A Scenario-driven Role Engineering Process for Functional RBAC Roles

Gustaf Neumann
gustaf.neumann@wu-wien.ac.at

Mark Strembeck
mark.strembeck@wu-wien.ac.at

Department of Information Systems, New Media Lab
Vienna University of Economics and BA, Austria

ABSTRACT

In this paper we present a novel scenario-driven role engineering process for RBAC roles. The scenario concept is of central significance for the presented approach. Due to the strong human factor in role engineering scenarios are a good means to drive the process. We use scenarios to derive permissions and to define tasks. Our approach considers changeability issues and enables the straightforward incorporation of changes into affected models. Finally we discuss the experiences we gained by applying the scenario-driven role engineering process in three case studies.

Categories and Subject Descriptors

D.2.1 [**Software Engineering**]: Requirements/Specifications - Elicitation methods, Methodologies; D.2.9 [**Software Engineering**]: Management - Life cycle, Software process models; D.4.6 [**Operating Systems**]: Security and Protection - Access controls

General Terms

Security, Design, Management, Human Factors

1. INTRODUCTION

Role engineering for role-based access control (RBAC) is the process of defining roles, permissions, constraints and role-hierarchies [3]. Roles can be differentiated between functional and organizational roles. Functional roles reflect the essential business functions that need to be performed within a certain company. Organizational roles correspond to the hierarchical organization in a company in terms of internal structures. Functional roles, in contrast to organizational roles, are robust against organizational restructuring since business tasks are most often not reflected in organizational structures. This paper presents a role engineering process designed for functional RBAC roles. It is scenario-driven and based on requirements engineering techniques. The process can be applied to build a concrete RBAC model, i.e. an actual instance of the abstract RBAC models defined in [9, 23]. The process definition presented in this paper provides guidance for security engineers and enables the engineering of adaptable models that facilitate the incorporation of changes into a configuration consisting of several RBAC related models. To date we gained practical experiences by applying the scenario-driven role engineering process in three case studies. Currently we are conducting a fourth case study that will most probably provide us with additional interesting insights.

1.1 Motivation

RBAC [9, 23] is a very popular approach in both research and industry. Many recent RBAC related publications deal with sophisticated technical aspects of RBAC and the realization of the corresponding techniques (e.g. [8, 12, 17]). Nevertheless only few contributions are concerned with the process of role engineering which is focused on the modeling of a concrete instance of an RBAC model (see [3]).

Before a concrete RBAC model (which is a result of the role engineering process) can be implemented technically, the role engineering activities must take place. Unfortunately many present approaches for role engineering are merely defined on an ad hoc basis or treat only a small part of the whole process. We thus aim at a systematic role engineering approach that is flexible enough to be applicable in different kinds of organizations. An important requirement for the process is to support change management activities to ease the propagation of changes that occur within the information system or its environment into all security relevant models and finally into the concrete RBAC model.

Role engineering is in essence a requirements engineering process. Our approach is based on the concept of scenario which is well known and widely applicable in (software) engineering. Scenarios model the usage of systems and facilitate the communication among engineers as well as the communication between engineers and non-technical stakeholders. Therefore scenarios are practical means that allow for the consideration of the strong human factor in role engineering.

1.2 Scenarios: An Overview

Requirements engineering [16, 20] differentiates functional requirements and quality (or non-functional) requirements. Functional requirements define a system's purpose, i.e. the intended use of the system, while quality-requirements define demands like maintainability, portability, interoperability or performance. To capture, depict and organize functional requirements three general categories of requirements models are distinguished:

- *Goal models*: Goals [25] depict requirements on an abstract level. Goals are, for instance, well suited to capture requirements on a business process level. Thus they could be used to facilitate e.g. the communication with managers.

- *Scenario models*: Scenarios [4] depict system usage in the form of action and event sequences. Scenarios are especially well suited to model systems from a user perspective and ease the communication with end users.

- *Solution models*: Those models capture the intended solution(s) in more detail and bridge the gap between requirements models and the system architecture. Therefore they facilitate the communication of requirements engineers with implementation engineers. UML [1] is the de facto standard for solution-oriented models. Its different diagram types can be used in almost every phase of system development.

In general, scenarios describe possible or actual action and event sequences. Scenarios enable reflections about (potential) occurrences, and opportunities or risks. Therefore they facilitate the detection of capable solutions/reactions to cope with the corresponding situations. The idea of scenarios is used since ancient history, for example to describe and assess alternative business-, politics-, or war-strategies.

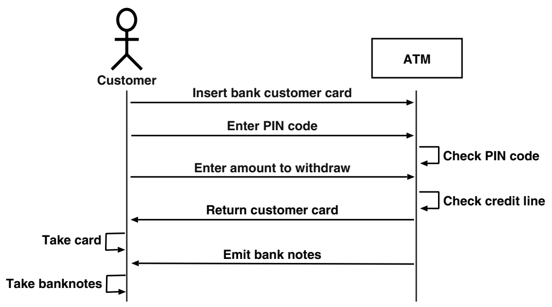

Figure 1: Example of a simple scenario

In the area of software (systems) engineering scenarios are used to explore and describe the system behavior as well as to specify user needs. Although scenarios were used in systems engineering before, Jacobson's use case approach [13] served as a catalyst for the acceptance and dissemination of scenarios in this area. The huge quantity of scenario-related literature (e.g. [2, 4, 13, 14, 22]) is an indicator for the great interest in scenarios and their multifaceted applicability in the field of systems engineering.

Scenarios can be described in many different ways. Commonly they are specified with (structured) text descriptions and different types of diagrams, e.g. message sequence charts, activity diagrams, or petri-nets. Figure 1 shows a simple scenario for the withdrawal of money from an automated teller machine (ATM), depicted as message sequence chart. Each scenario step is represented by an arrow and a short textual description. The same scenario could also be described through a video sequence or with structured text for example. Each type of description emphasizes different things and enables the detection of different connections and interrelations. Moreover scenarios on the requirements level can be further refined and concretized into "solution-oriented" scenarios that depict the dynamic runtime structures of a system, e.g. how a specific user functionality is realized on the level of interacting software components. For example the "check credit line" step of the scenario shown in Figure 1 could be described by an own scenario. The corresponding concretized scenario then shows how the ATM interacts with a bank server to check if the customer is allowed to withdraw a certain amount of money from her/his account. Thus concretized scenarios may also depict system internal activities that are not (directly) visible to the user.

