Deriving Role Engineering Artifacts from Business Processes and Scenario Models

Anne Baumgrass, Mark Strembeck
Institute of Information Systems, New Media Lab
Vienna University of Economics and Business
(WU Vienna), Austria
{firstname.lastname}@wu.ac.at

Stefanie Rinderle-Ma
Workflow Systems and Technology
Faculty of Computer Science
University of Vienna, Austria
stefanie.rinderle-ma@univie.ac.at

ABSTRACT

Scenario-driven role engineering is a systematic approach to engineer and maintain RBAC models. Such as every engineering process, this approach heavily depends on human factors and many of the corresponding engineering tasks must be conducted manually. However, based on the experiences we gained from our projects and case studies, we identified several tasks in role engineering that are monotonous, time-consuming, and can get tedious if conducted manually. These tasks include the derivation of candidate RBAC artifacts from business processes and scenario models. In this paper, we present an approach to automatically derive role engineering artifacts from process and scenario models. While our general approach is independent from a specific document format, we especially discuss the derivation of role engineering artifacts from UML activity models, UML interaction models, and BPMN collaboration models. In particular, we use the XMI (XML Metadata Interchange) representation of these models as a tool- and vendor-independent format to identify and automatically derive different role engineering artifacts.

Categories and Subject Descriptors

D.4.6 [**Operating Systems**]: Security and Protection—*Access Controls*; K.6.5 [**Management of Computing and Information Systems**]: Security and Protection

General Terms

Security

Keywords

Role Engineering, RBAC, UML, BPMN, XMI

1. INTRODUCTION

In recent years, role-based access control (RBAC) [8, 9, 22] – together with various extensions – has developed into a de facto standard for access control. In the context of RBAC, roles model different job-positions and scopes of duty within a particular organization or within an information system. *Scenario-driven role engineering* is a systematic approach for defining customized RBAC models, including roles, role-hierarchies, permissions, and constraints [23, 24, 26]. Since its first publication in 2002 [18], we gained many experiences with scenario-driven role engineering and the approach has been adopted by a number of consulting firms and international projects (see, e.g., [5, 15]).

1.1 Scenario-driven Role Engineering

In scenario-driven role engineering, we use scenario and process models as a primary communication and engineering vehicle. We model usage scenarios of an information system and use the respective scenario and process models to derive permissions. In general, a scenario describes a possible or actual action and event sequence (see, e.g., [13]). Thus, to perform a certain scenario, a subject needs to be equipped with the exact number of permissions that are needed to complete each step of the respective scenario. After deriving the permissions, we therefore group the scenarios to form tasks and work profiles. These work profiles serve as preliminary roles and are an important step toward the definition of a customized RBAC model (for details see [18, 23, 24, 26]). Figure 1 shows the main relations between role engineering artifacts and corresponding RBAC model artifacts.

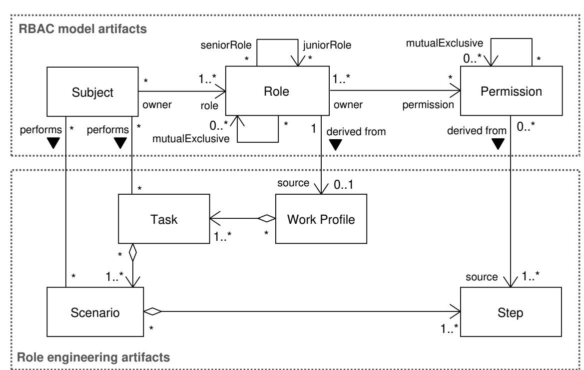

Figure 1: Role engineering and RBAC artifacts [24]

Such as every engineering process, the process of role engineering significantly depends on human factors. For this reason, many steps of the process cannot be automated (or at most partially). However, based on the experiences we gained from our role engineering projects and case studies (see Section 4), we identified several tasks in role engineering that are monotonous, time-consuming and can get tedious if conducted manually. These tasks include the derivation of candidate RBAC artifacts from scenario and process models. In this paper, we are especially concerned with the derivation of role engineering artifacts from UML activity, UML interaction, and BPMN collaboration models.

1.2 Approach Synopsis

In order to ease scenario-driven role engineering, we aim to automate the derivation of role engineering artifacts from different types of scenario and process descriptions. In general, scenario and process models can be defined in a wide variety of (modeling) languages, such as Unified Modeling Language (UML) activity and interaction models, event-driven process chains (EPCs), Business Process Model and Notation (BPMN) models, or via the Business Process Execution Language (BPEL). To automate the derivation of role engineering artifacts, we therefore chose an approach that is independent of the language which is used to define the scenarios and processes. In particular, we first assess the respective (modeling) language and specify a mapping between modeling language artifacts and role engineering artifacts. This mapping especially results in an integrated meta-model (see Figure 2). Based on this integrated meta-model we built a tailored analyzer component that extracts role engineering artifacts from corresponding scenario and process models.

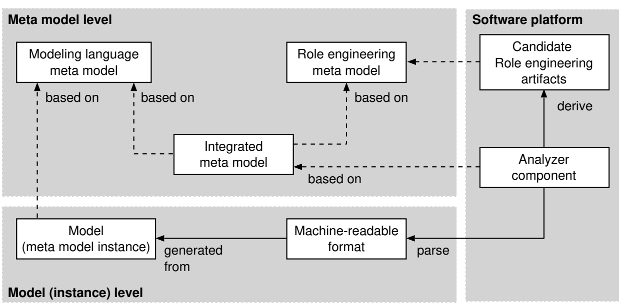

Figure 2: Generic approach for the automated derivation of role engineering artifacts from scenario and process models: conceptual overview

Thereby, our approach is independent of a certain modeling language or format. However, a detailed and dedicated investigation of different modeling languages is essential since modeling language meta models partly differ with respect to the representation of relevant artifacts (see Sections 2 and 3).

In this paper, we describe the derivation of role engineering artifacts from UML activity and interaction models as well as BPMN collaboration models. In particular, we use the XML Metadata Interchange (XMI) [20] representation of these models as a tool- and vendor-independent format to identify and derive different candidate role engineering artifacts.

The remainder of this paper is structured as follows. In Section 2, we give an overview of the different UML and BPMN models, and show how we use them for scenario and process modeling in the role engineering context. Subsequently, Section 3 presents our approach for the automated derivation of role engineering artifacts from the corresponding scenario and process models. Next, Section 4 discusses the practical relevance of our approach. Section 5 gives an overview of related work, and Section 6 concludes the paper.

2. USING UML AND BPMN FOR SCENARIO AND PROCESS MODELING

UML is a de facto standard for the definition of software-based systems. In scenario-driven role engineering, we use UML activity and interaction models as standard means to visualize scenario and process models (see also [24]). Moreover, in recent years BPMN emerged as a new standard for the definition of process models that was quickly adopted in both research and industry.

