# Scenario-Driven Role Engineering

Scenario-driven role engineering is a systematic approach for defining customized role-based access control models. Based on his role-engineering experiences, the author discusses the relations between different role-engineering artifacts, the need for process tailoring, and the use of preexisting documents in role-engineering activities.

MARK STREMBECK
*Vienna University of Economics and Business*

In role-based access control (RBAC),[1–3] roles model different job positions and scopes of duty within a particular organization or information system. Organizations assign human users and other "active" entities (or *subjects*) to roles according to their duties, and roles are equipped with the permissions those subjects need to perform their respective tasks. Thoroughly engineered roles tend to change more slowly than do the assignments of individuals to these roles. Thus, establishing roles as an abstraction mechanism for subjects significantly facilitates permissions administration.

With respect to access control, we can differentiate between *functional roles* and *organizational roles*. Functional roles reflect the essential business functions that must be performed within a certain company. Organizational roles, on the other hand, correspond to a company's hierarchical organization in terms of internal structures. Functional roles, in contrast to organizational roles, are robust against organizational restructuring because business tasks often aren't reflected in organizational structures.

Scenario-driven role engineering focuses on defining functional RBAC roles.[4] It provides process guidance for eliciting, specifying, and maintaining the different RBAC-related artifacts. The approach is adaptable and can be tailored for arbitrary organizations and information systems. In recent years, my colleagues and I have gained many experiences and obtained a deeper understanding of scenario-driven role engineering. Here, I examine the relationships among different role-engineering artifacts, the need to tailor the role-engineering process, and how preexisting documents can be used in role-engineering activities.

## Role-Engineering Core Artifacts

As its name indicates, scenario-driven role engineering is based on the scenario concept. In general, a scenario describes a possible or actual action and event sequence. The idea of scenarios has been used since ancient history—for example, to describe and assess alternative business, politics, or war strategies. In the software engineering domain, scenarios model software systems usage and facilitate communication among engineers as well as that between engineers and nontechnical stakeholders.[5,6] So, scenarios are practical means that let us consider the strong human factor in role engineering. We can describe scenarios in various ways. Commonly, they're specified with (structured) text descriptions and different types of diagrams, such as activity diagrams, sequence diagrams, or Petri nets.

Figure 1 shows the main relationships between role-engineering artifacts and corresponding RBAC model artifacts. A *permission* grants a subject the right to perform a specific operation (or operation type) on a specific object (or object type). An object can be a file, a row in a database table, a printer, a network interface, and so on. In access control terminology, we thus define a permission through an ⟨*operation*, *object*⟩ pair. At the modeling level, we can define both operations and objects on any suitable abstraction level.

We can view each action and event within a scenario as a step; each step is typically associated with a