After this short introduction to scenarios, the remainder of the paper is structured as follows. In Section 2 we give an overview of the scenario-driven role engineering approach. Section 3 presents the different models which are build in the course of the process and describes their interrelations. In Section 4 we give a detailed process description before we discuss the experiences we gained from the application of the process in Section 5. We then give an overview of related work in Section 6 and conclude the paper and give an outlook on future activities.

2. THE SCENARIO-DRIVEN ROLE-ENGINEERING APPROACH

In the scenario-based approach each action and event within a scenario can be seen as a *step* that is (typically) associated with a particular access operation. Thus a scenario is a good source for the derivation of permissions which are applied in a particular order to reach a predefined (user) goal. A subject (e.g. a user or an autonomous agent) performing a scenario must own all permissions that are needed to complete every single step of this scenario.

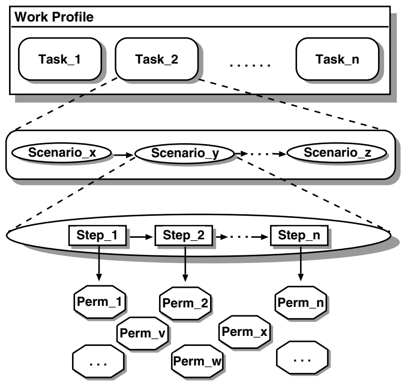

Figure 2: Composition of Work-Profiles

Every *task definition* (e.g. processing a damage event in an insurance company) corresponds to one or more scenarios. These tasks are again combined to form work profiles. A *work profile* comprises all tasks that a certain type of employee (or user in general) can perform. Since each scenario is linked to a set of permissions it is possible to derive the permissions for a particular work profile directly from the tasks/scenarios (see Figure 2). Therefore work profiles are the source for the definition of a preliminary role-hierarchy.

The scenario-driven role engineering process is composed of seven major activities which define own sub-processes respectively (depicted in Figure 3 as activity diagram):

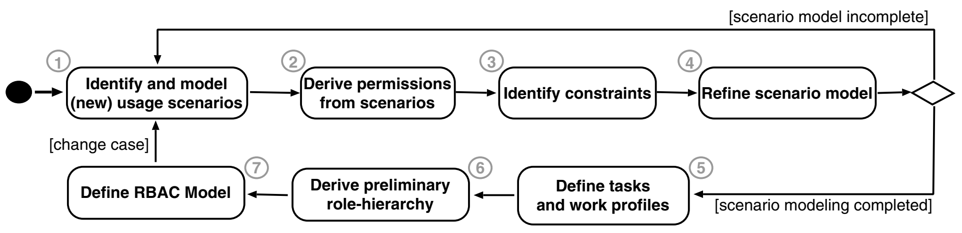
[scenario model incomplete]
1
Identify and model (new) usage scenarios
2
Derive permissions from scenarios
3
Identify constraints
4
Refine scenario model
[change case]
7
Define RBAC Model
6
Derive preliminary role-hierarchy
5
Define tasks and work profiles
[scenario modeling completed]

Figure 3: High-level view of the role-engineering process

1. *Identify and model usage scenarios*: In this activity sensible system usages are identified and explicitly modeled in terms of scenarios.

2. *Derive permissions from scenarios*: For each scenario the access operations that are necessary to execute the corresponding step-sequence are identified and stored in a permission catalog as ⟨operation, object⟩ pairs.

3. *Identify constraints*: Constraints to be enforced on permissions are identified and made explicit, e.g. separation of duties, cardinalities, or time-dependencies. All constraints are stored in a constraint catalog.

4. *Refine scenario model*: This activity reviews the current scenario model (defined in step 1). For similar scenarios a common generalization can be defined. In addition each scenario is examined if one or more of its steps can be further concretized by an own (sub)scenario. This activity is therefore similar to the definition of class-hierarchies in object-oriented design.

5. *Define tasks and work profiles*: Different scenarios are composed to form task definitions. This is done in accordance with the constraint catalog. Subsequently these tasks serve as building blocks for work profiles. A scenario may be associated with several tasks and a task may be associated with several work profiles, i.e. the scenario to task relation and the task to work profile relation are many-to-many relations respectively.

6. *Derive preliminary role-hierarchy*: The work profiles and the permission catalog are used for a semi-automatic creation of a preliminary role-hierarchy, i.e. obvious junior- and senior-roles are identified and arranged in an inheritance hierarchy. Potentially redundant roles are identified and marked for review.

7. *Define RBAC Model*: Here the preliminary role-hierarchy, the permission catalog and the constraint catalog serve as input for the definition of the concrete RBAC model. In this sub-process redundant roles are removed, new roles and role constraints are defined and role-hierarchies are merged or separated. These steps are repeated until the role model is complete, i.e. until the engineers who are responsible for this activity define the model as adequate.

As depicted in Figure 3 the activities 1 to 4 form a cycle that is repeated until the scenario model is complete. This is a prerequisite for activities 5 to 7. Nevertheless, the whole process (activities 1 to 7) is intended to be executed in an iterative and incremental manner, where each iteration results in a new evolutionary stage of the different models.

Once a scenario model is built (after the first iteration of the steps 1 to 4) changes that enrich the functionality of the system can be incorporated straightforwardly. Such a change case is characterized through the definition of a new usage scenario and may take place in step 4 (refinement of the scenario model) or after the concrete RBAC Model is built in step 7 (see Figure 3). The new scenario is then inserted into the existing scenario model. Afterwards the new permissions (if any) are derived from this scenario, the scenario is assigned to one or more task definitions and work profiles and finally the corresponding RBAC Model is updated accordingly. Of course one has to make provisions for such a change case in advance, so that the change can be correctly propagated into the different models. We deal with these aspects in more detail in the following sections.

3. MODEL INTERRELATIONS

Figure 4 depicts the interrelations of the models and documents that are produced during the scenario-driven role engineering process:

- The *Scenario Model* comprises all usage scenarios of the system under consideration and serves as the base model for our approach.

- The *Permission Catalog* consists of all permissions identified for a system. Since scenario steps are associated with access operations, the permissions are derived directly from the scenarios. Permissions consist of ⟨operation, object⟩ pairs and have a unique name or identifier.