2.1 UML Activity Models

Activity models specify processes and define the control and object flow between different actions. Figure 3 shows an excerpt of the UML2 meta-model that depicts selected elements of activity models (see [21]). In Section 3.1, we will use some of these activity elements to derive role engineering artifacts.

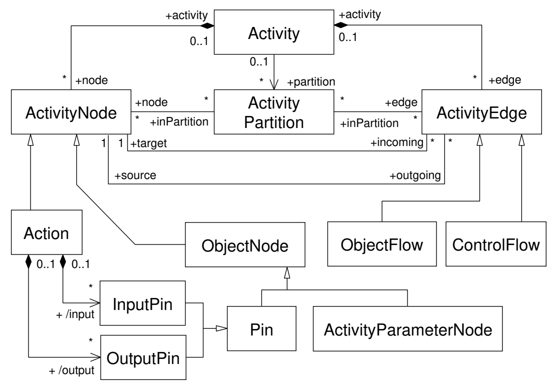

Figure 3: Selected elements of UML2 activity models

The left-hand side of Figure 4 shows the example of a simple credit application process modeled as UML activity diagram. An activity model may include (sub)partitions, and each partition may have a name. Partitions can be used to group actions that have common characteristics, for example the execution of all actions in a partition by the same actor. The example from Figure 4 includes three partitions using the so called swimlane notation (see [21]), the partitions are named "Credit Application Web-Frontend", "Bank Clerk A", and "Bank Clerk B".

Activity models have a token semantics, similar (but not equal) to petri nets. In general, two different types of tokens can travel in an activity model. Control tokens are passed along control flow edges and object tokens are passed along object flow edges (for details see [21]). To model object flows between actions, one uses corresponding object nodes. Pins are a specific type of object node and are visualized as small rectangles that are attached to action symbols. For example, in Figure 4 we have an object flow between the two actions "Negotiate contract" and "Approve contract". The object flow connects the two pins attached to the respective actions and accepts object tokens of type "Contract".

Each edge may be associated with a so called "guard" condition. The guard determines if a particular token is allowed to travel along the respective edge. A decision node is represented by a diamond-shaped symbol and has one incoming and multiple outgoing edges. A merge node is represented by a diamond-shaped symbol and has multiple incoming and one outgoing edge.

The right-hand side of Figure 4 shows an excerpt of the XMI representation of the activity model depicted on the left-hand side. The XML Metadata Interchange (XMI) specification (see [20]) defines an interchange and storage format and (among other things) allows for the transformation of graphical UML models to a generic (tool- and vendor-independent) model representation. Each element in an XMI document has an identifier defined through the `xmi:id` attribute. Via this identifier elements can reference other elements (see below). For demonstration purposes, Figure 4 highlights two

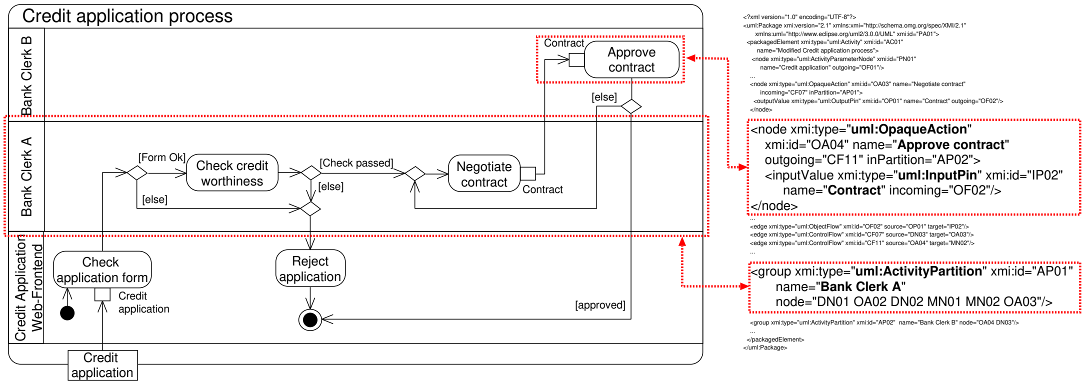

Figure 4: Example of an UML2 activity model and its XMI representation

areas of the activity model and the corresponding XMI representation, in particular:

- In the XMI representation, activity partitions are defined via a `group` element with the `xmi:type` attribute set to `uml:ActivityPartition`. Nodes/elements included in a partition are referenced via the `node` attribute. For example, in Figure 4 the partition with `name` "Bank Clerk A" includes the nodes DN01, OA02, DN02, MN01, MN02, and OA03[1].

- Actions are defined as `node` of the activity model and include an `xmi:type` attribute that specifies the corresponding action type. The action highlighted on the right-hand side of Figure 4 is of type `uml:OpaqueAction` and has the `name` "Approve contract". The `inPartition` attribute references the id of the activity partition that includes the respective action (see also Figure 3). In this example, the "Approve contract" action is included in partition `AP02` (short for "Activity Partition 02") named "Bank Clerk B". Moreover, this action is connected to an InputPin that has the `name` "Contract".

- Input pins as well as output pins (see Figure 3) are defined as subelements of the action they are attached to. Input pins are included in an `inputValue` element and their `xmi:type` is set to `uml:InputPin` (see Figure 4). Likewise, output pins are `outputValue` elements of the type `uml:OutputPin`.

2.2 Refining/Concretizing Activity Models via Interaction Models

While activity models describe the control flows and object flows between different actions on a higher abstraction level, interaction models are used to define the interactions of different actors in detail. Figure 5 shows an excerpt of the UML2 meta-model that depicts selected elements of interaction models (see [21]). In Section 3.1, we will use some of these interaction elements to derive role engineering artifacts.

In particular, interaction models describe a sequence of messages that are send between different lifelines. Here, a lifeline represents an actor that is participating in a particular interaction. In general, an actor may be a human user or a technical (software-based) system. UML includes different (sub)types of interaction models (see [21]). In scenario-driven role engineering, we especially use UML sequence diagrams to model interactions and to specify the actions modeled in an activity model in detail (see also [24]). The right-hand side of Figure 6 shows an example of a sequence diagram

[1]DN = Decision Node, MN = Merge Node, OA = Opaque Action

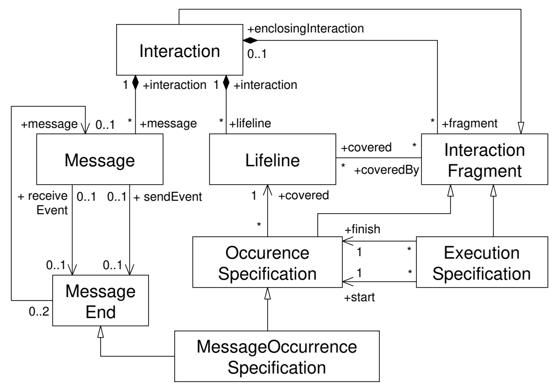

Figure 5: Selected elements of UML2 interaction models

which describes the "Check credit worthiness" action from Figure 4 in detail.