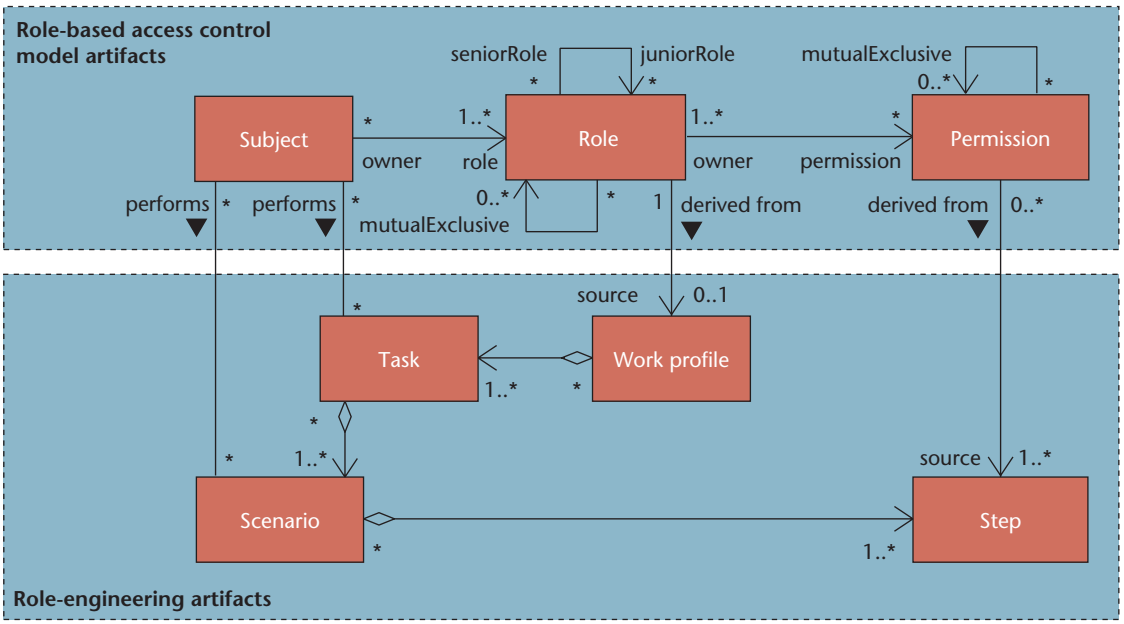


Figure 1. Role-engineering artifacts and role-based access control model artifacts. Role-engineering artifacts are used to define tailored RBAC artifacts for a particular organization or information system.

particular access operation.[4] Therefore, scenarios are a good source for deriving permissions. This means that a subject performing a scenario must own all permissions needed to complete each step of this scenario. Thus, each ⟨*operation*, *object*⟩ pair that we can derive from a scenario step is a candidate permission of the corresponding RBAC model.

Every *task definition* consists of one or more scenarios—for instance, for the task of processing a damage event at an insurance company. Subjects perform the scenarios that a task includes in succession or in parallel to reach a particular goal. A *work profile* contains all tasks that a certain type of employee (or subject in general) can or must perform. Because each scenario is transitively linked to a set of permissions (via the steps included in the scenario), we can derive the permissions for a particular work profile from the corresponding tasks and scenarios (see Figure 1). Thus, we don't assign permissions directly to work profiles, which is an essential difference between work profiles and RBAC roles. Furthermore, because a task might be associated with multiple work profiles, and because each scenario might be associated with multiple tasks, many redundancies might exist in the work profile definitions. This is another important difference between work profiles and RBAC roles, which we can arrange in inheritance hierarchies to potentially minimize redundancies: work profiles serve as the primary source for defining preliminary RBAC roles. Our experiences show that agreed-upon work profiles are a significant and important step toward defining RBAC roles.

Furthermore, because the role-engineering process precisely associates each task definition with the scenarios that subjects must perform to fulfill a specific goal (for example, a certain business function), the corresponding RBAC role can be equipped with the exact number of permissions necessary to perform those tasks. So, scenario-driven role engineering directly supports the principle of least privilege.

In actual role-engineering projects, we store the different artifacts via special-purpose catalogs. The *scenario catalog* comprises all usage scenarios for the system under consideration; the *permission catalog* consists of all permissions identified for a system; the *task catalog* includes the tasks that human users or other subjects perform; and the *work-profile catalog* consists of different work profiles. Each work profile is intended to be a complete description of all tasks that a specific type of subject (for example, a specific type of employee) must or can perform.

## The Scenario-Driven Role-Engineering Process

Figure 2 depicts the control flow for the main activities in scenario-driven role engineering, modeled via a UML activity diagram. Moreover, it shows the different artifacts required as input and produced as output for each corresponding activity. Each main activity defines its own subprocess.[4,7]

First, the *identify and model scenarios* activity explicitly models sensible system usages via scenarios, thereby producing the scenario catalog. Subsequently, the

