee \ldots \vee t_k)$ and $T' = (t'_1 \vee \ldots \vee t'_{k'})$ be disjunctions of predicate conjunctions. T implies T', denoted as $T \Rightarrow T'$, if and only if every evaluation context which is satisfying T is also satisfying T' [6]. Informally, $T \Rightarrow T'$ means that T' is more generic than T. To evaluate whether $T \Rightarrow T'$ holds, each predicate conjunction t_i of T is evaluated against the predicate conjunctions t'_j of T'. The following three cases can arrive:

1. t_i implies t'_j, i.e., $t_i \Rightarrow t'_j$. Then a match for t_i has been found.
2. t_i and t'_j are incomparable, i.e., $(t_i \wedge \neg t'_j) = t_i$. Then t_i has to be compared with the remaining predicate conjunctions of T' to find possible matches.
3. t_i and t'_j overlap partially. Then, the remainder $(t_i \wedge \neg t'_j)$ is separately compared with the predicate conjunctions of T'.

Alg. 2 shows a pseudo-code implementation of *implies* for evaluating predicate implications. T implies T', if all predicate conjunctions t_i of T are subsumed by T'. In this case the remainder Δ is equal to *false*. In line 6 the sub-procedure *subtract* is invoked which calculates the remainder of t_i w.r.t. t'_1, i.e., $\delta = (t_i \wedge \neg t'_1)$, given in disjunctive normal form (DNF). The individual predicate conjunctions of δ are separately compared to the remaining conjunctions of T' through a recursive invocation of *implies* in line 8 of Alg. 2.

A pseudo-code implementation of *subtract* is depicted in Alg. 3. Computing the predicate subtraction is done in a way similar to Alg. 1 by iteratively comparing the conjunctive terms t_i and t'_j in each dimension d (line 2–11). If t_i and t'_j do not overlap in any dimension d, t_i and t'_j represent disjoint data sets and the remainder is t_i. Otherwise, the overall overlap of t_i and t'_j is iteratively computed and stored in the variable *work*. The non-matching parts are described by δ.

Algorithm 2 ***implies***(T, T'), with $T = t_1 \vee \ldots \vee t_k$ and $T' = t'_1 \vee \ldots \vee t'_{k'}$

1: **if** $k' = 0$ **then**
2: **return** T // *i.e.,* $T' = false$
3: **end if**
4: $\Delta = \text{false}$
5: **for all** conjunctive terms t_i of $T = (t_1 \vee \ldots \vee t_k)$ **do**
6: $\delta = \text{subtract}(t_i, t'_1)$
7: **if** $\delta \neq \text{false}$ **then**
8: $\Delta = \Delta \vee \text{implies}(\delta, t'_2 \vee \ldots \vee t'_{k'})$
9: **end if**
10: **end for**
11: **return** Δ

$(T \Rightarrow T') \Leftrightarrow (\Delta = false)$

Algorithm 3 ***subtract***(t_i, t'_j), with $t_i = t_{i,1} \wedge \ldots \wedge t_{i,l}$ and $t'_j = t_{j,1} \wedge \ldots \wedge t_{j,l}$

1: $\delta = \text{false}, \text{work} = t_i$ // *work* $= w_1 \wedge \ldots \wedge w_l$
2: **for** $d = 1 \ldots l$ **do**
3: $w'_d = (t_{i,d} \wedge t'_{j,d})$ // *the overlap of* $t_{i,d}$ *and* $t'_{j,d}$
4: $\text{work} = (w'_1 \wedge \ldots \wedge w'_{d-1} \wedge w'_d \wedge w_{d+1} \ldots \wedge w_l)$
5: **if** $w'_d \equiv \text{false}$ **then**
6: **return** t_i // t_i *and* t'_j *represent disjoint data sets*
7: **else if** $w'_d \neq t_{i,d}$ **then**
8: $\omega = t_{i,d} \wedge \neg t'_{j,d}$ // *the remainder of* $t_{i,d}$ *minus* $t'_{j,d}$
9: $\delta = \delta \vee (w'_1 \wedge \ldots \wedge w'_{d-1} \wedge \omega \wedge w_{d+1} \ldots \wedge w_l)$
10: **end if**
11: **end for**
12: **return** DNF of δ // ω *in line 8 is a predicate disjunction*

As an example assume a relational database with the table *Employees* with the attributes *Name*, *Gender*, *Salary*, and *Job* (abbrev. *na*, *ge*, *sa*, and *jo*). Parameter values of *jo* are health, administrative, and technical personnel (for short *HP*, *AP*, and *TP*). Two privileges are defined on this relation. The first privilege states that the complete table can be accessed via *select*. The second restricts the *select*-access to the data of female health care employees that earn more than $50'$ \$ and less than $100'$ \$. We use the symbol $\perp$ to represent unrestricted attribute values. The object specifications of both privileges are represented by the following two predicate conjunctions:

$$t = (na = \perp \wedge\ ge = \perp \wedge\ sa = \perp \wedge\ jo = \perp)$$
$$t' = (na = \perp \wedge\ ge = female \wedge\ sa > 50' \wedge\ sa < 100' \wedge\ jo = HP)$$

It can easily be verified that t relaxes t'. Let's assume that on the other way round it shall be examined whether t' relaxes t, which is obviously not the case. The following table shows the evaluation steps of *subtract*, in case the attributes are processed in the order *Name*, *Gender*, *Salary*, and *Job*.