The Interaction from Figure 6 includes three lifelines, representing a "Bank Clerk", a (sub)system called "CustomerMgmt" and a (sub)system called "CustomerRating". Each message send between the lifelines defines a particular act of communication and is modeled via a directed edge pointing from the message sender's lifeline to the message receiver's lifeline. In UML, the start and end of the directed edges representing a message are called MessageEnds and define a so called MessageOccurrenceSpecification (see Figure 5), i.e. the occurrence of a respective send or a receive event on the corresponding lifeline. Asynchronous messages have an open arrow head, synchronous messages have a filled arrow head, and reply messages are drawn as a dashed line with an open arrow head (see Figure 6). Moreover, so called "execution specifications" specify the execution of a certain behavior or command within a lifeline (i.e. the execution of a behavior by the corresponding actor). ExecutionSpecifications are represented by thin rectangles on the lifeline, and may be nested/overlapping. Thus, execution specifications define when an actor (represented via a lifeline) is busy.

Moreover, interaction models may include CombinedFragments. A combined fragment models an interaction fragment which occurs in case a certain condition becomes true. In general, different types of CombinedFragments exist, e.g. to model alternative behaviors, optional behavior, loops, or breaking scenarios (for details see [21]). The example from Figure 6 includes a CombinedFragment modeling an optional behavior (indicated by the "opt" operator in the upper left corner of the fragment) that is executed if the "[decision is positive]" condition in the CombinedFragment evaluates to true.

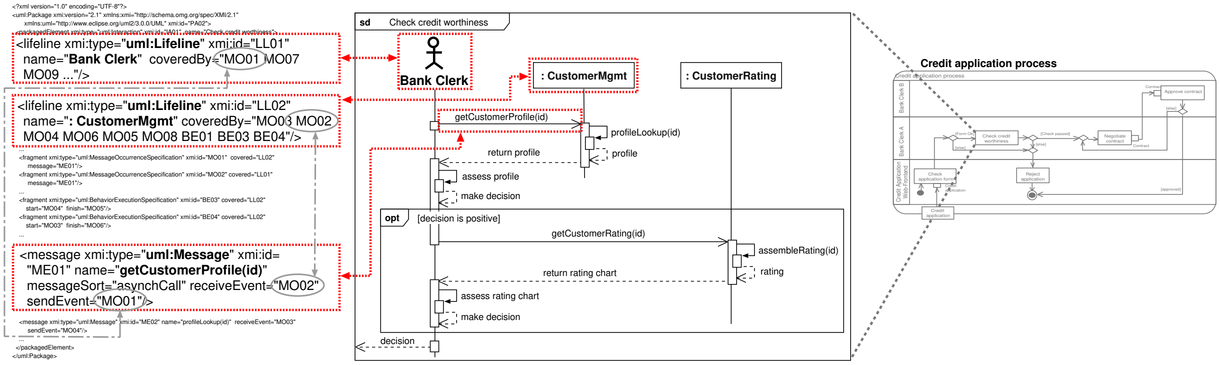

Figure 6: Example of an UML2 interaction model and its XMI representation

The left-hand side of Figure 6 shows an excerpt of the XMI representation of the interaction model depicted on the right-hand side. For demonstration purposes, Figure 6 highlights three areas of the interaction model and the corresponding XMI representation. In particular, the highlighted areas include the following elements:

- In the XMI representation, lifelines are defined via a `lifeline` element with the respective `xmi:type` attribute set to `uml:Lifeline`. Moreover, each lifeline includes a `coveredBy` attribute which contains id-references to the occurrence specifications of this particular lifeline. In the topmost highlighted area from Figure 6, the lifeline with `name` "Bank Clerk" is covered by a number of message occurrences ("MO01, MO07, MO09, ...")[2].
- A Message is defined via a `message` element with the corresponding `xmi:type` attribute set to `uml:Message`. Moreover, the `sendEvent` and `receiveEvent` attributes refer to the respective MessageOccurrenceSpecifications (see also Figure 5) that define the start and end points of a certain message. In the example from Figure 6, the message with `name` "getCustomerProfile(id)" connects the `sendEvent` "MO01" (which is covered by the lifeline "Bank Clerk") and the `receiveEvent` "MO02" (which is covered by the lifeline "CustomerMgmt").

2.3 BPMN Collaboration Models

BPMN2 provides three diagram types named Process, Collaboration, and Choreography respectively (for details see [19]). For our purposes, we especially focus on BPMN collaboration diagrams which model interactions between different entities (so called Participants).

Figure 7 shows an excerpt of the BPMN meta-model that depicts selected elements of BPMN collaboration models. In Section 3.2 we will use some of these elements to derive role engineering artifacts. Figure 8 shows an example of a descriptive BPMN model.

In a collaboration a participant is responsible for the execution of the process enclosed in a so called pool. In BPMN processes group the flow or sequence of different process steps. The steps within a process are categorized and organized via Lanes encapsulated by a LaneSet, whereas each lane can consist of sub-lanes to further partition the included process steps (for details see [19]). In version 2.0, BPMN introduces process modeling conformance classes to simplify the interchange between modelers and developers [19]. Below, we use the descriptive conformance class. The *Descriptive Conformance* class allows to establish a high-level understanding between modelers. The *Common Executable Conformance* class enables a detailed definition of the corresponding processes.

[2]MO = Message Occurrence

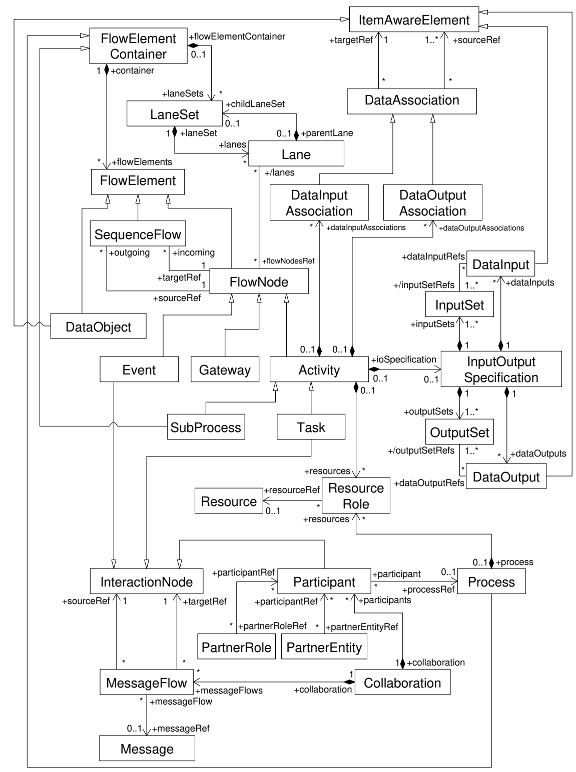

Figure 7: Selected elements of BPMN 2.0 collaboration models

Figure 8 shows our credit application example modeled via a BPMN model using the descriptive conformance class. It includes one Pool named "Bank Company" consisting of the three lanes named "Credit Application Web-Frontend", "Bank Clerk A", and "Bank Clerk B".