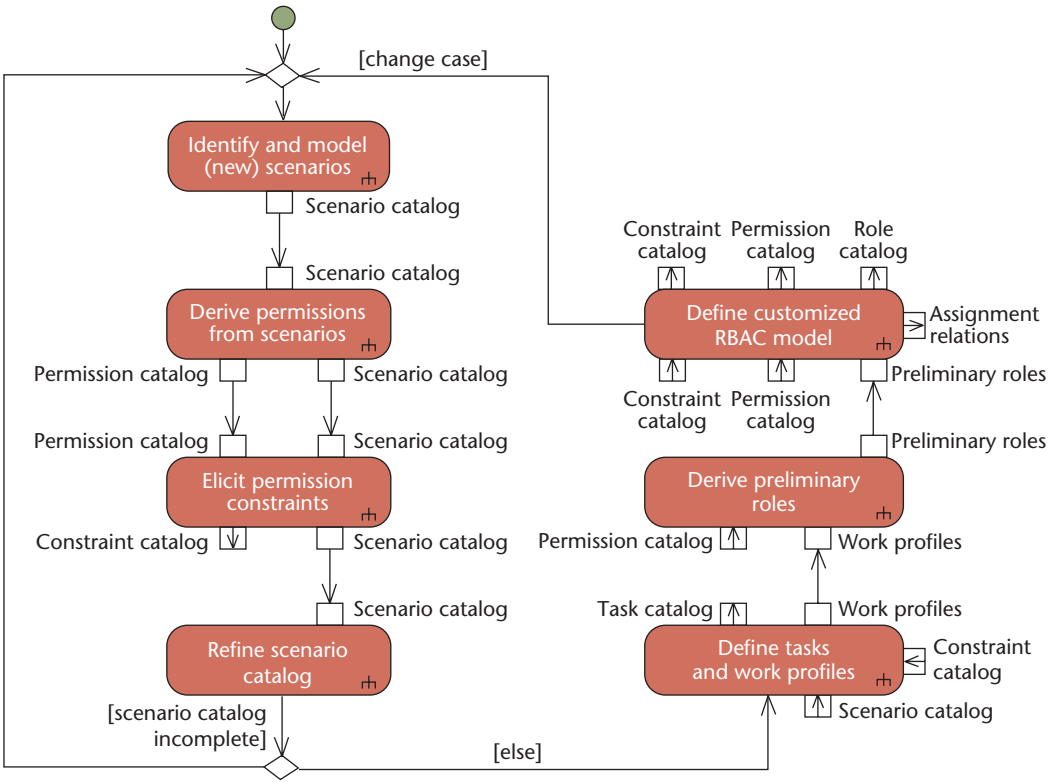


Figure 2. The scenario-driven role-engineering process: main control and object flows. The process can be tailored to fit the needs of a particular role-engineering project. For demonstration purposes, this figure shows a specific process variant.

*derive permissions from scenarios* activity uses the scenario catalog to produce a version of the permission catalog. Next, we *elicit permission constraints* and store all constraints in the *constraint catalog*, which includes constraints that must be enforced on permissions, roles, or assignment relations. However, defining constraints isn't mandatory in scenario-driven role engineering.

In the fourth activity, we *refine the scenario catalog*—specifically, role engineers and domain experts review the current version. Here, a domain expert is a human user who is a professional in a particular domain, such as a stock analyst in the investment banking domain, a physician in the healthcare domain, or a network administrator in the computer networking domain. For similar scenarios, we can define a common generalization, if necessary. In addition, we examine each scenario to determine whether we can further concretize its steps via a subscenario.

After the scenario catalog is complete, we *define tasks and work profiles* (see Figure 2). In this activity, the role engineers compose different scenarios to form task definitions in accordance with the constraint catalog. Subsequently, these tasks serve as building blocks for work profiles. The *derive preliminary roles* activity then uses the work profiles and permission catalog to semiautomatically create preliminary roles. In this activity, we also identify obvious junior and senior roles and define a preliminary role hierarchy. Moreover, we identify potentially redundant roles and mark them for review.[4]

In the seventh activity, the preliminary roles, the permission catalog, and the constraint catalog serve as input to *define a customized RBAC model* for the corresponding organization or information system. In particular, role engineers remove redundant roles, define new roles and role constraints, and merge or separate role hierarchies. We repeat these steps until the customized RBAC model is complete—that is, until engineers and domain experts define the customized RBAC model as adequate.

The resulting system- or organization-specific RBAC model is thus a composite comprising the role catalog, the permission catalog, the constraint catalog, and assignment relations between the different artifacts. The role catalog includes all roles and interrole relations, as defined through role hierarchies, for example. Here, role hierarchies are inheritance hierarchies in which senior roles inherit permissions and constraints (transitively) from all of their junior roles.[1–3]

As Figure 2 shows, activities 1 to 4 form a cycle that's repeated until the scenario catalog is complete. This is a prerequisite for activities 5 to 7. Moreover, the whole process (activities 1 to 7) should be executed

in an iterative and incremental manner, whereby each iteration results in a new evolutionary stage of the different models. For this reason, we develop scenarios, tasks, work profiles, and constraints in close cooperation with domain experts and check and refine them in an evolutionary fashion. From our experiences, cooperating with domain experts significantly increases the respective models' quality.

After building a version of the scenario catalog (that is, after the first iteration of activities 1 to 4), we can incorporate changes in the modeled information system's functionality straightforwardly. Scenario-driven role engineering typically initiates such a change case by modifying an existing scenario or defining a new one. This usually takes place in activity 4 (refinement of the scenario catalog) or after an organization releases a version of the customized RBAC model in activity 7 (see Figure 2). Role engineers can integrate new scenarios with the existing scenario catalog and afterward derive the new permissions (if any) from this scenario. We then assign the scenario to one or more task definitions and work profiles and update the corresponding RBAC model accordingly. We apply these steps analogously in case existing scenarios are modified or deleted. However, as I describe in a later section, you must make preparations to facilitate the correct and efficient propagation of changes into the different models.