- The *Constraint Catalog* contains the constraints that must be enforced for permissions. In the further course of the process the constraint catalog may be extended with constraints that must be enforced for roles (which are defined later). However, we do not restrict the kind of constraints that can be defined, nor do we require certain types of constraints to be defined. Therefore the constraint types to be modeled are only restricted by the RBAC service that is applied to implement the concrete RBAC model (i.e. the constraint types that can be enforced by a particular RBAC service).

- The *Task Definitions* describe tasks that are performed by certain users of the system, or by other subjects as autonomous agents for example. Every task consists of one or more scenarios which are performed in succession or in parallel to reach a particular goal.

- The *Work Profiles* consist of different task definitions. Every single work profile is (intended to be) a complete description of all tasks that a specific kind of

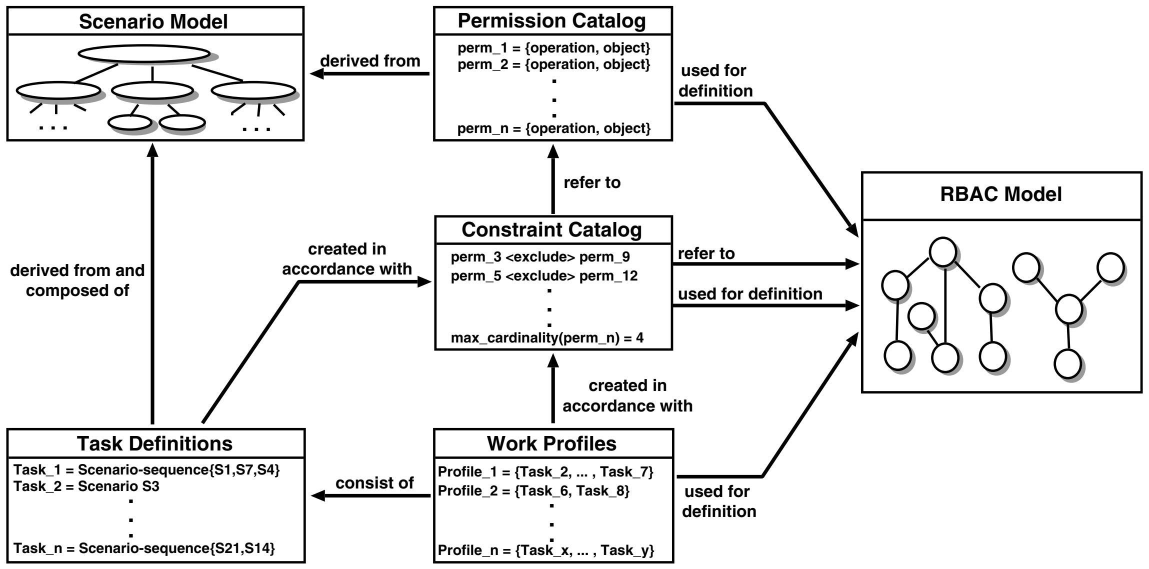
Scenario Model
. . .
. . .
Permission Catalog
perm_1 = {operation, object}
perm_2 = {operation, object}
perm_n = {operation, object}
derived from
used for definition
refer to
RBAC Model
Constraint Catalog
perm_3 <exclude> perm_9
perm_5 <exclude> perm_12
max_cardinality(perm_n) = 4
refer to
used for definition
derived from and composed of
created in accordance with
created in accordance with
Task Definitions
Task_1 = Scenario-sequence{S1,S7,S4}
Task_2 = Scenario S3
Task_n = Scenario-sequence{S21,S14}
consist of
Work Profiles
Profile_1 = {Task_2, ... , Task_7}
Profile_2 = {Task_6, Task_8}
Profile_n = {Task_x, ... , Task_y}
used for definition

Figure 4: Interrelations of the models and documents used and produced in the role-engineering process

user needs to perform or is allowed to perform. In our approach work profiles can thus be seen as *preliminary RBAC roles.* In the following we provide more details concerning the differences of our work profiles and RBAC roles, and on the process of deriving an RBAC role-hierarchy from these work profiles.

- The concrete *RBAC Model* is the final result of the role-engineering process and comprises all roles of the system arranged in one or more role-hierarchies. We define role-hierarchies as inheritance hierarchies were senior-roles inherit permissions and constraints from all of their junior-roles (transitively).

3.1 Work Profiles vs. Roles

As described above permissions are not explicitly associated with work profiles but can be transitively obtained through the scenarios associated with a specific work profile (see Figure 2). This is an essential difference between work profiles and RBAC roles, since in RBAC permissions are directly assigned to roles.

Furthermore work profiles are standalone definitions and have no direct link to other work profiles. Since a task may be assigned to more than one work profile and each scenario may be assigned to more than one task there are potentially many redundancies in the work profile definitions. This is another important difference between work profiles and RBAC roles which are arranged in inheritance-hierarchies and thus try to minimize redundancies. Therefore work profiles can be viewed as a preliminary stage of RBAC roles. Nevertheless, work profiles are a significant step towards the definition of a concrete RBAC model.

It is important to mention that the principle of least privelege is directly supported by the scenario-driven process. Since each task definition is associated precisely with the scenarios that need to be performed to fulfill a specific goal (e.g. a business function) the corresponding RBAC role can be equipped with the exact number of permissions that are needed to perform the respective task.

3.2 Traceability: Design for Change

The model interrelations depicted in Figure 4 already indicate that explicit traceability links [11] between the models have to be established. Those traces are essential to enable an efficient management of the models, e.g. to easily review which permissions are needed in a particular scenario as well as all scenarios (and therefore tasks, and work profiles) a specific permission is used in. Moreover traceability links facilitate the comprehensibility of models and are a prerequisite for an effective change management that enables the correct and cost efficient propagation of changes into affected models, e.g. if a new usage scenario is defined which must be assigned to tasks and work profiles and may finally result in an updated RBAC model. Some examples for such trace relations are: *concretized by* between two scenarios, *needed to perform* between a permission and a scenario, *defined by* between a constraint and the origin of this constraint, *part of* between a scenario and a task, or *implemented through* between a work profile and an RBAC role.