While BPMN tasks are atomic process steps, subprocesses can be broken down to a finer level of detail. In Figure 8, the process step "Check credit worthiness" represents a collapsed subprocess, while all other steps in this figure are tasks. The control flow in a process is defined via events and gateways. Gateways coordinate the direction and choices of the process flow, while events can directly affect this flow. Our example process shows five exclusive Gateways (diamonds), a Start Event (circle) to indicate where to begin the process and a End Event (circle with thick line) to indicate where the path of the process will end. So called Data Objects are used to store and convey items during process execution. Data Associations model how data is extracted from a data object into a

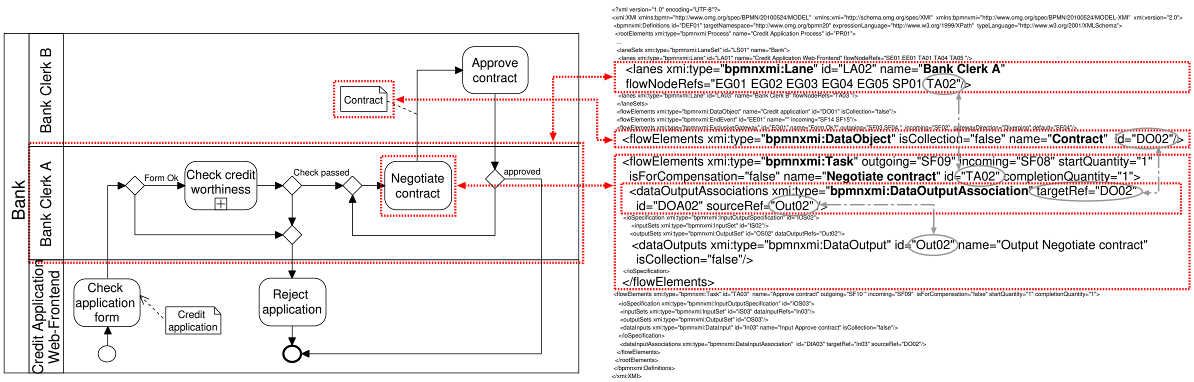

Figure 8: Example of a BPMN 2.0 collaboration model in descriptive conformance class and its XMI representation

task. In particular, this is done via the DataInput included in a task's InputSet. Similarly, a newly created data object is extracted from a task's DataOutput in the respective OutputSet (see also Figure 7). In our example the data object "Credit application" is connected to the task "Check application form" via a Data Input Association. A data object that is passed from one task to another can be attached directly to the sequence flow connecting these tasks. In Figure 8, this visualization option is shown for the data object "Contract".

The right-hand side of Figure 8 shows an excerpt of the XMI representation of the BPMN 2.0 collaboration model depicted on the left-hand side[3].

For demonstration purposes, Figure 8 highlights three areas of the BPMN model and the corresponding XMI representation. In particular, the highlighted areas include the following elements:

- Lanes partitioning the process are defined via `lanes` elements with the `xmi:type` attribute set to `bpmnxmi:Lane`. Each flow node that belongs to this lane is referenced by its id in the `flow-NodeRefs` attribute. For example, the lane named "Bank Clerk A" includes the flow nodes EG01, EG02, EG03, EG04, EG05, SP01, TA02[4].

- Data Objects are defined as `flowElements` with the `xmi:type` attribute set to `bpmnxmi:DataObject`. Figure 8 shows the XMI representation of the data object "Contract".

- Elements describing the flow in a process are also defined as `flowElements`. These elements are flow nodes and sequence flows (see also Figure 7). Figure 8 shows the XMI representation of the "Negotiate contract" task as `flowElements` element of the type `bpmnxmi:Task`.

- Data associations between data objects and tasks are defined as subelement of a task. They are either defined as `dataOutput-Associations` or `dataInputAssociations`. Figure 8 shows that the "Contract" data object is related to "Negotiate contract" as output and to "Approve contract" as input. In Figure 8 the "Contract" data object is referenced via its id "DO02" (short for "Data Object 02") in the `dataOutputAssociations` element of the "Negotiate contract" task.

[3]The transformation to XMI from BPMN is conducted via a XSLT document that is provided as part of BPMN specification (see [19]).
[4]EG=Exclusive Gateway, SP=Sub-Process, TA=Task

3. DERIVING ROLE ENGINEERING ARTIFACTS FROM PROCESS AND SCENARIO MODELS

In this section, we show how we derive role engineering artifacts from the XML Metadata Interchange (XMI) [20] representation of UML activity, UML interaction, and BPMN collaboration models.

However, note that the role engineering artifacts derived from the XMI documents are only *candidate artifacts* and are subject to a subsequent selection and/or refinement performed by human role engineers. This means that after the role engineering artifacts are fed into the role engineering tool they can be renamed or deleted (see, e.g., [23]). For example, a certain candidate artifact can be deleted if two different artifacts refer to the same role engineering entity, as it may be the case with two candidate roles "Bank Clerk A" and "Bank Clerk B" which will most likely refer to a single role called "Bank Clerk".

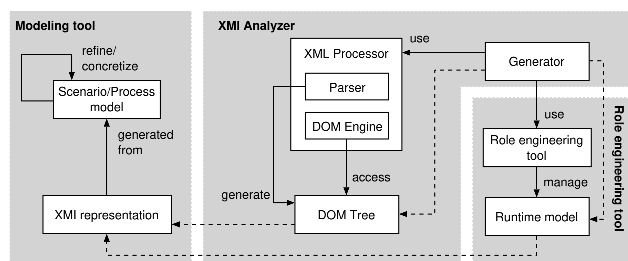

Figure 9: Deriving role engineering artifacts from scenario/process models: Structural overview

In particular, a special-purpose XMI Analyzer component transforms XMI model representations to candidate role engineering artifacts. Figure 9 depicts a structural overview of this XMI Analyzer while Figure 10 depicts the different steps of this transformation.

The XMI Analyzer uses a XML processor to parse the respective XML document and generate a corresponding DOM tree (see [2, 11, 12]). The DOM tree is an in-memory representation of the respective XML document and makes the corresponding document content accessible to software components. The Generator component then accesses the DOM tree to derive/extract role engineering artifacts and feeds them into a role engineering tool (such as the xoRET tool [23]). In turn, the role engineering tool produces a corresponding runtime model and provides an interface for human users (role engineers) to further manipulate/refine the respective role engineering artifacts and to build a tailored RBAC model (see also [23, 24]).