### Using Preexisting Documents and Process Descriptions

Because role engineering is a complex and time-consuming task, preexisting documents and process descriptions might significantly ease the role-engineering process. Such documents might exist on different abstraction levels and range from business process models and job descriptions, over (technical) documentation of the corresponding information systems, to a preexisting role or permission catalog. Even if such preexisting models and documents are incomplete—in that they aren't sufficient for role engineering—they're still a valuable input for the role-engineering process. The preexisting documents then serve as a starting point, and we can further refine and complement them as the process proceeds.

For example, in previous work, my colleagues and I introduced an approach to derive RBAC models from Business Process Execution Language (BPEL) processes.[8] By extracting information from preexisting BPEL processes (or other machine-readable formats), we can make role engineering more efficient by automating several steps. Furthermore, because RBAC artifacts and process models are highly interrelated, automation in role engineering facilitates consistency between deployed business processes and their corresponding RBAC artifacts.

Similar to BPEL processes, we could use other types of preexisting models in role engineering. However, most preexisting models and documents aren't suited to directly automating some steps of the role-engineering process. Nevertheless, even if such models are available only as text descriptions or in other formats we can't use for automation, they're still an important input for different role-engineering activities. For example, event-driven process chains (EPCs) or UML activity diagrams can serve as a starting point for scenario refinement and for deriving permissions. Other types of scenarios that we often find in organizations are narrative text instructions describing certain tasks. Engineers can then translate such narrative scenarios into more formal scenario or process descriptions—for example, using UML activity and interaction diagrams.

### Tailoring the Role-Engineering Process

Various factors affect role-engineering projects, so it's sensible to tailor the role-engineering process to its environment rather than apply a "one size fits all" process for each organization and information system. Here, tailoring typically includes adding or removing (sub)activities and adapting the process's control flow. For example, we can tailor this control flow so that we first identify preliminary tasks and define the scenarios in a subsequent activity. Thus, we can adapt the role-engineering process to a certain project's or organization's characteristics.

We determine the project's characteristics via several factors:

- The *preexisting artifacts* consist of the models, job descriptions, and documentation artifacts that serve as input for the role-engineering process.
- The *IT system properties* determine the corresponding information system's characteristics as well as those of the respective access control subsystem. We need this information to ensure that the customized RBAC model can actually be mapped to and enforced by the corresponding software system.
- The factor *involved people* determines which people (for example, security experts, software engineers, or experts from the system's application domain) are needed for a project and which will actually be available during the project.
- The *budget* determines the monetary budget for the role-engineering project (including costs for human resources).
- The *project deadlines* determine the project's time frame. At a project's start, we typically have only a few predetermined deadlines—for example, a (tentative) deadline for the customized RBAC model's release date. Depending on the project, such predetermined deadlines are complemented via other deadlines—for instance, for the release cycles of alpha and beta versions of the RBAC model.

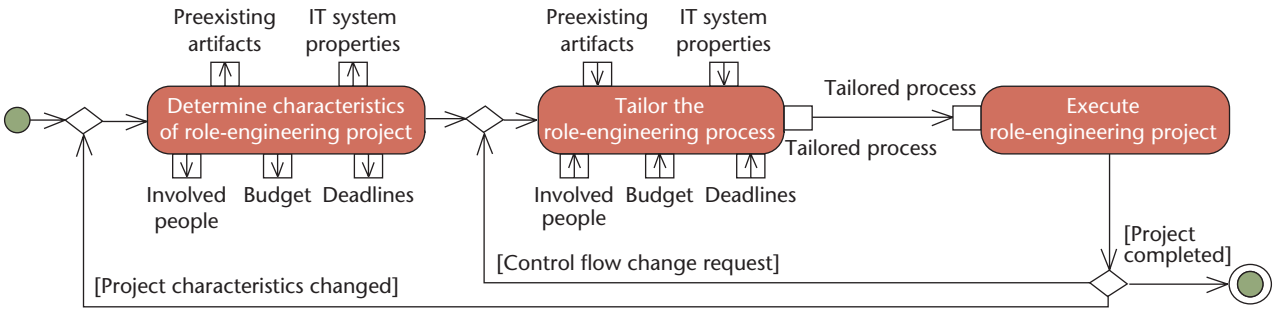


Figure 3. Tailoring the role-engineering process. We can adapt a tailored process if we encounter significant changes in the project characteristics or if the engineers decide that adapting the control flow better suits the respective project's needs.

