Delegation Assistance

Achim D. Brucker
SAP Research
Vincenz-Priessnitz-Str. 1
76131 Karlsruhe
Germany
Email: achim.brucker@sap.com

Helmut Petritsch
SAP Research
Vincenz-Priessnitz-Str. 1
76131 Karlsruhe
Germany
Email: helmut.petritsch@sap.com

Andreas Schaad
SAP Research
Vincenz-Priessnitz-Str. 1
76131 Karlsruhe
Germany
Email: andreas.schaad@sap.com

***Abstract*—Today's IT systems typically comprise a fine-grained access control mechanism based on complex policies. The strict enforcement of these policies, at runtime, always contains the risk of hindering people in their regular work.**

An efficient support for assisted delegation can help in resolving the conflict between too tight access control and the required flexibility as well as support the resolution of conflicts. Here, assisted delegation means that, additional to denying the access, a user is informed about a list of users that could either grant him access to the requested resource or which could execute this task in behalf of the user.

In this paper, we present an approach for determining a set of users which are able to resolve an access control conflict. This set is based on various information sources and are ordered with respect to different distance functions. We show that one distance function can be used to serve different types of contextual input, e. g., role hierarchies, geospatial information as well as shared business object structure data or social network graphs.

***Keywords*-delegation and revocation, policy enforcement, security services, security architecture**

I. Introduction

Today's IT systems comprise a fine-grained access control mechanism based on complex policies. The enforcement of these policies, at runtime, always contains the risk of hindering people in their regular work [1]. Thus, writing security policies reflects always a trade-off between the risk of accessing secured data or operations and the benefit gained by using them.

An efficient support for *assisted delegation* can help in resolving this conflict. Here, assisted delegation means that a user is, additional to a deny, informed about a list of users that could either grant him access to the requested resource or which could execute this task (or a sub-task thereof) in behalf of the user. For ensuring a timely response, the request should be delegated to the "closest" person, which has or could establish mutual trust and is able to resolve the situation. These requirements raise the following two questions: first, what is the meaning of being close to each other, and second, who is able to resolve a given situation. With respect to the second questions, we need at least to consider the following two possibilities of determining the users which are most well-suited for resolving the conflict:

- users that are able to resolve the situation by executing a specific task. In this case, the originator (i. e., the user whose access rights are not sufficient) can ask them for help by either performing the task themselves or delegating the required access rights, to him.
- users that can change the current security policy, i. e., extend the access rights of the requester. In this case, after changing the policy, the requester is able to resolve the situation (and future occurrences thereof) himself.

With respect to the first question, several information sources could be considered, for example, users that are

- close with respect to the organizational hierarchy,
- close with respect to their office location,
- close with respect to their current location,
- close with respect to the security policy, and
- currently available.

These information sources can be represented by distance functions that increase the chance of asking a user for help that knows the originator of the request or his tasks personally.

This kind of delegation assistance can help to avoid deadlocks and bad usability [2] caused by too strictly enforced security policies. In our experience, such situations arise in particular often within

- disaster management teams,
- small and midsize enterprises (SME),
- agile environments within large enterprises, and
- industries working with privacy relevant data.

While IT systems for these application areas are in particular need for supporting assisted delegation, we believe that supporting assisted delegation is applicable to general IT systems, e. g., reporting and analysis [3]. Moreover, we see a special benefit of assisted delegation in process-based systems, e. g., workflow-management systems [4].

The rest of the paper is structured as follows: we briefly summarize our approach in Section II and present, in Section III, an assisted delegation architecture. In Section IV, we explain the delegation assistance procedure in more detail and present a prototype. We discuss different information sources in Section V, in Section VI we present an evaluation of our prototype. Finally, Section VII concludes the paper.

II. Delegation Assistance in a Nutshell

In this section, we describe a new concept, called *delegation assistance*, for assisting end users finding a user (mentor) which is permitted to execute a specific task (or, in systems supporting delegation, find a person which is permitted to delegate a specific task). Such a delegation assistance could be used in two different ways:

1) for providing feedback to users not permitted to access a resource. For these users (requester) mentors are suggested, which could either execute the required task or could delegate the required access rights to the requester.
2) for finding a user to whom a task could be delegated, without that access rights have to be delegated (e. g., find a representative for the time of holidays).

The result of the delegation assistance approach is a set of subjects based on various information sources that are combined using user configurable distance-functions for finding the optimal set of persons.