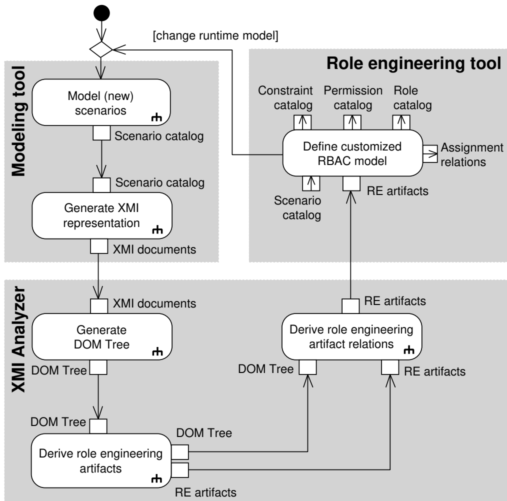

Figure 10: Deriving role engineering artifacts from scenario/process models: Task sequence

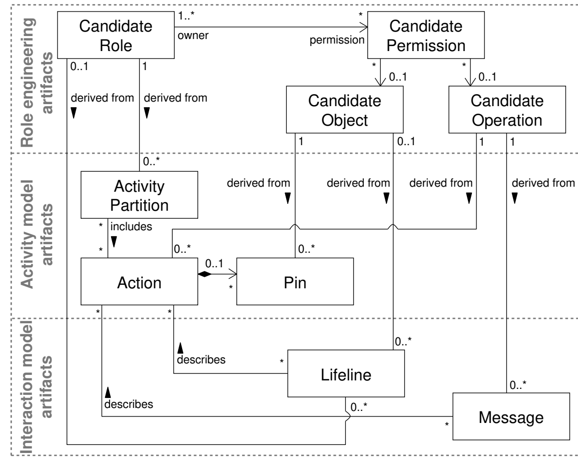

Figure 11: Derivation of role engineering artifacts from UML activity and interaction models: Integrated meta-model

Table 1 gives an overview what role engineering artifacts can be derived from the XMI representation of UML2 and BPMN 2.0 models. The details of this derivation are described in the following sections.

Table 1: Model elements to derive role engineering artifacts from UML2 and BPMN 2.0 models in XMI representation

	***xmi:Type* attribute**	**Role engineering artifacts**
UML Activity Diagram	ActivityPartition	Candidate ROLE
	OpaqueAction	Candidate PERMISSION and Candidate OPERATION
	OutputPin	Candidate OBJECT
	InputPin	Candidate OBJECT
UML Interaction Diagram	Lifeline	Candidate ROLE or Candidate OBJECT
	Message	Candidate PERMISSION and Candidate OPERATION
BPMN Descriptive Collaboration Model	Lane	Candidate ROLE
	Participant	Candidate ROLE
	Task	Candidate PERMISSION and Candidate OPERATION
	DataObject	Candidate OBJECT
	Message	Candidate OBJECT
BPMN Common Executable Collaboration Model	Resource	Candidate ROLE
	PartnerEntity	Candidate ROLE
	PartnerRole	Candidate ROLE
	Task with implementation attribute	Candidate OBJECT

3.1 Deriving Role Engineering Artifacts from UML Models

Figure 11 shows the integrated meta-model for the derivation of role engineering artifacts from UML activity and interaction models. In particular, it indicates which UML model elements are used to derive corresponding role engineering artifacts: In a nutshell, we use ActivityPartitions and Lifelines to identify candidate roles, Pins and Lifelines to identify candidate objects, as well as Actions and Messages to identify candidate operations.

3.1.1 Derivation from Activity Models

Figure 12 shows an example of how we use the XMI representation of activity models to identify role engineering artifacts. In particular, Figure 12 highlights an excerpt of Figure 4 and shows what role engineering artifacts can be derived from the corresponding XMI representation. In general, the following derivation rules are applied (see also Figures 2, 9, 10, and 11, as well as Table 1):

- We use `group` elements of the type `uml:ActivityPartition` to identify candidate roles. These candidate roles are then associated with the candidate permissions that are derived from actions included in the respective ActivityPartition. For example, the activity partition with name "Bank Clerk B" is used to derive a corresponding candidate role (see Figure 12).

- We use `node` elements of type `uml:OpaqueAction` to identify candidate operations. Moreover, the name of the respective action is also used to determine the name of the corresponding permission candidate. In Figure 12, we can thus use the `uml:OpaqueAction` with name "Approve contract" to derive the corresponding artifacts.

- We use `inputValue` elements with the `uml:InputPin` type and `outputValue` elements with the `uml:OutputPin` type to identify candidate objects. The candidate objects are then associated with the candidate operation that is identified from the corresponding action defined as `node` element of type `uml:OpaqueAction` (see also Figure 11). For instance, from Figure 12 we can derive the "Contract" candidate object for action "Approve contract" from the respective input pin.

In addition to the role engineering artifacts described above, we can also derive candidate *mutual exclusive constraints* (ME) from activity models. Mutual exclusive constraints enforce conflict of interest policies (see, e.g., [1, 3, 7, 25]). Conflict of interest arises as a result of the simultaneous assignment of two mutual exclusive tasks or roles to the same subject. In general, we use `group` elements of the type `uml:ActivityPartition` to identify candidate ME constraints. In particular, we assume that the actions included in different activity partitions must be executed by different actors. For instance, in the example from Figure 4 we can derive a candidate ME constraint on the actions "Negotiate contract" and "Approve contract". In the further course of the role engineering process, we would further refine this candidate ME constraint into a dynamic ME constraint on the corresponding permissions defined for the "Bank Clerk" role. This means, each user assigned to the

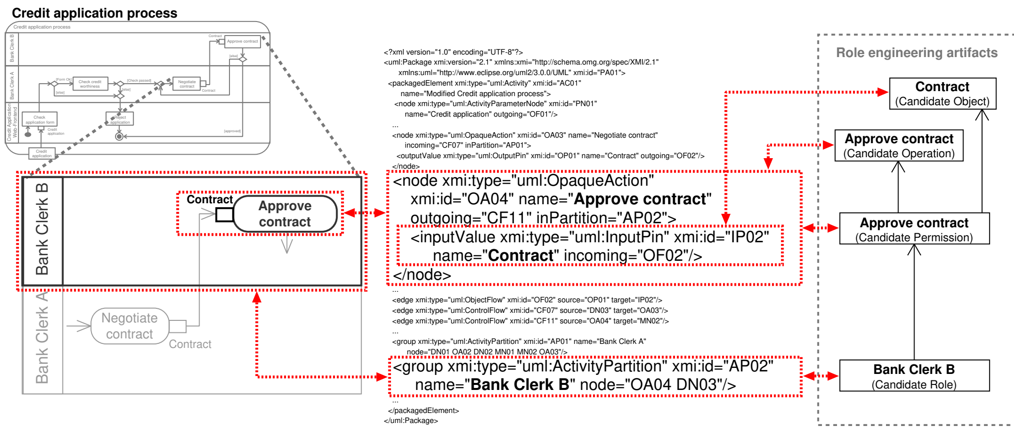

Figure 12: Example for the derivation of role engineering artifacts from activity models

"Bank Clerk" role owns both permissions and can, in principle, perform both tasks. However, due to the dynamic ME constraint on the respective permissions one always needs two different individuals acting in the "Bank Clerk" role to complete the credit application process (as it is reflected in the graphical model via two different swimlanes labeled "Bank Clerk A" and "Bank Clerk B").