Figure 3 shows the general activity flow that we use to tailor the role-engineering process or to dynamically adapt that tailored process. In particular, we can adapt a tailored process if we encounter significant changes in the project characteristics (for example, if a domain expert believed to be available daily is in fact available only every two weeks) or if the engineers decide that adapting the control flow better suits the respective project's needs.

Typical examples of role-engineering activities we might remove during process tailoring are the definition of constraints or role hierarchies. One reason for this is that access control subsystems don't always support the definition of role hierarchies or can't (yet) enforce constraints. Another reason is that engineering constraints is very time-consuming and requires a specifically tailored subprocess for each constraint type.

However, from our experiences, good reasons exist to model constraints even if we can't yet enforce them on a technical level. The aim to specify and maintain a comprehensive, and preferably complete, customized RBAC model is probably the most important reason. Such a "complete" model provides valuable information for the corresponding security engineers. For example, those engineers can then identify which subset of an organization's customized RBAC model can be enforced by the runtime system and which control objectives aren't yet achievable. We can apply this information to thoroughly configure the respective system and avoid security breaches that could result from unavailable information. Furthermore, a complete description of an RBAC model on the requirements and design levels can drive RBAC services' technical evolution to close the gap between a customized RBAC model and its enforceable subset.

An option that, from our experience, works well in role-engineering projects is to define a basic customized RBAC model in a first project and refine this model in one or more follow-up projects. For example, in an initial project, we could define an RBAC model that includes roles and permissions only and add different constraint types in a second project.

## Definition of Constraints

In scenario-driven role engineering, defining constraints is an optional activity that you can skip when tailoring the engineering process. However, if constraints are modeled, we recommend defining them on the lowest possible abstraction level. This means that you should first try to define constraints on the permission level and specify them on the role level only if you can't sensibly define them on the former. In our experience, this measure eases constraint management because constraints significantly raise the complexity of the assets they're assigned to (on a logical and on the implementation level).

We basically distinguish two categories of constraints: *endogenous constraints* are those that completely relate to an RBAC model's intrinsic properties (such as roles or permissions) and inherently affect a customized RBAC model's structure and construction (such as separation of duty constraints). *Exogenous constraints* are those that either exclusively involve attributes that don't belong to a model's core elements (for example, time constraints) or that refer to a specific model element's external attributes or properties (such as a subject's geographical location or current project assignment).[4,7]

As mentioned, constraint definition is a time-consuming activity, and each constraint type requires a specifically tailored engineering subprocess. We can often identify separation of duty constraints, for example, on the level of task definitions; we then derive corresponding permission-level constraints from the corresponding task-level exclusions.

In prior work, we presented a subprocess for engineering *context constraints*.[7] Such contraints are an RBAC extension to model and enforce context-dependent access control policies. Thus, context constraints are exogenous constraints that we define to consider contextual information (such as time, location, or access history) in authorization decisions. A