As example, we define a situation where a requester searches a mentor to access a restricted resources. Thus, we focus on a situation where the access decision of the policy evaluation is *not* granting access (the access decision is "deny," an exception like "rule not applicable," or "system not available"). In such situations, further assistance can be required for preventing systems stagnation.

For resolving this situation, the user could ask someone (i. e., a mentor) to access the required resources in his place (e. g., take over a (well-defined) sub-task which requires access to the protected resource). Or, the user could ask someone to grant him access to the required resource (e. g., by delegating the required access rights). Our approach particularly addresses the problem of finding the right (set of) users, which can be characterized by to opposed scenarios:

1) there are a lot of possible mentors which are directly related to the user searching for assistance. Delegation assistance can support the user in, first, who of his direct contacts is permitted to act as mentor and, second, which of these persons are actual reachable (e. g., logged into the system).
2) there are only a few possible mentors which may not be directly related to the searching user. In this case, a mediator is required to establish mutual trust between the requester and a mentor. Delegation assistance helps the requester to discover indirect mentors to which he has a "close" contact, and who of his indirect contacts may have the permissions to be a mentor.

In practice, both situations, depending on the policies and the organizational structure of a company, may occur.

Figure 1 illustrates our running example: a company where the employees are distributed over several locations: Bob and Alice work in a group which is managed by Mark, several groups work together on a project, which is managed by the project lead Linus. The groups are divided in respect to their tasks, whereas the project lead Linus is working on another location as Bobs group.

Information from several information sources are represented as directed weighted graphs, called information graphs. While deriving these information graphs requires, usually, expert knowledge, there is a general rule: subjects that are close to each other should be connected with an edge having a (relatively) small weight.

Figure 1a shows an information graph which could be used to represent the distance function of this organizational situation: contacts within the group have least distance 10, the relation to a superior is estimated with 20, the contact with a super-superior with 30. For superiors to teammembers the distance is always 10.

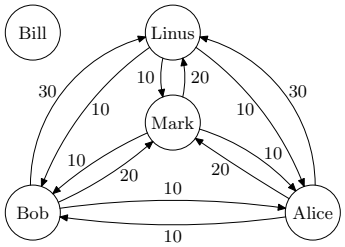
The (already existing) access control policy can be used as further information source. In our example, we assume a simplified RBAC policy: roles are assigned in respect to the function of the users, i. e., Bob and Alice are both in the role project member. Moreover, we assume Bob to be also in the role for system administrators and Linus and Bill are project managers (more powerful roles) of different projects. Finally, all users are assigned to project specific roles. Figure 1b shows the derived distance function graph. Here, subjects having similar roles are connected by an edge with low weight.

Finally, Figure 1c represents the location distances: Bob and Alice are working in the same room (10), all others are working in nearby rooms (20). We do not model edges between subjects working in different locations.

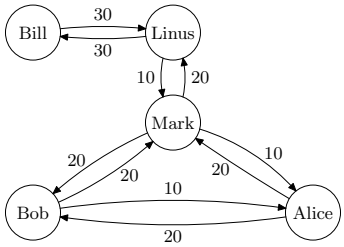
Recall our two different ways of using assisted delegation techniques. First, if Bob tries to access a resource for which many people from his location have access to, assisted delegation suggests users from Bobs location, preferring those from his group, project, or room. If none of them has the required permission (or is currently not available), currently reachable users from his location will be suggested. Thus, delegation assistance selects a reachable person from his location which therefore knows Bob as a colleague.

Second, assume Bob tries to access a resources assigned to a project from a different location, i. e., none of the project members share their location with Bob. Moreover, no member of Bob's team has access to the required resource. Applying delegation assistance helps in finding a person "close" to Bob which can introduce Bob to someone having the required access rights. In our example, Bill could be the project lead of the external project and, in this case, delegation assistance will suggest Bill to Bob. Linus will be suggested as a mediator known to both Bob and Bill.

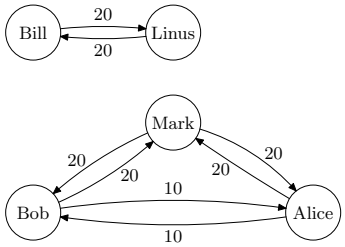
For both scenarios, the different relationship information need to be merged using a combining algorithm. We present such an algorithm in Section IV.



(a) Representing the distance functions of an organizational structure: Bill is the project lead of a team formed by Alice and Bob.



(b) Representing the system policy as distance graph: role similarities are estimated with a low weight.



(c) The location distance functions estimates least weight for connections within one room, and does not represent connections between locations.