3.1.2 Derivation from Interaction Models

Figure 13 shows an example how we use the XMI representation of interaction models to identify role engineering artifacts. In particular, Figure 13 highlights an excerpt of Figure 6 and shows what role engineering artifacts can be derived from the respective XMI representation. In general, the following derivation rules are applied (see also Figures 2, 9, 10, and 11, as well as Table 1):

- We use `message` elements to identify candidate operations and candidate permissions. Moreover, the `receiveEvent` and `sendEvent` attributes are used to determine the respective candidate object and the corresponding candidate role (see below).

- We use `lifeline` elements to identity candidate roles and candidate objects:

  - We derive a *candidate role* if one of the `sendEvents` covered by the respective lifeline is part of a `message` which is received by another lifeline. For instance, in Figure 13 the `sendEvent` of the `message` with name "getCustomerProfile(id)" is covered by the "Bank Clerk" lifeline, while the `receiveEvent` of this message is covered by the "CustomerMgmt" lifeline. Therefore, we derive a candidate role from the "Bank Clerk" lifeline.

  - We derive a *candidate object* if one of the `receiveEvents` covered by a lifeline is part of an `message` element which was sent by another lifeline. In the example from Figure 13, we therefore derive a candidate object from the "CustomerMgmt" lifeline, because it receives the "getCustomerProfile(id)" message from the "Bank Clerk" lifeline[5].

[5]Note that a candidate role as well as a candidate object may be derived from the very same lifeline in case the respective lifeline is both a sender and a receiver of `messages`. However, this is perfectly in sync with the typical object/component-based nature of today's software systems where different objects/components are connected and mutually invoke each others methods/procedures.

Because interaction models concretize and/or refine activity models, they are a valuable source to identify role engineering artifacts that cannot be derived from more abstract activity models.

3.2 Deriving Role Engineering Artifacts from BPMN Collaboration Models

Figure 14 shows the integrated meta-model for the derivation of role engineering artifacts from BPMN collaboration models. In particular, it indicates what BPMN model elements are used to derive corresponding role engineering artifacts: We use participants and lanes to identify candidate roles. Tasks are used to identify candidate operations, and Messages, Data Objects as well as the implementation of a task are used to identify candidate objects. All elements related to participants and lanes, such as resource, sublanes, PartnerRole and PartnerEntity are used to further refine candidate roles and to identify candidate role-hierarchies.

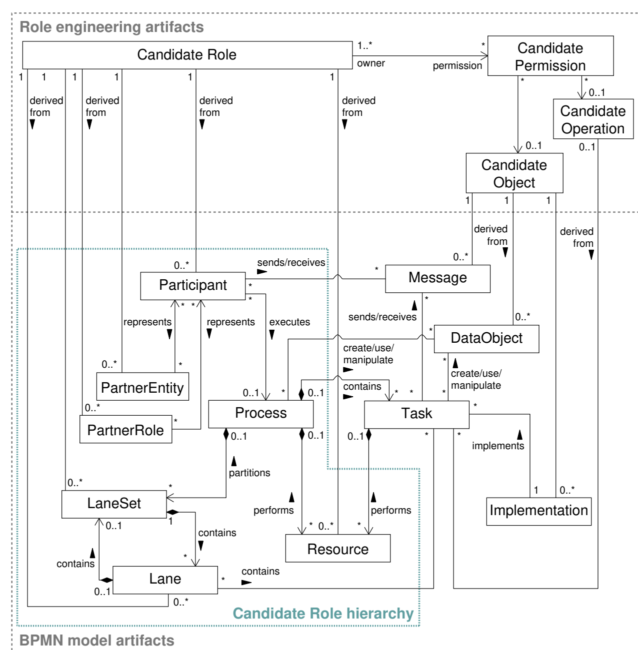

Figure 14: Derivation of role engineering artifacts from BPMN collaboration models: Integrated meta-model

Figure 13: Example for the derivation of role engineering artifacts from interaction models

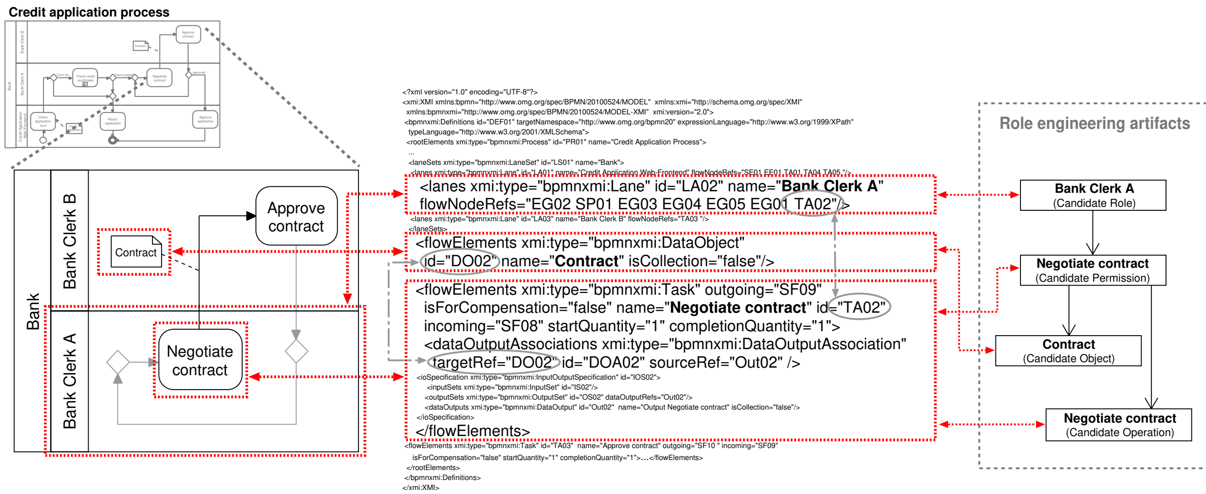

Figure 15: Example for the derivation of role engineering artifacts from collaboration models in descriptive conformance class

Figure 15 shows an example of how we identify role engineering artifacts from the XMI representation of BPMN collaboration models in the descriptive conformance class. In particular, Figure 15 highlights an excerpt of Figure 8 and shows what role engineering artifacts can be derived from the corresponding XMI representation. In general, the following derivation rules are applied (see also Figures 2, 9, 10, and 14, as well as Table 1):

- We use `lanes` elements to derive candidate roles. Tasks, including their input and output, are used to derive corresponding candidate permissions. For example, the "Bank Clerk A" lane shown in Figure 15 results in such a candidate role.
- We use `flowElements` of the type `bpmnxmi:Task` to derive candidate operations. Moreover, the name of the respective task element is also used to determine the name of the corresponding permission candidate. For example, in Figure 15 we see the "Negotiate contract" candidate permission and candidate operation.
- We use `flowElements` of the `bpmnxmi:DataObject` type to derive candidate objects. We use the data associations (either `bpmnxmi:DataOutputAssociation` or `bpmnxmi:DataInputAssociation`) of a data object to associate the candidate objects with the corresponding candidate permission. In the example, we can derive the candidate object "Contract" for the candidate permission "Negotiate contract" (see Figure 15).