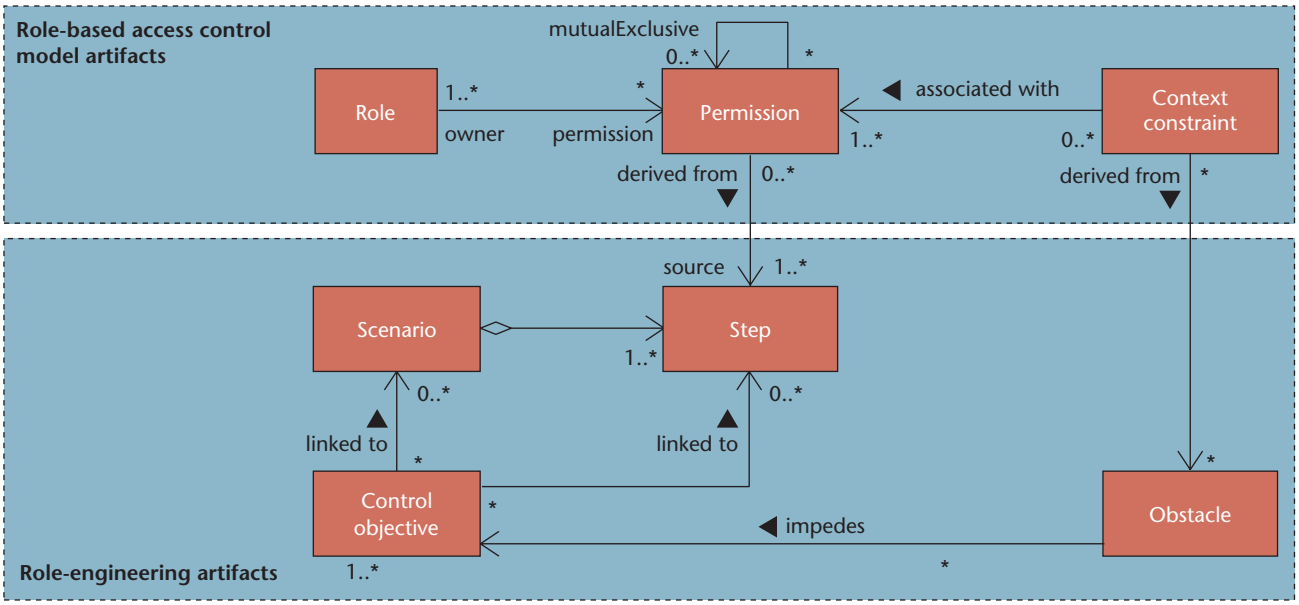


Figure 4. Context constraints and related artifacts. We use context contraints to define conditional permissions that consider contextual information in authorization decisions.

*conditional permission* is one that's associated with one or more context constraints (see Figure 4) and grants access if each associated context constraint is fulfilled.

Like the role-engineering process as a whole, context-constraint engineering is based on requirements-engineering techniques. In particular, we use goals and obstacles to elicit context constraints. The goal catalog thus contains the control objectives and obstacles that we derive from or associate with the corresponding scenarios (see Figure 4). Here, a control objective is a goal specified by the authority that manages a particular information system.

Thus, control objectives define acceptable system behavior that the system authority intends. In contrast, we define an individual goal as a goal representing a subject's intentions when using an information system. However, individual goals don't necessarily conform to the control objectives defined for a system and might even be contrary to them.[9] This is especially true for malicious individual goals, such as attempts to hack or crack a system, deliberately circumvent protection measures, or use system functions in an unintended manner—for example, spy on another subject's behavior. So, (malicious) individual goals can be obstacles impeding the control objectives defined for a system and thereby impeding the whole system's correct operation (see Figure 4).

## Traceability, Maintenance, and Tool Support

To efficiently deal with change and maintenance requests for role-engineering artifacts, we record specific trace information. We define traceability as the ability to describe and follow the life of an artifact used or produced in engineering projects (for example, scenarios, permissions, or work profiles) in both a forward and backward direction.[10] This definition implies that we can understand each artifact's life from its origin, through its evolutionary refinement and specification, to its subsequent deployment and use. Trace management is very important for efficiently handling evolving complex models of all kinds.[10,11]

The interrelations that Figures 1 and 4 depict indicate the trace relations that are implicitly defined and maintained when conducting the scenario-driven role-engineering process. We can use these traces, for example, to easily review which permissions are needed in a particular scenario as well as to find all scenarios—and therefore all tasks and work profiles—in which a specific permission is used. However, tool support is necessary to cope with the role-engineering process's complexity and to efficiently handle the different interrelated artifacts used and produced during the process. In previous work, I described the design and implementation of the xoRET software tool, which directly supports the scenario-driven role-engineering process.[12] xoRET is a graphical software tool that facilitates the specification and inspection of trace relations to ease change-management activities. Moreover, it implicitly records traces between role-engineering artifacts and provides functions for explicitly defining and inspecting (additional) traces between the different artifacts. Thus, it supports engineering adaptable models that facilitate the incor-