Figure 1. Representing the distance function of the organizational structure, the policy and location as directed weighted graphs (information graphs).

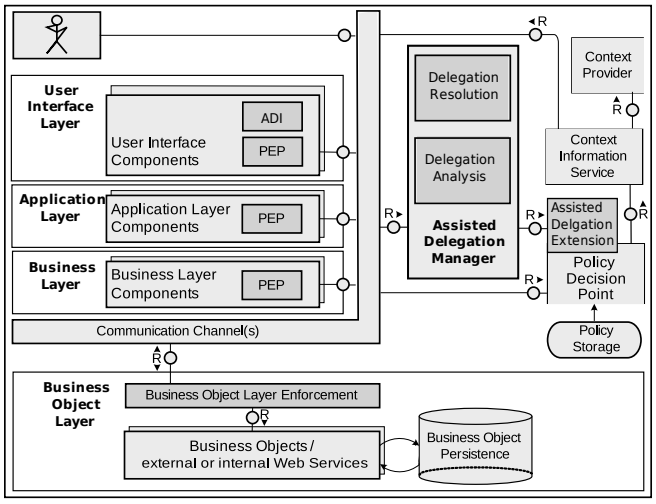


Figure 2. An Exemplary Delegation Assistance Architecture

III. A Delegation Assistance Architecture

In this section, we present a system architecture supporting assisted delegation (see Figure 2), comprising: A layered application containing a *user interface layer*, an *application layer*, a *process layer*, and a *business object* or *back-end layer*. Each of these layers may contain a Policy Enforcement Point (PEP) which enforces the decisions made by the Policy Decision Point (PDP). The layered application may also serve as context provider for providing context information via the context information service. The *Policy Decision Point* (PDP) is responsible for evaluating access decision requests and returning a respective access decision response. The *context providers* deliver, through the context information service, the context information required for both access control evaluations and assisted delegation.

The *Assisted Delegation Interface* (ADI) comprises the assisted delegation functionality visible to the end user. In case of a non-permitted access decision from the PDP, the ADI allows the user to start the consequential actions. The *Assisted Delegation Manager* (ADM) provides means for calculating the set of subjects that can help in resolving a denied access request. Moreover, the ADM assists the ADI in taking the actions required for resolving the current situation.

A. Assisted Delegation Interface (ADI)

The ADI defines the interface to the assisted functionality for the user. The search for a mentor optionally can be parametrized with user preferences (instead of using the default settings). These preferences may include

- which information sources are included (or excluded)
- how these information sources are combined
- which type of mentor is needed, whereas the option elements depend on the implementation of the access control infrastructure. Possible choices are users
  - with the required access rights,
  - with the right to delegate the required access rights,
  - able to modify (the affected part of) the policy, or
  - able to act as second user for an access requiring the four-eye principle

  Depending on the delegation implementation, these choice are not necessarily disjoint.

The ADI displays the results of the search (i. e., the mentors found) in a suitable way (e. g., a list of mentors with, if present, the mediator and a definition of the relationship between mediator and mentor). Furthermore, the ADI helps the user to trigger the required actions resolving the situation. For example, this can include further assistance for generating support-tickets or sub-tasks that can be delegated.

B. Assisted Delegation Manager (ADM)

The implementation of the ADM can be split into:

- The *delegation analysis component* processes the search requested from the ADI, whereas this request at least contains the (denied) access request, and optional user preferences; resulting in two types of mentor sets:
  - an ordered set of mentors containing subjects that can access the requested resource. i. e., the system stagnation can be prevented by delegating a sub-task to them.

– an ordered set of mentors containing users that can (temporarily) extend the users access right for the requests resource, i. e., the system stagnation can be prevented by extending the users access rights.

The delegation assistance algorithms and information sources are user-configurable and an optional part of the search request.

- The *delegation resolution component* assists the user in taking the required actions. For example, it could assists the user in generating tickets for a support-ticket system or in delegating a sub-task to a user. Thus, this component implements the support functionality provided by the ADI.

The decision, how the concrete implementation should look like depends mainly on the already existing system environment. Figure 2 depicts an architecture which is especially suitable for extending existing systems with support for delegation assistance.

C. Delegation Analysis

The delegation analysis component determines the set of users which are, first, permitted to execute or delegate a specific access, and, second, are close in respect to the searching user. With respect to the first task: policy decision frameworks are designed to decide efficiently if a specific user has access to a resource. The inverse problem, i. e., which users have access to a specific resource, is for usually not supported. Due to the complexity and expressiveness of most modern policy specification languages (often including arbitrary constraints, as, e. g., in XACML [5]), such implementations may only be able to approximate the result.