Similar to UML activity models (see Section 3.1), we can also derive candidate mutual exclusion constraints (ME) from BPMN collaboration models. In BPMN models, we use the `lanes` element to identify candidate ME constraints. In particular, we assume that the tasks included in different lanes have to be executed by different actors. In the example from Figure 8 we can derive such a candidate ME constraint for the "Negotiate contract" and "Approve contract" tasks.

In general, the derivation rules presented above are valid for BPMN collaboration models of all conformance classes. Moreover, models in the common executable conformance class are more detailed and can further concretize a BPMN collaboration model. Thereby, these more detailed models can be used to further refine the automated derivation of role engineering artifacts, similar to the refinement of activity models via interaction models discussed in Section 3.1. Due to the page restrictions, we had to cut the derivation rules and examples for BPMN models in the common executable conformance class from the paper. On our webpage, we provide an extended version of this paper where we re-inserted the text we had to cut from the paper.

4. PRACTICAL RELEVANCE AND DISCUSSION

Since its first publication in June 2002, numerous consulting firms and international projects have adopted the scenario-driven role-engineering process. The most visible of which is probably the Health Level 7 (HL7) role-engineering process defined by the US National Healthcare RBAC Task Force (see [5]). Among other things, the task force applied this process to produce HL7 RBAC healthcare scenarios and a HL7 RBAC healthcare permission catalog. In addition to such international projects, we are continuously conducting role engineering projects and gained many experiences in this area (see, e.g., [15, 18, 23, 24]). For example, in 2008 we conducted a role engineering project with the Austrian

Federal Ministry of Finance[6], in 2009 we conducted a corresponding case study with the German branch of ABB[7], in 2009/10 we were involved in a rights management project with Ernst & Young[8], and currently we are conducting a role engineering project with the Vienna City Municipality[9]. In each of these (as well as in other) projects we received requests for an extended automation support of different role engineering tasks. In particular, these requests revealed the demand for an automation support of the monotonous derivation of role engineering artifacts from scenario and process models.

The automatically derived candidate artifacts serve as input for the definition of a customized RBAC model for the respective organization or information system. However, note that the candidate artifacts are subject to a subsequent selection and refinement by human role engineers and domain experts. This is because an automated derivation is well-suited to derive a first version of the respective candidate artifacts, yet it cannot produce a set of tailored, integrated, and non-redundant role engineering artifacts. For example, the subsequent refinement aims to identify redundancies resulting from the automatic derivation such as the "Bank Clerk A" and "Bank Clerk B" candidate roles which will most likely be combined into a single "Bank Clerk" role (see Section 3). Nevertheless, although the candidate artifacts require a subsequent refinement, the automated derivation facilitates a monotonic and thus error-prone task, and thereby significantly eases the tasks of human role engineers.

5. RELATED WORK

Role mining is related to role engineering and aims to derive RBAC policy sets from permissions and role definitions that exist in the software systems of an organization. In [14], Kuhlmann et al. apply data mining techniques to detect patterns in a set of access rights. Subsequently, they use these patterns to derive candidate roles combined with business (organizational and functional) information. In [10], Frank et al. present an approach for hybrid role mining. In particular, they first review preexisting business information to determine their relevance for the role mining process. Afterwards, they include the preexisting business information, such as the organizational hierarchy or job descriptions, in the role creation step of role mining. Colantonio et al. [4], present a similar approach to use business information in a role mining approach. The approach is applied to identify the roles that are to be included in a candidate role set. Molloy et al. [17] applied role mining techniques to identify roles with semantic meaning. In case subject-to-permission relations are the only information available, they apply formal concept analysis to find roles. If certain user-attribute information is also available (e.g. job positions, departments, or job responsibilities) they propose to derive roles from such user-attribute expressions. Our approach is complementary to role mining approaches and can be used in combination with role mining.

Similar to our approach, Wolter et al. derive access control policies from BPMN 1.0 models in the domain of Web Services [27]. They provide authorization constraint artifacts as extension for the BPMN meta-model. These constraints can be assigned to groups, lanes, and respective activities to define separation of duty and binding of duty constraints. To automate the extraction of security policies from process models, they propose a mapping from selected meta-model entities of BPMN and XACML (eXtensible Access Control Markup Language). XSLT is applied to automate the generation of enforceable XACML policies.

Fernandez and Hawkins [6] suggested an early approach to determine role rights from use cases. In particular, they propose to extend the textual description of use cases in order to define security requirements for use cases. Authorization rules are then derived from the specifications defined in the use case descriptions. In addition, they complement the use case descriptions with scenario diagrams to discover role rights. In [16], Mendling et al. introduced an approach to extract RBAC models from BPEL (Business Process Execution Language) processes. The approach integrates BPEL and RBAC on the meta-model level and describes how certain RBAC artifacts can be automatically derived from BPEL processes. Similar to the approach presented in this paper, the approach from [16] can be used to automate steps of the role engineering process.

Our work complements the contributions mentioned above by providing an approach for the automated derivation of role engineering artifacts from UML activity models, UML interaction models and BPMN collaboration models. In principle, it can be combined with each of the above mentioned approaches.

6. CONCLUSION

The scenario-driven role engineering process provides a systematic approach to engineer and maintain customized RBAC models. In recent years, we gained many experiences which resulted in an evolutionary enhanced role engineering process and a much better understanding of related activities and artifacts. In addition to our own projects and case studies, scenario-driven role engineering is used by several consulting firms and in international projects (see Section 4).

Such as every engineering process, the role engineering process depends significantly on human factors and cannot be completely automated. However, the automated derivation of role engineering artifacts from scenario and process models can significantly ease role engineering tasks. In particular, the automation of certain role engineering steps can help to facilitate monotonic and thereby error-prone tasks. In this paper, we presented an approach to derive role engineering artifacts from UML activity models, UML interaction models and BPMN collaboration models. However, our general approach for the derivation of role engineering artifacts is based on meta-model integration and is therefore independent of the UML, BPMN, or any other modeling language (see Section 1.2).