Unfortunately it is by far too expensive and time consuming to capture all possible traces. Furthermore, recording all possible trace relations would result in an unmanageable amount of information, and efficiently selecting the relevant traces to be considered in a given situation would be almost impossible. Therefore the information to be recorded has to be carefully selected and must be organized in a way which facilitates the use of the information in later development/change situations. However, the selection and management of trace information is an own field of research and a very complex task (see for instance [11, 19]) which is beyond the scope of this article. Nevertheless, we mentioned this problem domain since it is very important for the efficient handling of evolving complex models of all kinds.

4. A DETAILED PROCESS DESCRIPTION

In this section we describe the different activities of the role-engineering process in more detail. Each activity defines an own sub-process and is described in an own subsection.

4.1 Identify and model usage scenarios

In this sub-process sensible usage scenarios for the system under consideration are identified and modeled. At first the identified usage scenarios are described with a short sentence. Simple examples of such short descriptions for scenarios from different domains could be: "Enter an exam result into a student profile" in a university information system; "Transfer money from one bank account to another" in a banking application; "Create a new patient record" in a hospital information system.

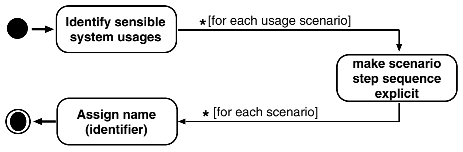
Identify sensible system usages
* [for each usage scenario]
make scenario step sequence explicit
* [for each scenario]
Assign name (identifier)

Figure 5: Scenario modeling sub-process

Since these scenarios subsequently serve as the basis for the derivation of permissions and the definition of tasks and work profiles, it is essential that the step sequence within each scenario is explicitly defined and written down (see Figure 5). Therefore each scenario is described through a detailed description in the form of structured text and a corresponding diagram (cf. Section 1.2). To identify scenarios and the corresponding step sequences, security engineers rely on the assistance of domain experts, like a professor, students and an administration secretary for a university information system, or a physician, a nurse, and a hospital clerk for a hospital information system. In the third step of the scenario modeling sub-process each scenario is provided with a unique name to identify the scenario and to facilitate search operations within the scenario model.

4.2 Permission derivation

The permission derivation sub-process is depicted in Figure 6. The goal of the corresponding activities is the identification of the permissions which are necessary to perform the usage scenarios of the system. The result of this sub-process is the permission catalog (see Figure 4), which contains all permissions that were detected from the scenarios.

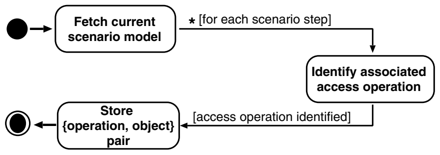
Fetch current scenario model
* [for each scenario step]
Identify associated access operation
[access operation identified]
Store {operation, object} pair

Figure 6: Permission derivation sub-process

During permission derivation every single scenario is reviewed. To identify permissions we take each scenario step and check which operation a subject (e.g. a user) needs to perform to complete this step. For each of these operations we define and store an ⟨operation, object⟩ pair in the permission catalog. Figure 7 illustrates a generic scenario as message sequence chart. The left hand side shows the subject depicted by an actor symbol, the object is shown on the right hand side. The arrows between the subject and the object represent operations that the subject invokes on the object. Therefore we are able to derive the ⟨operation, object⟩ pairs directly from the scenarios.

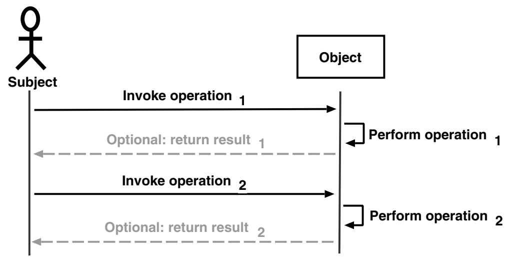

Figure 7: A generic scenario

Some of the basic steps are often included in many different scenarios (e.g. "load customer record" or "check credit line"). Nevertheless each permission is registered exactly once in the permission catalog. However, each permission (typically) has links to several scenarios (see Section 3).

Permissions can be differentiated in abstract and basic permissions (see [23]), i.e. permissions on different levels of granularity. Abstract permissions (like "transfer money") are composed of basic permissions (like "read account" or "write account") or of other abstract permissions. The scenario-based approach also enables the detection of permissions with different granularity. As described in Section 1.2 each step in a scenario may be further concretized by a new scenario. The operations performed in the concretized scenario are the elements of the abstract permission defined in the more general scenario.

4.3 Identification of permission constraints

The identification of constraints is one of the most difficult parts of the role-engineering process. The first step is to define which types of constraints should be modeled. Two of the most common types are separation of duties and cardinalities. However one may also model other kinds of constraints like time-dependencies (e.g. only between 6 a.m. and 8 p.m.) or maximum executions in an interval (e.g. maximum number of money transfer operations per day). The constraint types that are effectively modeled are only restricted by the RBAC service that should be used to implement the corresponding constraints. Nevertheless it may be sensible to model constraints even though the applied RBAC service is not (yet) able to enforce these constraints.

Having defined the relevant constraint types the identification of the actual constraints can begin. In essence an own sub-process is needed for every constraint type, e.g. one for static separation of duties (SSD) constraints, one for cardinality constraints and so on. Figure 8 shows an example for the identification of statically mutual exclusive permissions (SSD constraints). If for instance the RBAC model to be build should contain SSD constraints and cardinalities, a similar sub-process has to be executed for the identification of cardinality constraints. Of course several of these constraint identification sub-processes may run in parallel, but on a logical level each of them is an autonomous sub-process.

The constraint definition process in general is difficult because every organization has an individual access control policy and sometimes every department within an organization has specific rules that need to be enforced in addition to the corporate policy. Small inaccuracies could already result in severe security leaks.

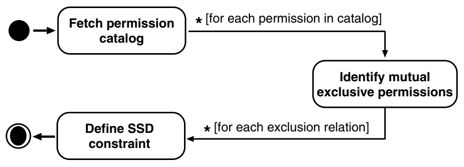
Fetch permission catalog
* [for each permission in catalog]
Identify mutual exclusive permissions
* [for each exclusion relation]
Define SSD constraint

Figure 8: A constraint definition sub-process

Constraints are identified by talking to domain experts like the executives that define the access control policy for an organization. These persons are (or should be) able to indicate the permissions which must never be given to the same person or the minimal and maximum number of people who must or should posses a particular access right. Another possibility for constraint identification is that the security engineers who build the RBAC model try to identify the constraints from their experiences. Subsequently this initial constraint model is refined together with domain experts, e.g. stock-brokers for a stock and bond trading system.