	Variable *work*	**Remainders**
1.	$(na = \perp \wedge ge = \perp \wedge sa = \perp \wedge jo = \perp)$	—
2.	$(na = \perp \wedge ge = female \wedge\ sa = \perp \wedge jo = \perp)$	$\delta_1 = (na = \perp \wedge ge = male \wedge\ sa = \perp \wedge jo = \perp)$
3.	$(na = \perp \wedge ge = female \wedge$ $sa > 50' \wedge\ sa < 100' \wedge\ jo = \perp)$	$\delta_2 = (na = \perp \wedge ge = female \wedge\ sa \leq 50' \wedge\ jo = \perp)$ $\delta_3 = (na = \perp \wedge ge = female \wedge\ sa \geq 100' \wedge\ jo = \perp)$
4.	$(na = \perp \wedge ge = female \wedge$ $sa > 50' \wedge\ sa < 100' \wedge\ jo = HP)$	$\delta_4 = (na = \perp \wedge ge = female \wedge\ sa > 50' \wedge$ $sa < 100' \wedge jo \in \{TP, AP\})$

When comparing the terms in the *Salary*-dimension t divides *work* into three components, the overlapping part and two remainder predicates δ_2 and δ_3. This is the maximum number of remainder predicates that can be generated in one step if the attribute's domain is a totally ordered uncountable set (the domain of *Salary* is $[0, +\infty[$). Things are different if the attribute's domain is a partially ordered finite set, as is the case for the dimension *Job*. Instead of enumerating all attribute values (AP and TP) in distinct predicates, the internal representation is an aggregate of the form ($jo \in \{AP, TP\}$) as illustrated in the table.

As a consequence, a comparison of two predicate conjunctions results in up to $2l$ remainder predicate conjunctions in the worst case. As each of these are individually compared with T' (line 8 of Alg. 2) this leads to an exponential worst case complexity of *implies* w.r.t. the input parameter k'. Thus, the described privilege implication problem is closely related to other well known computationally hard issues like query subsumption or the satisfiability problem [6]. Nevertheless, the worst case is supposed to arrive rarely. This is due to the fact that, for the worst case to occur, privileges have to be described through distinguished, partially overlapping predicate conjunctions – which would be the case if policies are written in a complex (unstructured and almost unmanageable) way. Instead, average complexity is assumed to be close to the best case complexity, which is in polynomial time.

4.3 Implementing the Subtraction Operator

The semantics of the subtraction of two terms T and T' are defined as $[\![T - T']\!]_e = [\![T]\!]_e \wedge \neg([\![T']\!]_e)$. Thus, the subtraction operator can be realized through the already presented algorithm *implies* (Alg. 2), as the remainder Δ of *implies*(T, T') is equivalent to $T - T'$.

5 Related Literature

In Section 2 we defined the policy model that constitutes the basis for the specification of our proposed policy consolidation technique. Syntax and semantics of this policy model are closely related to those proposed by Bonatti et al. [2] and Wijesekera and Jajodia [7]. We extended them through additional operators and relaxation rules for defining policy consolidation. The access control policy of a composite application is composed of rules that codify the individual access rights that relate to the underlying sub-activities. The enforcement of such a policy depends on the applied evaluation algorithm. If negative or mixed authorization should be employed, which could be expressed in our model as well by means of the subtraction operator, conflict resolution techniques like those proposed by Jajodia et al. [8] have to be employed. In this work we focussed on positive authorization which is suitable for almost all enterprise applications.

Our work is also related to research on models for the specification and analysis of workflow processes. Adam et al. [9] use Petri-nets to model and evaluate control flow dependencies. Bettini et al. [10] identify temporal constraints that might cause inconsistencies which restrict the executability of workflows. Temporal constraints must also be considered when interpreting the result of the static policy analysis (see 3.2.b)). Nevertheless, even if dynamic dependencies have to be evaluated at runtime, policy consolidation still offers optimization potential. The composed policy allows to perform access control at the workflow layer, filtering unsuccessful execution attempts as early as possible. The enforcement of access rules at the workflow layer is also proposed by Gudes et al. [11]. Access control models and architectures for workflow systems are for example proposed by [12, 13] and [14]. Atluri et al. [13] present an approach for analyzing dependencies between sub-activities that operate on the same data but are assigned to different security levels. A framework supporting static and dynamic separation of duties is provided by Bertino et al. [14]. In contrast to this, we concentrate on *single-user / single-role* execution schemes which we assume to be prevalent for most enterprise applications. Identifying the set of authorized users of composite applications was also addressed in previous work [15]. There, a practical approach was shown, how task-specific user profiles and roles can be determined by analyzing the source code. In this paper we presented a generic, implementation independent policy consolidation framework which also supports the reverse engineering of appropriate user / role profiles by determining the least required privileges. Compliance with LP can be inferred if the policies of the underlying sub-activities have been preprocessed and minimized. How this can be achieved for Web services that rely on database interaction has been shown in foregoing work [16].