Human role engineers as well as domain experts from the respective organization can adapt and refine the derived candidate role engineering artifacts in order to specify a tailored RBAC model. Furthermore, to ease the work of role engineers and to reduce ambiguities in the derived role engineering artifacts, we recommend the following simple modeling guidelines for UML and BPMN: a) the name of an UML interaction model should be identical to the name of the action it refines; b) the names of subjects and objects should be consistent across the models (i.e. in UML the same subject or object is always referenced via the same identifier string or id, such as the "CustomerMgmt" sub-system or the "Bank Clerk" actor from our example) c) a clear understanding of lanes in a model should be defined, since BPMN leaves the meaning of lanes up to the modelers and d) the usage of PartnerRoles, PartnerEntities and Resources across all BPMN models for participants, lanes and tasks should be accurately defined.

In our future work, we will further investigate how we can derive different types of candidate constraints (such as context constraints, see [26]) from UML and BPMN models. In addition, we

[6]http://english.bmf.gv.at/
[7]http://www.abb.com/
[8]http://www.ey.com/
[9]http://www.wien.gv.at/ma14/

are currently investigating the options to combine the derivations from scenario and process models based on different languages to automatically propose a candidate RBAC model. Moreover, we plan to investigate further options to integrate role engineering and related role mining and process mining approaches.

7. REFERENCES

[1] G. J. Ahn and R. Sandhu. Role-Based Authorization Constraints Specification. *ACM Transactions on Information and System Security (TISSEC)*, 3(4), November 2000.

[2] V. Apparao, S. Byrne, M. Champion, and et. al. Document Object Model (DOM) Level 1 Specification. available at: http://www.w3.org/TR/1998/REC-DOM-Level-1-19981001/, October 1998. W3 Consortium Recommendation.

[3] D. Clark and D. Wilson. A Comparison of Commercial and Military Computer Security Policies. In *Proc. of the IEEE Symposium on Security and Privacy*, April 1987.

[4] A. Colantonio, R. Di Pietro, A. Ocello, and N. V. Verde. A Formal Framework to Elicit Roles with Business Meaning in RBAC Systems. In *Proc. of the 14th ACM Symposium on Access Control Models and Technologies (SACMAT)*, June 2009.

[5] E. Coyne and J. Davis. *Role Engineering for Enterprise Security Management*. Artech House, 2008.

[6] E. B. Fernandez and J. C. Hawkins. Determining Role Rights from Use Cases. In *Proc. of the 2nd ACM Workshop on Role-Based Access Control (RBAC)*, New York, NY, USA, 1997.

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

[8] D. Ferraiolo and D. Kuhn. Role-Based Access Controls. In *Proc. of the 15th National Computer Security Conference (CSC)*, October 1992.

[9] D. Ferraiolo, D. Kuhn, and R. Chandramouli. *Role-Based Access Control, Second Edition*. Artech House, 2007.

[10] M. Frank, A. P. Streich, D. A. Basin, and J. M. Buhmann. A Probabilistic Approach to Hybrid Role Mining. In *Proc. of the 16th ACM Conference on Computer and Communications Security (CCS)*, 2009.

[11] A. L. Hors, P. L. Hegaret, L. Wood, and et. al. Document Object Model (DOM) Level 2 Core Specification. available at: http://www.w3.org/TR/2000/REC-DOM-Level-2-Core-20001113/, November 2000. W3 Consortium Recommendation.

[12] A. L. Hors, P. L. Hegaret, L. Wood, G. Nicol, J. Robie, M. Champion, and S. Byrne. Document Object Model (DOM) Level 3 Core Specification, Version 1.0. available at: http://www.w3.org/TR/DOM-Level-3-Core, April 2004. W3 Consortium Recommendation.

[13] M. Jarke, X. Bui, and J. Carroll. Scenario Management: An Interdisciplinary Approach. *Requirements Engineering Journal*, 3(3/4), 1998.

[14] M. Kuhlmann, D. Shohat, and G. Schimpf. Role Mining - Revealing Business Roles for Security Administration using Data Mining Technology. In *Proc. of the 7th ACM Symposium on Access Control Models and Technologies (SACMAT)*, New York, NY, USA, 2003.

[15] S. Kunz, S. Evdokimov, B. Fabian, B. Stieger, and M. Strembeck. Role-Based Access Control for Information Federations in the Industrial Service Sector. In *Proc. of the 18th European Conference on Information Systems (ECIS)*, June 2010.

[16] J. Mendling, M. Strembeck, G. Stermsek, and G. Neumann. An Approach to Extract RBAC Models from BPEL4WS Processes. In *Proc. of the 13th IEEE International Workshops on Enabling Technologies: Infrastructures for Collaborative Enterprises (WETICE)*, June 2004.

[17] I. Molloy, H. Chen, T. Li, Q. Wang, N. Li, E. Bertino, S. Calo, and J. Lobo. Mining roles with semantic meanings. In *Proc. of the 14th ACM Symposium on Access Control Models and Technologies (SACMAT)*, pages 21–30, New York, NY, USA, 2008. ACM.

[18] G. Neumann and M. Strembeck. A Scenario-driven Role Engineering Process for Functional RBAC Roles. In *Proc. of 7th ACM Symposium on Access Control Models and Technologies (SACMAT)*, June 2002.

[19] OMG. Business Process Modeling Notation (BPMN). available at: http://www.omg.org/spec/BPMN/2.0/Beta2/, May 2010. Version 2.0 - Beta 2, dtc/2010-06-04, The Object Management Group.

[20] MOF 2.0 / XMI Mapping Specification. available at: http://www.omg.org/technology/documents/formal/xmi.htm, December 2007. Version 2.1.1, formal/2007-12-01, The Object Management Group.

[21] OMG Unified Modeling Language (OMG UML): Superstructure. available at: http://www.omg.org/technology/documents/formal/uml.htm, February 2009. Version 2.2, formal/2009-02-02, The Object Management Group.

[22] R. Sandhu, E. Coyne, H. Feinstein, and C. Youman. Role-Based Access Control Models. *IEEE Computer*, 29(2), February 1996.

[23] M. Strembeck. A Role Engineering Tool for Role-Based Access Control. In *Proc. of the 3rd Symposium on Requirements Engineering for Information Security (SREIS)*, August 2005.

[24] M. Strembeck. Scenario-Driven Role Engineering. *IEEE Security & Privacy*, 8(1), January/February 2010.

[25] M. Strembeck and J. Mendling. Generic Algorithms for Consistency Checking of Mutual-Exclusion and Binding Constraints in a Business Process Context. In *Proc. of the 18th International Conference on Cooperative Information Systems (CoopIS), Lecture Notes in Computer Science (LNCS), Vol. 6426, Springer Verlag*, October 2010.

[26] M. Strembeck and G. Neumann. An Integrated Approach to Engineer and Enforce Context Constraints in RBAC Environments. *ACM Transactions on Information and System Security (TISSEC)*, 7(3), August 2004.

[27] C. Wolter, A. Schaad, and C. Meinel. Deriving xacml policies from business process models. In M. Weske, M. Hacid, and C. Godart, editors, *Web Information Systems Engineering, WISE 2007 Workshops*, volume 4832 of *Lecture Notes in Computer Science*, pages 142–153. Springer Berlin / Heidelberg, 2007.