The ideal case for the constraint definition would, however, be a situation were the organization that runs the analyzed information system has a clearly defined access control policy and a dedicated security officer who is able to thoroughly define the security requirements. Unfortunately even in large organizations the security officer position is often a "sideline job" of system administrators. However, administrators usually have good knowledge of the technical issues of computer system security but are not involved in the planning and definition of organization wide security policies for sensitive information. Thus they are mostly not the correct contact persons for the identification of security policies in general, and for constraints in specific.

Another important issue is that from our experiences it is sensible to identify constraints on individual permissions prior to the identification of constraints for roles (which "contain" the permissions). The reason for that is twofold, on the one hand some constraints may only be sensibly defined on the permission level and not on the role level and vice versa, on the other hand certain types of constraints may be defined on both levels (for permissions and roles).

A good example are mutual exclusive permissions. Two mutual exclusive permissions must never be assigned to the same role (or user). Therefore a role to which such a permission is assigned must automatically inherit the separation of duties constraint that is attached to this permission, although the constraint is defined on the permission and not on the role. The consequence of that is that no user must ever acquire two roles which possess two mutual exclusive permissions. Nevertheless these two roles are only mutual exclusive since each of them owns a permission that is mutual exclusive to at least one permission of the other role.

Aside from that it may also be sensible to define two roles as mutual exclusive to prevent that the same user can perform certain access-sequences (scenarios). This means that every single operation may be unobjectionable but their combination (sequential application) is not. An alternative way to resolve such a situation is to define an abstract permission (see Section 4.2) that represents this access sequence. Afterwards this abstract permission can be assigned to a role and defined as mutual exclusive to other permissions. Nevertheless situations may occur where the definition of an additional (abstract) permission is not possible or undesired, it is then sensible to relocate this problem from the permission level to the role level (to prevent the execution of particular sequences).

Another example can be time-dependencies. While it may be inoffensive to execute a certain access right in general, one can imagine a constraint where the members of a particular role (e.g. customers) should only be able to apply the corresponding permission from 8 a.m. to 6 p.m. for instance. In such a situation it is sensible to define a respective constraint on one particular role and not on the permission in general.

Permissions can be seen as basic building blocks while roles (at least) consist of permissions and are therefore more complex entities. As a rule of thumb we recommend to define constraints on the lowest possible level. This means that one should first try to define constraints on the permission level and only specify constraints on the role level if the corresponding constraint cannot be sensibly defined on the permission level. From our experiences this eases the constraint management in general since constraints exponentially raise the complexity of the assets they are assigned to (on a logical and on the implementation level).

4.4 Scenario model refinement

Here the initial scenario model that was built in step one (see Section 4.1) is reviewed and further refined. In essence one can distinguish two essential activities in this sub-process:

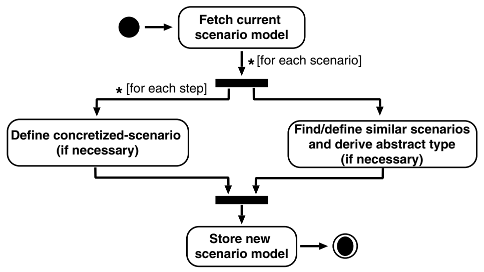
Fetch current scenario model
* [for each scenario]
* [for each step]
Define concretized-scenario (if necessary)
Find/define similar scenarios and derive abstract type (if necessary)
Store new scenario model

Figure 9: Scenario model refinement

- *Concretion*: each step within each scenario is reviewed if it is complex enough to be described more detailed through an own sub-scenario.

- *Generalization*: first the current scenario model is reviewed if similar scenarios exist. Second, for each scenario additional (similar) scenarios are defined. Third, for each group of similar scenarios it is examined if an abstract type for these scenario can be defined. An example are scenarios that describe the process of withdrawing money from a bank account, e.g. at an ATM, at the counter, or with an internet banking application. These scenarios are then grouped and a common abstract type is derived, e.g. the type "withdraw money".

4.5 Definition of tasks and work profiles

In this sub-process scenarios that logically belong together are combined to tasks. These tasks are then used to define work profiles (see Figure 2):

- A *task* is a collection of scenarios which can be combined to perform a complex operation. The processing of a damage event in an insurance company (like a car accident) may for instance consist of the scenarios: register new damage event, request survey of a motor vehicle expert, close case and process payment.
- A *work profile* consists of one or more tasks. Therefore each work profile is a job description for a certain position within the organization under consideration.

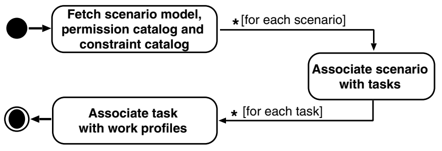
Fetch scenario model, permission catalog and constraint catalog
* [for each scenario]
Associate scenario with tasks
* [for each task]
Associate task with work profiles

Figure 10: Definition of tasks and work profiles

The definition process for tasks and work profiles is by far more complex than the corresponding sub-process depicted in Figure 10 suggests. Like constraints, the specifications for tasks and work profiles are usually very different within diverse organizations and for diverse information systems. Thus in the majority of cases it is inevitable to define them together with domain experts (e.g. a state attorney, a judge and a secretary for a court information system). The most challenging part is to select the correct group of scenarios for a particular task.

4.6 Derivation of a preliminary role-hierarchy

At this stage of the role engineering process enough information has been gathered to build a first version of the RBAC role-hierarchy. Figure 11 depicts the corresponding sub-process. The work profiles and the permission catalog are the starting points to derive the preliminary role-hierarchy. For each work profile we first create a role with the same or a similar name (e.g. Front-Office, Back-Office, or Role144). Since work profiles consist of tasks which again consist of scenarios, we can directly identify all permissions that need to be assigned to a particular role. Remember that we already derived the permissions that are needed to perform the scenarios in a previous step (see Section 4.2).