6 Conclusions and Ongoing Work

As motivated in the beginning of this paper, introducing access control on the layer of composite applications that depend on autonomous sub-applications can be employed to filter unsuccessful execution attempts as early as possible, thus, avoiding unnecessary work. Additionally, a consolidated view onto the access control of a composite application allows to revise the security configuration by restricting it to the applications' functionality. Further optimization potential is given, in case the access control of the underlying sub-activities can be regulated through the composite application.

Based on the formal specification of policy consolidation, a prototype has been implemented to show its feasibility regarding the consolidation of the access control of Web service workflows. We employ XACML as policy language which due to the attribute based description of access rules is well suited to map our policy model. Workflows are specified using BPEL4WS. Currently, we are working on the integration of the consolidation functionality in a business process modeling tool [17] and the extension of the prototype to support all kinds of policy consolidations as described in Section 3 of this paper.

References

1. M. Wimmer, M.-C. Albutiu, and A. Kemper, "Optimized Workflow Authorization in Service Oriented Architectures," in *Proceedings of the International Conference on Emerging Trends in Information and Communication Security (ETRICS)*, vol. 3995 of *LNCS*, (Freiburg, Germany), pp. 30–44, June 2006.
2. P. Bonatti, S. De Capitani di Vimercati, and P. Samarati, "An Algebra for Composing Access Control Policies," *ACM Transactions on Information and System Security (TISSEC)*, vol. 5, no. 1, pp. 1–35, 2002.
3. R. S. Sandhu, E. J. Coyne, H. L. Feinstein, and C. E. Youman, "Role-Based Access Control Models," *IEEE Computer*, vol. 29, no. 2, pp. 38–47, 1996.
4. ANSI INCITS 359-2004, *Role Based Access Control.* American National Standards Institute, Inc. (ANSI), New York, NY, USA, Feb. 2004.
5. A. Nadalin *et al.*, "Web Services Secure Conversation Language (WS-SecureConversation)." `http://www-128.ibm.com/developerworks/library/specification/ws-secon/`, Feb. 2005.
6. S. Guo, W. Sun, and M. A. Weiss, "Solving Satisfiability and Implication Problems in Database Systems," *ACM Trans. Database Syst.*, vol. 21, no. 2, pp. 270–293, 1996.
7. D. Wijesekera and S. Jajodia, "A Propositional Policy Algebra for Access Control," *ACM Transactions on Information and System Security (TISSEC)*, vol. 6, no. 2, pp. 286–325, 2003.
8. S. Jajodia, P. Samarati, M. L. Sapino, and V. S. Subrahmanian, "Flexible Support for Multiple Access Control Policies," *ACM Transactions on Information and System Security (TISSEC)*, vol. 26, no. 2, pp. 214–260, 2001.
9. N. R. Adam, V. Atluri, and W.-K. Huang, "Modeling and Analysis of Workflows Using Petri Nets," *Journal of Intell. Inf. Syst.*, vol. 10, no. 2, pp. 131–158, 1998.
10. C. Bettini, X. S. Wang, and S. Jajodia, "Temporal Reasoning in Workflow Systems," *Distrib. Parallel Databases*, vol. 11, no. 3, pp. 269–306, 2002.
11. E. Gudes, M. S. Olivier, and R. P. van de Riet, "Modelling, Specifying and Implementing Workflow Security in Cyberspace," *Journal of Computer Security*, vol. 7, no. 4, pp. 287–315, 1999.
12. W.-K. Huang and V. Atluri, "SecureFlow: a Secure Web-enabled Workflow Management System," in *RBAC '99: Proceedings of the 4th ACM Workshop on Role-based Access Control*, (New York, NY, USA), pp. 83–94, ACM Press, 1999.
13. V. Atluri, W.-K. Huang, and E. Bertino, "A Semantic-Based Execution Model for Multilevel Secure Workflows," *Journal of Computer Security*, vol. 8, no. 1, 2000.
14. E. Bertino, E. Ferrari, and V. Atluri, "The Specification and Enforcement of Authorization Constraints in Workflow Management Systems," *ACM Trans. Inf. Syst. Secur.*, vol. 2, pp. 65–104, Feb. 1999.
15. M. Rits, B. D. Boe, and A. Schaad, "Xact: a Bridge between Resource Management and Access Control in Multi-layered Applications," in *SESS '05: Proceedings of the 2005 Workshop on Software Engineering for Secure Systems*, (New York, NY, USA), pp. 1–7, ACM Press, 2005.
16. M. Wimmer, D. Eberhardt, P. Ehrnlechner, and A. Kemper, "Reliable and Adaptable Security Engineering for Database-Web Services," in *Proceedings of the Fourth International Conference on Web Engineering*, vol. 3140 of *LNCS*, (Munich, Germany), pp. 502–515, July 2004.
17. "Advanced Technologies for interoperability of Heterogeneous Enterprise Networks and their Applications (ATHENA), European project." Project homepage: `http://www.athena-ip.org`.