As a consequence, we suggest to over-approximate the access control policy by converting it (statically) into an information graph. This information graph should provide (in combination with other information graphs) an efficient way for determining the set of users that potentially have access to a given resource. Assuming that this over-approximation is not too coarse, we can use the existing PDP for testing that a user can in fact access the given resource.

Finally, we suggest the following procedure for the ADM:

1) combine the selected information graphs (e. g., locations, project roles). In Section IV, we present an algorithms for combing such graphs.
2) based on the weights on the (combined) information graphs, compute the ordered set of the n best mentors.
3) the abstracted policy, i. e., represented as an information graph, is used to remove all mentors which cannot access the given resource.
4) using an assisted delegation PDP extension, compute PDP requests for the n best potential users, i. e., the set of users computed in the previous step. By evaluating the generated requests, the set of potential users can be refined to the subset thereof that has access.

An optimized ADM implementation may require to reduce the list of candidates as early as possible. For example, limiting the set of subjects to users that can potentially access the given resource as first step (i. e., before the information graphs are combined), reduces the effort for the information graph combinations.

On the other hand, it can also prevent solutions by eliminating users needed for intermediate steps: if no direct contact of the searching user has the required access right, a trusted mediator is required to establish a trusted relationship between requester and mentor.

IV. A Delegation Assistance Algorithm

In this section, we present a generic framework for combining and analyzing a set of information sources. The main idea of our approach is to represent all information sources as graphs modeling the relations between different subjects. As a prerequisite, we define the concept of an information graph:

Definition 1: (Information Graph) An *Information Graph* is a directed weighted graph $G = (V, E, \omega, \omega_m)$ where V is a set of vertices (representing the subjects), E is a set of edges (representing the relations between the subjects), and $\omega : V \to \mathbf{N}_0$, where $\forall v \in V.\ 0 \leq \omega(v) \leq \omega_m$, is a function assigning a weight, i. e., a numeric label, to each edge which has to be smaller than ω_m.

Intuitively, the edge weight represent the (degree of) familiarity, on scale from 0 (very familiar) to ω_m (not familiar), between two subjects. We use directed graphs for representing asymmetric relations, e. g., a project lead usually has a close connection (can delegate tasks easily) to its regular project member, whereas the opposite is not necessarily true. Finally, we denote the set of information graphs (with maximum weight ω_m) with $\mathcal{G}_{\omega_m}$.

Our approach is based on the notion of merging two information graphs. Assume two information graphs $G_1, G_2 \in \mathcal{G}_{\omega_m}$ with $G_1 = (V_1, E_1, \omega_1, \omega_m)$ and $G_2 = (V_2, E_2, \omega_2, \omega_m)$. Conceptually, we merge G_1 and G_2 into a new information graph $G = (V, E, \omega, \omega_m) \in \mathcal{G}_{\omega_m}$ using a three-folded algorithm:

1) we merge the set of vertices:

$$V = V_1 \odot V_2, \text{ where } \odot \in \{\cup, \cap, \setminus, \ominus\}. \tag{1}$$

2) we merge the set of edges:

$$E = \{e \in E_1 \cup E_2 \mid \mathrm{src}(e) \in V \wedge \mathrm{dest}(e) \in V\} \tag{2}$$

where $\mathrm{src}(e)$ denotes the source vertex and $\mathrm{dest}(e)$ the destination vertex of the edge e.

3) we update edge weights for all edges $e \in E$:

$$\omega(e) = \begin{cases} \omega_1(e) & \text{if } e \in E \cap E_1 \setminus E_2 \\ \omega_2(e) & \text{if } e \in E \cap E_2 \setminus E_1 \\ f\big(\omega_1(e), \omega_2(e)\big) & \text{otherwise.} \end{cases} \tag{3}$$

where $f : \{0, \ldots, \omega_m\}^2 \to \{0, \ldots, \omega_m\}$ is a user-defined function merging the weights of edges that are part of both input graphs.

Thus, our algorithm is not only parametrized over the two input graphs, but also over the functions for merging vertices ($\_ \odot \_$) and edges ($f(\_, \_)$). In the following, we discuss different choices for these functions:

Vertices: The join and disjoin are, in our experience, the most often used methods for merging vertices. On the one hand, joining all vertices, i. e., $V = V_1 \cup V_2$, guarantees the maximal set of solutions. Especially in situation in which one subset has only a few connections or, more formally, if the maximum degree of the graphs to be merged is small. On the other hand, disjoining the vertices, i. e., $V = V_1 \cap V_2$, allows for a quick reduction of the solution set in cases where otherwise the end user would be swamped with to much information or the system would need to handle to large data sets not contributing to the solution.