Now that we have transformed all work profiles into roles that possess permissions we identify potentially redundant roles. That means we are looking for roles which possess exactly the same permissions as one or more other roles (permissions of r_1 = permissions of r_2). These roles are, however, not deleted but marked for later review. We do this since it may sometimes be sensible to have two separate role definitions although they *temporary* possess the same permissions. In the further course of the role-engineering process such roles may be equipped with additional permissions or permissions may be revoked from them. Another possibility is that a common junior-role for two such roles is defined or that one of two equivalent roles may later become the junior-role of the other.

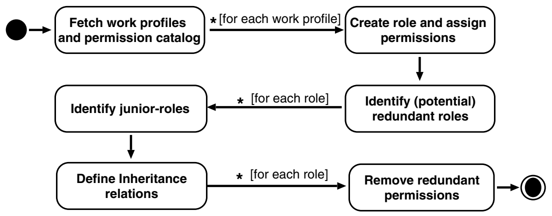
Fetch work profiles and permission catalog
* [for each work profile]
Create role and assign permissions
Identify (potential) redundant roles
* [for each role]
Identify junior-roles
Define Inheritance relations
* [for each role]
Remove redundant permissions

Figure 11: Derivation of a preliminary role-hierarchy from work profiles

However, before the final RBAC role-hierarchy can be defined, the preliminary role-hierarchy has to be build. Therefore the next step is the identification of junior-roles (see Figure 11). In this activity we look for roles whose permissions are a real subset of the permissions assigned to another role. For two roles r_1 and r_2 where the permissions of r_2 compose a subset of the permissions of r_1 we say that r_1 is greater than r_2 ($r_1 > r_2$). After we identified all of those greater than relations, we define an inheritance relation between each two roles r_1 and r_2 were $r_1 > r_2$ applies. Each r_2 is defined as junior-role for the corresponding r_1. Finally we remove the redundant permissions from each role. That means that we remove all permissions that are directly assigned to a role and are also inherited from its junior-roles. When this step is finished we have defined a preliminary role hierarchy.

```
for each work-profile {
  create role and assign permissions
  add role to allRoles
}
for each role1 in allRoles {
  for each role2 in allRoles {
    if {permissions of role1 = permissions of role2} {
      add role1 and role2 to potentiallyRedundantRoles
    }
    if {role1 > role2} {
      add role2 to juniorRoles(role1)
    }
  }
}
for each role in allRoles {
  if {juniorRoles(role) exists} {
    for each jrole1 in juniorRoles(role) {
      for each jrole2 in juniorRoles(role) {
        if {jrole1 > jrole2} {
         delete jrole2 from juniorRoles(role)
      }
    }
  }
}
for each role in allRoles {
  for each jrole in juniorRoles(role) {
    role addInheritanceRelationTo jrole
  }
  role remove redundant permissions
}
```

Figure 12: Pseudo code to derive a preliminary role-hierarchy from work profiles

As you may have already noticed, the derivation of the preliminary role-hierarchy is a good structured process which is suited for a support by a software tool that generates this first version of the role-hierarchy semi-automatically. Figure 12 shows the corresponding algorithm in pseudo code. Here we do only (semi-automatically) define roles as junior-roles if their permissions are a real subset of the respective senior-role. Thus we do not define semantically new relationships. Since the work profiles were already created in accordance with the constraint catalog (see Figure 4), the constraint definitions need not be considered for the derivation of the preliminary role-hierarchy. As described in the following section, constraints are, however, very important to refine the resulting role-hierarchy in the further course of the process.

It has to be mentioned that the preliminary role-hierarchy resulting from the steps described above has in the general case the form of a directed acyclic graph (DAG). From our experience role-hierarchies can be described adequately only through DAGs since tree structures are often not flexible enough to depict real world hierarchies. If due to some other constraints the role-hierarchy has to be in the form of a tree, a further step needs to be executed that transforms the DAG into a tree. Since this has a direct impact on each affected role and the semantics of the RBAC model, the transformation needs to be performed by security engineers and domain experts and may hardly be automated.

4.7 RBAC Model definition

The preliminary role-hierarchy, the permission catalog and the constraint catalog serve as input for the RBAC model definition sub-process. Figure 13 depicts the order of the corresponding activities. Unlike the derivation of the preliminary role-hierarchy, this sub-process must be conducted by security engineers and can only be assisted (and not fully automated) by a software tool.

At first all roles which were previously marked as potentially redundant are reviewed. The security engineers decide together with domain experts which roles are actually redundant and can be removed from the model, and which roles do only temporary have the same access rights and must therefore be kept in the model.

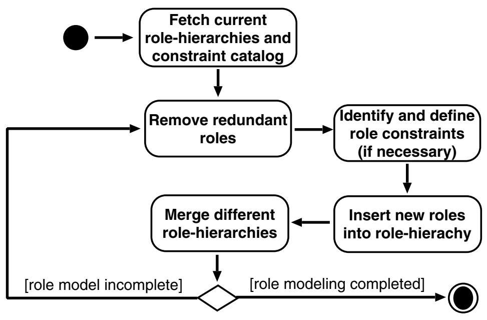
Fetch current role-hierarchies and constraint catalog
Remove redundant roles
Identify and define role constraints (if necessary)
Insert new roles into role-hierachy
Merge different role-hierarchies
[role model incomplete]
[role modeling completed]

Figure 13: Sub-process: RBAC model definition

Until this point in time the constraint catalog contains only constraints on individual permissions (see Section 4.3). As a next step constraints on roles are defined. As already discussed in Section 4.3, the identification of constraints is a complex task that cannot be performed without the input and feedback from domain experts.

As for the definition of permission constraints in step 3 of the role-engineering process (Section 4.3), we must first decide which types of constraints should be modeled (e.g. time-dependencies, cardinalities etc.). Afterwards for each constraint type an own sub-process is started to identify the actual constraints (e.g. the minimum user cardinality for the role Front-Office is two, or the roles Front-Office and Back-Office are mutual exclusive). Subsequent to the completion of the constraint catalog, all information that is needed to define new roles is available to the security engineers. Now we can for example define private roles (see [23]) or interrelate previously independent roles in accordance with the constraint catalog. Moreover one may merge two different role hierarchies by defining a new role whose junior-roles are from different autonomous role-hierarchies. As shown in Figure 13 the four steps are repeated until the RBAC model is complete, i.e. until the security engineers (and domain experts) define the model as adequate.