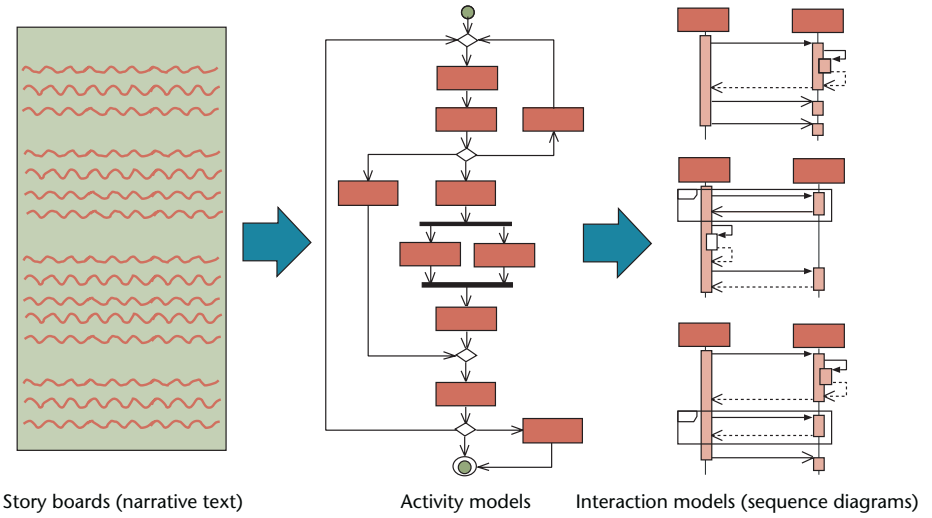


Figure 5. Scenario refinement and concretion. We first define storyboards consisting of narrative text; then, we model these storyboards via UML activity models and use UML sequence diagrams to describe interactions in detail.

poration of changes into a configuration of several related models.

## Scenario-Driven Role Engineering in Practice

Since its first publication in June 2002,[4] numerous consulting firms and international projects have adopted the scenario-driven role-engineering process. The most visible of these is probably the Health Level 7 (HL7) role-engineering process defined by the US National Healthcare RBAC Task Force.[13,14] Among other things, the task force applied this process to produce HL7 RBAC healthcare scenarios and an HL7 RBAC healthcare permission catalog.

The permission catalog, scenarios, and other documents are publicly available on the task force's Web page (www.va.gov/RBAC/). I recently conducted an assessment project together with a subdivision of the Austrian Federal Ministry of Finance (http://english.bmf.gv.at/) to build a preliminary role catalog for this subdivision and to estimate the costs and timeframe for a role-engineering project for other subdivisions. In the course of this project, we also assessed which existing documents and process descriptions the ministry can use as an input for the role-engineering process.

In addition to such projects and our previous case studies, other researchers and I have conducted several smaller case studies with a research focus in recent years. Our main goal was to gain further experience with the role-engineering process, verify previous findings, and evolutionarily enhance the process. We conducted two of our case studies in the context of EducaNext, a brokerage platform for the collaborative development and exchange of learning resources among European universities. Researchers conducted R&D activities for this platform in a project funded within the European Commission's Infomation Society Technologies (IST) program. We conducted another case study on a subsystem of the Learn@WU Web-based learning environment at the Vienna University of Economics and Business Administration (WU Vienna). With 27,000+ registered users, 4,600+ courses, and 60,000+ learning resources, it's probably one of the most intensely used e-learning environments worldwide.

Three different teams conducted the case studies, each consisting of three graduate students with no previous role-engineering experience. We chose this setup to study how different teams learn the role-engineering process and to complement the process based on our observations. A senior role engineer trained and guided each team throughout the case studies. At the beginning of each study, the teams received basic training for the different role-engineering activities I described previously. During the project, the teams could request additional on-demand training for specific techniques (such as scenario modeling) or process guidance for role-engineering activities (such as permission derivation). However, aside from such on-demand training and process guidance, the senior role engineer's main task was to monitor the project and act as an external observer.

In particular, these case studies further improved our knowledge of process tailoring and the iterative refinement of role-engineering models as well as how to use preexisting documents in role-engineering tasks. Moreover, we verified our previous finding that it's sensible to use three basic scenario-modeling techniques in role engineering. Given that scenario descriptions might differ because of different modeling notations, we use narrative text, activity diagrams,

and sequence diagrams as standard means for scenario modeling and refinement, which minimizes variations that might result from these differences. In particular, we first define storyboards consisting of narrative text; then, we model these storyboards via UML activity models, and, finally, we use UML sequence diagrams to describe interactions in detail (see Figure 5).

Activity models are especially well suited to model various scenario alternatives (for example, branches resulting from different choices) in the same model, whereas the sequence diagrams each model a particular path through the activity model or define a certain subactivity in detail. Storyboards, activity models, and sequence diagrams thereby represent different abstraction levels, supporting an iterative and incremental refinement and understanding of scenario models. Moreover, if preexisting models or process descriptions are available, we can review and use them as input to role-engineering activities.