The selection of the vertices merge function depends on the information the graph contains, whereas two main types can be distinguished: information graphs enhancing the quality of the edge weights are for usual combined with $\_ \cup \_$ (e. g., combining the organizational information graph and the policy information graph), information graphs enhancing the quality of the vertices are for usual combined with $\_ \cap \_$ (e. g., only consider users logged in or from the same location) or $\_ \setminus \_$ (e. g., remove users that are currently not available).

Edges: Choosing a good strategy for merging edge weights is more difficult than choosing a strategy for merging vertices. Overall, the class $\mathcal{G}_{\omega_m}$ should be closed under the application of this function, i. e., we require for a merge function f and a given ω_m that

$$\forall w_1, w_2 \in \mathbf{N}_0.\ (0 \leq w_1 \leq \omega_m) \wedge (0 \leq w_2 \leq \omega_m) \implies (0 \leq f(w_1, w_2) \leq \omega_m) \quad (4)$$

holds. Recall that a small edge weight represents a strong connection between the two subjects (vertices). Therefore, an intuitive (but not formal) requirement is

$$\forall w_1, w_2 \in \mathbf{N}_0.\ f(w_1, w_2) \leq w_1 \wedge f(w_1, w_2) \leq w_2\,, \quad (5)$$

i. e., the merged edge weight is smaller or equal the minimum of the two input weights. Thus, an obvious choice, satisfying both requirements, for f is the minimum function:

$$f(w_1, w_2) = \begin{cases} w_1 & \text{if } w_1 \leq w_2, \\ w_2 & \text{otherwise.} \end{cases} \quad (6)$$

As an alternative, we will use the function

$$f(w_1, w_2) = \frac{w_1 \cdot (\omega_m - w_2)}{\omega_m} \quad (7)$$

which also satisfies our requirements and, moreover, guarantees for all $0 < w_1, w_2 < \omega_m$ that the weight $f(w_1, w_2)$ is strictly smaller than the minimum of both input weights. Intuitively, we interpret w_2 as the percentage to be subtracted from w_1.

Further, we present a variant allowing to influence the reduction of merged edges by a factor $c \in \mathbf{R}$ ($c \geq 1$). Without loss of generality, we assume $w_1 \leq w_2$ and define:

$$f(w_1, w_2, c) = \frac{w_1 + \frac{w_1 \cdot w_2}{\omega_m} \cdot (c - 1)}{c} \quad (8)$$

Informally, c describes the gradient used for decreasing edge weights. This function has the following properties:

- If $c = 1$ then $f(w_1, w_2, c) = w_1$ holds.
- If $c_1 \geq c_2$ then $f(w_1, w_2, c_1) \leq f(w_1, w_2, c_2)$ holds.
- If $w_2 = \omega_m$ then $f(w_1, w_2, c) = w_1$ holds. Thus, if one of the input weights is equal to ω_m, the other weight remains unchanged.

Summarizing, choosing the appropriate methods for merging information graphs requires domain knowledge, i. e., an understanding about the information represented in the graphs. This hold especially for the function merging edge weights.

```
signature DELEGATION_INFORMATION_GRAPH = sig
  type vertex        (* abstract types for vertices  *)
  type edge          (* abstract types for edges     *)
  type weight = int  (* edge weights are integers    *)
  type graph         (* abstract types for graphs    *)

  (* enumeration which input graph contains a node: *)
  datatype contained = src | dest | both

  (* omitted several helper functions               *)

  (* a generic wrapper for applying weight merges   *)
  val apply: (int * int → int)
             → edge option → edge option → edge option

  (* an example function merging vertices          *)
  val vertices_un: (contained → vertex → vertex option)

  (* an example function merging edge weights      *)
  val edge_min: edge option → edge option → edge option

  (* merging two graphs: utilizing the higher-order
     functions of SML, this functions takes the two
     merging functions and the graphs as input.    *)
  val merge:  (edge option → edge option → edge option)
           → (contained → vertex → vertex option)
           → graph → graph → graph

  (* subgraph of subjects that are reachable with
     limited costs                                 *)
  val reachable_limit: graph → vertex → int → graph

  (* subgraph of the best reachable n subjects     *)
  val reachable_best:  graph → vertex → int → graph
end
```

Listing 1. Signature for merging information graphs.