5. EXPERIENCES: APPLICABILITY AND LIMITATIONS

So far we completed three case studies based on the scenario-driven role engineering process. The first case was a web-based information system for the management of student and alumni communities, while the second was a specialized medical information system for the management and processing of electronic patient records. The third case study was a brokerage platform for the collaborative development and exchange of learning resources among European universities. Research and development activities for this platform are conducted in the UNIVERSAL project [24] which is funded within the IST-program of the European commission. These case studies provided us with significant insights about the scenario-driven role engineering process, its applicability, and possible limitations.

An expected (though important) finding is that one could hardly reach a complete scenario coverage of a system. Nevertheless the same holds for (software) testing of non-trivial systems where a complete test case coverage of the whole system (including requirements, design, components, etc.) is nearly impossible (see e.g. [15, 18]). However, from our experiences we can say that a thoroughly conducted scenario-driven role engineering process where security engineers interact with domain experts leads to promising results. Moreover the definition of a concrete RBAC model is a process with a huge share of social elements. Every organization has a distinct security policy that is tailored to the specific needs and duties of the corresponding organization. Therefore we think that a scenario-driven role engineering process can be a good means to build actual RBAC models.

Changes that may occur when new scenarios are incorporated (e.g. if the system is extended with new functionality) can be integrated straightforwardly (cf. Section 3). The new scenario is added to the scenario model, new permissions and constraints are derived (if necessary), the scenario is assigned to one or more tasks and work profiles, and finally the changes are propagated into the RBAC model.

Another important finding was the suitability of constraints for both permissions and roles and the need to explicitly model each of these types. Before we applied our approach for the first time we were not sure about the "best" point in the process where the different constraint categories should be modeled. We finally chose the straightforward way to model the respective constraints as soon as all necessary information is available. As described in this paper both are distinct and important activities that are a vital part of the role engineering process (see Section 4.3 and Section 4.7).

We found that in the majority of cases it is not possible to reflect all roles of a complex system in a single role-hierarchy, rather we often have the situation that an RBAC model comprises several small hierarchies with approximately ten or less roles and a (small) number of standalone roles or role-hierarchies consisting of only two or three roles. Nevertheless role-hierarchies are well suited to remove redundancies from an RBAC model and thus make a good contribution to an improved maintainability and comprehensibility. In so far we found that real systems often need to be depicted by several rather than one hierarchy. However from our experiences we cannot completely agree with [10] which states that the concept of role-hierarchies usually translates only badly into practical application.

Our final remark relates to the granularity of the permissions which are modeled in the role engineering process and are later assigned to roles. As mentioned in Section 4.2 the scenario-based approach may also facilitate the detection and modeling of abstract and basic permissions. Although we did not encounter any concrete difficulties with the simultaneous modeling and assignment of abstract and basic permissions we are not completely sure about the long-term implications of such a step. We therefore suggest to model both kinds of permissions but to assign only abstract permissions to roles, if possible.

6. RELATED WORK

In [3] Coyne shortly describes a basic approach that could be used to identify RBAC roles. In essence, he suggests to collect different user activities and describe them as verb/object pairs. These activities may then be clustered to define candidate roles. Duplicate candidate roles are deleted, the remaining roles are equipped with the minimal possible set of permissions. In the next step constraints are defined before role-hierarchies can be build.

Fernandez and Hawkins suggest to determine role rights from use cases [7]. In their approach authorizations are derived from the preconditions modeled for a use case. They propose a UML stereotype to extend use case descriptions with additional security requirements. These security requirements are then specified within the textual description of a use case (usually defined in a use case template) by writing an additional comment marked with the keyword "security" (e.g. in the pre- or postcondition slot). The permissions a specific actor (role) needs can then be determined from the use cases this actor participates in. The approach does, however, not describe when and how constraints are elicited, nor does it deal with the definition of role-hierarchies.

In [5] Epstein and Sandhu present an approach to express the different parts of an RBAC model with different UML diagrams. On an example from the health care domain they show how the UML may be applied in principle to document an RBAC model. Nevertheless they do not present a role engineering process or framework, and do therefore not deal with the definition process of constraints and role-hierarchies or the derivation of permissions.

Roeckle et al. suggest a process-oriented approach for role finding [21]. They present the experiences they gathered from a case study conducted at Siemens. They distinguish three different layers, the process layer, the role layer and the access rights layer. At first the business processes of the corresponding organization are modeled (process layer). Then role candidates are identified from these process descriptions and registered in a role catalog (role layer). The role layer is used as input to derive the access rights layer. Roeckle et al. draw the important conclusion, that RBAC roles are closely related to the core (business) functions, and that role engineering itself should therefore revert to the business processes supported by the corresponding information system. However, they describe the process of role finding only on a meta level and do not go into detail about the derivation of permissions, the assignment of permissions to roles or the definition of role-hierarchies or constraints.

In [6] Epstein and Sandhu describe a model for the engineering of role-permission assignments which builds upon the RBAC96 model [23]. They introduce three additional layers between roles and permissions to divide role-permission assignment into smaller and better manageable steps. The new layers are tasks, workpatterns and jobs. In their approach a task represents a specific piece of work (a step) and is associated with the permissions that are required to perform this step. A workpattern is a sequence of tasks. Each task thus represents a specific step within a workpattern. Jobs are mapped to workpatterns. The task to workpattern relation is a many-to-many relation, while the workpattern to job relation is a many-to-one-relation. A role is composed of one or more jobs. Since the focus is on role-permission assignment only, the approach does, however, not go into detail how constraints and role-hierarchies affect the role-permission assignment process and vice versa (e.g. for mutual exclusive permissions, or maximum cardinalities defined on permissions). Nevertheless the approach shows that every single activity of the role engineering process as a whole needs to be further elaborated.

7. CONCLUSION AND FUTURE WORK

The scenario-driven role engineering process supports the definition of a concrete RBAC model and provides adaptability of the resulting models through the straightforward propagation of changes into all affected models. In the course of the process a permission catalog, a constraint catalog and definitions of work profiles are produced. These models/documents serve as the foundation for the definition of the RBAC model which consists of permissions, roles, role-hierarchies and constraints.