An observation from our projects and case studies is that standard operating procedures, processes, work profiles, roles, and permission assignments often aren't documented, and existing documents are sometimes incomplete or out of date. Moreover, sometimes such information is buried deep in the current software system's configuration, or only specific individuals are aware of certain procedural details. Thus, the first activity at the beginning of a role-engineering project is to assess and consolidate existing documents before we define additional scenario and process models to build a consistent scenario catalog.

Communication between domain experts and role engineers is a key factor for the success of role-engineering projects. Inexperienced teams often underestimate the level of effort and the timeframe for a project. In particular, we learned through our experiences that such false estimations result from a false perception of role-engineering activities because the process appears straightforward at first glance. However, although scenario and process modeling might seem to be a simple task, it's complex to elicit and define an actual system or organization's scenarios and define a customized RBAC model from this information.

Nevertheless, scenarios are a natural means nontechnical stakeholders can easily learn. Throughout our projects and case studies, we received positive feedback concerning the "dual use" of scenarios as communication and primary engineering vehicles. Thus, from our experiences, we can say that scenarios are well suited to enable close cooperation with domain experts and incorporate them into the engineering process. □

### References


1. D.F. Ferraiolo and D.R. Kuhn, "Role-Based Access Controls," *Proc. 15th Nat'l Computer Security Conf.*, NIST, 1992, pp. 554–563; http://csrc.nist.gov/groups/SNS/rbac/documents/ferraiolo-kuhn-92.pdf.
2. R.S. Sandhu et al., "Role-Based Access Control Models," *Computer*, vol. 29, no. 2, 1996, pp. 38–47.
3. D.F. Ferraiolo, D.R. Kuhn, and R. Chandramouli, *Role-Based Access Control*, 2nd ed., Artech House, 2007.
4. G. Neumann and M. Strembeck, "A Scenario-Driven Role-Engineering Process for Functional RBAC Roles," *Proc. 7th ACM Symp. Access Control Models and Technologies* (SACMAT 02), ACM Press, 2002, pp. 33–42.
5. J.M. Carroll, ed., *Scenario-Based Design: Envisioning Work and Technology in System Development*, John Wiley & Sons, 1995.
6. M. Jarke, X.T. Bui, and J.M. Carroll, "Scenario Management: An Interdisciplinary Approach," *Requirements Eng. J.*, vol. 3, nos. 3–4, 1998, pp. 155–173.
7. M. Strembeck and G. Neumann, "An Integrated Approach to Engineer and Enforce Context Constraints in RBAC Environments," *ACM Trans. Information and System Security*, vol. 7, no. 3, 2004, pp. 392–427.
8. J. Mendling et al., "An Approach to Extract RBAC Models from BPEL4WS Processes," *Proc. 13th IEEE Int'l Workshop Enabling Technologies: Infrastructures for Collaborative Enterprises* (WETICE 04), IEEE CS Press, 2004, pp. 81–86.
9. M. Strembeck, "Embedding Policy Rules for Software-Based Systems in a Requirements Context," *Proc. 6th IEEE Int'l Workshop Policies for Distributed Systems and Networks* (POLICY 05), IEEE CS Press, 2005, pp. 235–238.
10. O. Gotel and A. Finkelstein, "An Analysis of the Requirements Traceability Problem," *Proc. IEEE Int'l Conf. Requirements Eng.* (ICRE 94), IEEE CS Press, 1994, pp. 94–101.
11. B. Ramesh and M. Jarke, "Toward Reference Models for Requirements Traceability," *IEEE Trans. Software Eng.*, vol. 27, no. 1, 2001, pp. 58–93.
12. M. Strembeck, "A Role Engineering Tool for Role-Based Access Control," *Proc. 3rd Symp. Requirements Eng. for Information Security* (SREIS 05), 2005; www.sreis.org/SREIS_05_Program/full7_strembeck.pdf.
13. E.J. Coyne et al., "Role Engineering in Healthcare: Process, Results, and Lessons Learned," Dec. 2004, www.va.gov/rbac/.
14. E.J. Coyne and J.M. Davis, *Role Engineering for Enterprise Security Management*, Artech House, 2008.



***Mark Strembeck*** *is an associate professor of information systems at the Vienna University of Economics and Business (WU Vienna). His research interests include access control, role engineering, secure business systems, model-driven development, and the modeling and management of dynamic software systems. Strembeck has a PhD and a Habilitation degree (venia docendi) from WU Vienna. Contact him at mark.strembeck@wu.ac.at.*