A. Implementation

We implemented a prototype of our framework in the functional programming language SML [6]. Using higher-order functions leads to an implementation that is very close to the previously presented abstract algorithms. Listing 1

presents an excerpt of the SML signature describing the types of our implementation. Here, edge option describes the type allowing for "null values" (None), i. e., the absence of an edge. Overall, this implementation is in the following two aspects more flexible than our abstract representation:

1) our implementation supports arbitrary functions for merging vertices and
2) our implementation allows also functions for merging edges that change, or even omit, arbitrary edges of the input graphs (and not only edges that are contained in both graphs).

In Listing 1, we omitted several helper functions (e. g., constructors for edges, conversion function) and, of course, our implementation also contains implementations of all presented functions for merging edges or vertices.

Using the provided default implementations for joining all vertices (vertice_un) and merging edge weights using the minimum function (edge_min), we can merge two graphs G1 and G2 by using our implementation as follows:

```
val G  merge edge_min vertices_un  G1 G2
```

Due to support for higher-order functions in SML (in this case, roughly similar to function pointers in C), we can also provide user defined merge functions. For example, we can choose to use the average of both edge weights and define our own union of vertices:

```
fun merge_edge_weights (w1, w2) = w1 + w2 / 2
fun merge_vertices lft v1 v2 = v1
  | merge_vertices _   v1 v2 = v2
val G = merge (apply merge_edge) merge_vertices G1 G2
```

We applied our framework successfully to several case studies which we describe in Section VI.

V. Information Sources

In the following, we discuss several information sources, together with suitable distance functions that could be used for computing the lists of users that can help in resolving an access conflict.

Role hierarchy: For systems using hierarchical role-based access control, we propose the similarity of assigned roles as distance function, whereas the estimation algorithm should especially consider the role hierarchy. Using this information source as distance function prefers the delegation of sub-task to users with the least additional access rights needed for executing the given task.

Security labels: In systems using an access control systems based on labeling (e. g., Bell-LaPadulla [7]), we propose to use the hierarchy of security labels, which prefers resolution strategies that minimize the declassification distance of data.

Organizational structure: By using information about the organizational structure of a company (membership of users to divisions, administrative areas of accountability), assisted delegation can prefer subjects that are, somehow, responsible for the user. For example, this strategy would prefer asking the system administrator of ones own division for extending someones access rights over asking the company wide IT support.

Management hierarchy: Similar to the organizational structure, the management hierarchy can be exploited: this would prefer users within the same level or sub-tree of the hierarchy.

Process model: In systems driven by business processes, the process models are another information source. Here, one would prefer to delegate sub-tasks to subjects that are already involved in the overall process.

Office location: Subjects that know the user personally should be preferred in resolving a given access control conflict. This sorting can be done by using the office location as a measurement and preferring users that work close to each other.

Availability and physical location: Besides the static information discussed above, also situational data can be integrated into our approach. This includes for example the current location (e. g., measured via GPS) or free/busy information based on groupware entries. While the previously discussed sources are mainly static, i. e., they can be pre-computed efficiently, the dynamic (i. e., depending on the concrete system context) information needs to be updated at runtime.

First, this list of information sources is not meant to be exhaustive; other information sources can easily be integrated. Second, in practice, these information sources should be combined for increasing the quality of the proposed resolution strategy.

VI. Case Studies

We validated our approach using different organizational scenarios:

Scenario 1: In this scenario, we model a small company with 100 employees, distributed over three locations and working on 15 projects. The company has a flat management hierarchy.

Scenario 2: In this scenario, we model a mid-size company with 500 employees, distributed over four locations and working on 20 different projects. The company has a flat management hierarchy.

Scenario 3: In this scenario, we model a large company with 10 000 employees, distributed over many (i. e., 15) locations and working on 60 different projects. The company has a complex management hierarchy.

For each of these models, we generated information graphs representing the policy (based on a simple, hierarchical RBAC model), the location information (locations in different cities and different offices within one location), management hierarchy, and a graph representing the project teams. Moreover,

	Scenario 1	Scenario 2	Scenario 3
policy	14.2	52.7	782.4
combined	17.6	23.2	54.6
not in policy	0.0	6.4	13.2
policy (login)	7.3	22.1	326.1
combined (login)	10.5	17.9	41.3
not in policy (login)	0.0	5.2	9.5

Table I
APPLYING ASSISTED DELEGATION TO DIFFERENT SCENARIOS.

we generated a dynamic graph representing that about half of the users are currently logged into the system (i. e., the users that are currently available).