The scenario concept is of central significance for our approach. A scenario can be seen as collection of permissions that are applied in a particular order to reach a predefined (user) goal. In order to perform a certain scenario, a subject must therefore own all permissions that are needed to complete every single step of this particular scenario. Once a scenario model is built changes that enrich the functionality of the system can be incorporated straightforwardly. In our model such a change case is characterized through the definition of a new usage scenario. The new scenario is added to the scenario model, new permissions and constraints are derived (if necessary), the scenario is assigned to one or more tasks and work profiles, and finally the changes are propagated into the RBAC model. The deletion of a scenario has the reverse effect.

The defined process provides a systematic approach to role engineering and has already proven its applicability in some case studies. Nevertheless we are continuing to apply the process in other application areas to extend our knowledge of role engineering and to further improve the process definition. This also includes the detection of gaps in the current process definition. For instance in the scenario-driven ap-

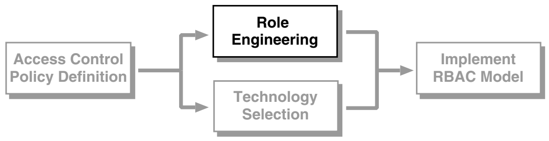

Figure 14: Role engineering context

proach it is difficult to detect system intrinsic permissions (and roles) that do not result from the system's (abstract) functionality but from the chosen technology. For example if a certain system function is implemented through web-based services using CGI scripts or internal web-server functions, additional permissions need to be defined for the corresponding configuration files or scripts. Therefore we further refine and elaborate the role engineering process to extend its comprehensiveness and applicability.

Furthermore Figure 14 suggests that role engineering is of course only one part of a larger process for the definition and implementation of access controls. Therefore the interrelations of the role engineering process and its context need to be further investigated to design a comprehensive life-cycle model for (role-based) access controls. Especially the mapping of an (abstract) access control policy on a concrete RBAC model and the seamless realization with a specific RBAC service are interesting fields for future work. Therefore the definition of a technology selection process that enables the engineers to select an RBAC service that is well suited to implement all aspects of a certain access control policy (e.g. separation of duties) is an important subject.

Like every requirements engineering process, the process of role engineering depends significantly on human factors. Therefore many elements of the process cannot be automated (or at most partially). Nevertheless from our point of view tool support is certainly sensible, since a tool can provide user/process guidance for security engineers, keep track of changing models and remind the security engineer about unfrequent tasks. We are currently developing such a tool for the support of security engineers.

8. REFERENCES

[1] G. Booch, I. Jacobson, and J. Rumbaugh. *The Unified Modeling Language User Guide*. Addison-Wesley, 1999.

[2] J.M. Carroll. Five reasons for scenario-based design. In *Proc. of the IEEE Annual Hawaii International Conference on System Sciences (HICSS)*, 1999.

[3] E.J. Coyne. Role engineering. In *Proc. of the ACM Workshop on Role-Based Access Control*, 1996.

[4] J.M. Carroll (ed.). *Scenario-Based Design: Envisioning Work and Technology in System Development*. John Wiley & Sons, 1995.

[5] P. Epstein and R. Sandhu. Towards A UML Based Approach to Role Engineering. In *Proc. of the ACM Workshop on Role-Based Access Control*, 1999.

[6] P. Epstein and R. Sandhu. Engineering of Role/Permission Assignments. In *Proc. of the 17th Annual Computer Security Applications Conference (ACSAC)*, December 2001.

[7] E.B. Fernandez and J.C. Hawkins. Determining role rights from use cases. In *Proc. of the ACM Workshop on Role-Based Access Control*, 1997.

[8] D.F. Ferraiolo, J.F. Barkley, and D.R. Kuhn. A Role-Based Access Control Model and Reference Implementation within a Corporate Intranet. *ACM Transactions on Information and System Security*, 2(1), February 1999.

[9] D.F. Ferraiolo, R. Sandhu, S. Gavrila, D.R. Kuhn, and R. Chandramouli. Proposed NIST Standard for Role-Based Access Control. *ACM Transactions on Information and System Security*, 4(3), August 2001.

[10] C. Goh and A. Baldwin. Towards a more complete model of role. In *Proc. of the ACM Workshop on Role-Based Access Control*, 1998.

[11] O. Gotel and A. Finkelstein. An analysis of the requirements traceability problem. In *Proc. of the IEEE International Conference on Requirements Engineering (ICRE)*, 1994.

[12] K. Gutzmann. Access control and session management in the HTTP environment. *IEEE Internet Computing*, January/February 2001.

[13] I. Jacobson. *Object-Oriented Software Engineering*. Addison-Wesley, 1992.

[14] M. Jarke, X.T. Bui, and J.M. Carroll. Scenario management: An interdisciplinary approach. *Requirements Engineering Journal*, 3(3/4), 1998.

[15] C. Kaner, J. Falk, and H.Q. Nguyen. *Testing Computer Software (second edition)*. John Wiley & Sons, 1999.

[16] G. Kotonya and I. Sommerville. *Requirements Engineering - Processes and Techniques*. John Wiley & Sons, 1998.

[17] G. Neumann and M. Strembeck. Design and Implementation of a Flexible RBAC-Service in an Object-Oriented Scripting Language. In *Proc. of the 8th ACM Conference on Computer and Communications Security (CCS)*, November 2001.

[18] W.E. Perry. *Effective Methods for Software Testing (second edition)*. John Wiley & Sons, 2000.

[19] B. Ramesh and M. Jarke. Toward reference models for requirements traceability. *IEEE Transactions on Software Engineering*, 27(1), January 2001.

[20] S. Robertson and J. Robertson. *Mastering the Requirements Process*. Addison-Wesley, 1999.

[21] H. Roeckle, G. Schimpf, and R. Weidinger. Process-oriented approach for role-finding to implement role-based security administration in a large industrial organization. In *Proc. of the ACM Workshop on Role-Based Access Control*, 2000.

[22] C. Rolland, G. Grosz, and R. Kla. Experience with goal-scenario coupling in requirements engineering. In *Proc. of the IEEE International Symposium on Requirements Engineering (RE)*, 1998.

[23] R.S. Sandhu, E.J. Coyne, H.L. Feinstein, and C.E. Youman. Role-based access control models. *IEEE Computer*, 29(2), February 1996.

[24] The UNIVERSAL Brokerage Platform Homepage. http://www.ist-universal.org.

[25] A. van Lamsweerde. Goal-Oriented Requirements Engineering: A Guided Tour. In *Proc. of the 5th IEEE International Symposium on Requirements Engineering (RE)*, August 2001.