We evaluated our approach as follows: for each of the scenarios we computed the combined graph of all static information sources (i. e., not considering the login status) using the function from Equation 7 for merging edge weights. Thereafter, we simulated queries (representing not permitted access control request) and computed the size of the following sets:

- based on the policy graph as single information source, we computed the set of users that are reachable with lower or same costs as the tenth best user,
- based on combining all information sources, we computed the set of users that are reachable with lower or same costs as the tenth best user, and
- we computed the set of users that are, in the combined graph, reachable from the originating subject but were not connected in the policy graph.

In a second experiment, we "subtracted" after combining all other information graphs, the login graph. This ensures that only users currently available are part of the proposed solution set. Table I summarizes result of our experiments based on 100 queries. From these figures, we conclude that delegation assistance is especially helpful for large and mid-size scenarios. In more detail, we see that in small companies the employees have a high degree of connections. In its extreme, combining all information leads to a graph where most of the subjects are directly connected, which explains why in this case the set of people reachable with lower or same costs as the tenth best user is larger than when only considering the policy in isolation. In larger and mid-size applications, this is not the case: here, combining several information graphs leads to significantly more structure and results in better (smaller) sets of candidates. Moreover, in large and mid-size applications, there is a significant amount of users suggested that are not in the policy graph, i. e., there is a non-zero set of people that can act as mediator.

Finally, if we compare the first experiment with the second one, we observer that the decrease of candidates based on the policy (i. e., without combining several information sources) roughly decreases with the percentage of users currently logged in and therefore available. This effect is significantly reduced if we base the delegation assistance on combined information. As, after combining several information source, users are much better connected, the loss of a subject can often be replaced by a mediator.

Summarizing, our evaluations shows that assisted delegation improves both the quality (and size) of the set of possible candidates and also opens new solutions that are (at least not directly) possible without such a system.

VII. Related Work and Conclusion

A. Related Work

We see three areas of related work: system for supporting delegation, systems for dynamically extending access rights, and policy analysis systems.

Research on delegation [8], [9] mainly focuses on algorithms for determining the set of rights (e. g., expressed as certificates or as changes to a given policy) that a delegator has to transfer to a delegatee. This problem is orthogonal to our work, as in our case, we already know which access rights need to be changed. Existing delegation systems and their analysis techniques can be integrated into our approach. In particular, integrating an analysis for the rights needed to delegate a sub-task (and all its depending tasks) could decrease the chances that further delegation assistance is needed in further subsequent a specific sub-task.

Research on extending access rights dynamically [10], [11] comprises approaches that allow users to automatically, i. e., without the supervision of other users, extend their access rights. These extensions are usually implemented by providing temporary accounts with higher access rights. This approach is often called "Break Glass" and is also supported by commercial business software, e. g., as Virsa Firefighter for SAP. We see these approaches, that work without or only post-hoc human supervision, only applicable for emergency cases. Thus, assisted delegation positions itself in-between regular system operation and exceptional system operation.

There are several works on policy analysis systems [12], [13], [14], [15], e. g., for measuring the similarity of security policies or role hierarchies. These approaches can directly be integrated as distance functions (together with the required information sources) into our approach.

B. Conclusion and Future Work

We presented an architecture and a framework supporting users to resolve access control conflicts by finding a set of subjects, called mentors, which either have access to the required resource or can act as mediators. Overall, our approach for assisted delegation of tasks is independently from the underlying delegation system. In fact, our approach can be understood as a pre-computation of the, either supported by the underlying system or manual, delegation taking place.

As a main contribution of this paper, we presented a generic combination algorithm for information graphs, i. e.,

directed weighted graphs. The vertices of the graph represent the subjects (users), while the (rated) relationship between them are represented by (weighted) edges. This allows for combining several information sources into one combined model with increased expressiveness. We also presented a generic framework together with an implementation of the information graph combining algorithm.

Moreover, we presented an architecture that extends classical access control systems, supporting assisted delegation. Our proposal for an assisted delegation architecture does not require major modifications of the existing security relevant components (i. e., the PDP and PEP).

We see several lines of future work. On the application level, the combination with advanced caching strategies for access control decisions, e. g., [16], is particular valuable. As our implementation relies on evaluating different access control requests for testing if a user has actually access to a specific resource, our approach profits significantly from an overall performance gain in access control evaluations. Overall, this should be possible seamlessly, as our framework does not require modifications of the existing access control structure. On the theoretical side, more advanced translations of policies, and the analysis thereof, need to be developed to decrease the number of "false positive" users. For example, the assisted delegation extension for the PDP could be extended for providing a request dependent policy information graph. Such a graph would still be an over-approximation, but would provide a better estimation of potentially permitted user.

Finally, further functions for merging edge weights need to be considered and evaluated. Here we clearly envision that different types of information sources require specialized merge functions to be most valuable to the overall result.

ACKNOWLEDGMENT

This work has been supported by the German "Federal Ministry of Education and Research" in the context of the project "SoKNOS." The authors are responsible for the content of this publication.

REFERENCES

[1] D. Povey, "Enforcing well-formed and partially-formed transactions for Unix," in *Proceedings of the 8th conference on USENIX Security Symposium*, vol. 8. USENIX Association, 1999, pp. 5–5.

[2] B. D. Win, F. Piessens, W. Joosen, and T. Verhanneman, "On the importance of the separation-of-concerns principle in secure software engineering," in ACSA *Workshop on the Application of Engineering Principles to System Security Design (Serban, C., ed.)*, 2003, pp. 1–10.

[3] T. Priebe and G. Pernul, "Towards OLAP security design: survey and research issues," in *Proceedings of the 3rd* ACM *international workshop on Data warehousing and* OLAP. New York, NY USA: ACM Press, 2000, pp. 33–40.

[4] V. Atluri and J. Warner, "Supporting conditional delegation in secure workflow management systems," in *Proceedings of the tenth ACM symposium on Access control models and technologies (SACMAT)*. New York, NY USA: ACM Press, 2005, pp. 49–58.

[5] "eXtensible Access Control Markup Language (XACML), version 2.0," 2005. [Online]. Available: http://docs.oasis-open.org/xacml/2.0/XACML-2.0-OS-NORMATIVE.zip

[6] L. C. Paulson, *ML for the Working Programmer*. Cambridge Press, 1996.

[7] D. E. Bell and L. J. LaPadula, "Secure computer systems: A mathematical model, volume II," in *Journal of Computer Security 4*, 1996, pp. 229–263, an electronic reconstruction of *Secure Computer Systems: Mathematical Foundations*, 1973.

[8] D. W. Chadwick and A. Otenko, "The PERMIS X.509 role based privilege management infrastructure," in *Proceedings of the seventh ACM symposium on Access control models and technologies (SACMAT)*. New York, NY USA: ACM Press, 2002, pp. 135–140.

[9] C. Ye and Z. Wu, "Using XML and XACML to support attribute based delegation," in *CIT '05: Proceedings of the The Fifth International Conference on Computer and Information Technology*. Washington, DC, USA: IEEE Computer Society, 2005, pp. 751–756.

[10] "Break-glass: An approach to granting emergency access to healthcare systems," Joint NEMA/COCIR/JIRA Security and Privacy Committee (SPC), White paper, 2004.

[11] A. D. Brucker and H. Petritsch, "Extending access control models with break-glass," in *ACM symposium on access control models and technologies (SACMAT)*. ACM Press, 2009.

[12] J. Warner, V. Atluri, R. Mukkamala, and J. Vaidya, "Using semantics for automatic enforcement of access control policies among dynamic coalitions," in SACMAT, V. Lotz and B. M. Thuraisingham, Eds. New York, NY USA: ACM Press, 2007, pp. 235–244.

[13] K. Fisler, S. Krishnamurthi, L. A. Meyerovich, and M. C. Tschantz, "Verification and change-impact analysis of access-control policies," in ICSE, G.-C. Roman, W. G. Griswold, and B. Nuseibeh, Eds. New York, NY USA: ACM Press, 2005, pp. 196–205.

[14] J. Bryans, "Reasoning about XACML policies using CSP," in *Proceedings of the 2005 workshop on Secure Web services*. New York, NY USA: ACM Press, 2005, pp. 28–35.

[15] D. Lin, P. Rao, E. Bertino, and J. Lobo, "An approach to evaluate policy similarity," in SACMAT, V. Lotz and B. M. Thuraisingham, Eds. New York, NY USA: ACM Press, 2007, pp. 1–10.

[16] M. Kohler and A. Schaad, "Pro active access control for business process-driven environments," in *Annual Computer Security Applications Conference*. Los Alamitos, CA, USA: IEEE Computer Society, 2008, pp. 153–162.