A UML Extension for Modeling Break-Glass Policies

Sigrid Schefer-Wenzl, Mark Strembeck

Institute for Information Systems, New Media Lab
Vienna University of Economics and Business (WU Vienna)
Augasse 2-6, 1090 Vienna, Austria
{firstname.lastname}@wu.ac.at

Abstract: In emergency situations, certain subjects sometimes have to perform important tasks although they are usually not authorized to perform these tasks. Break-glass policies have been introduced as a sophisticated exception handling mechanism to resolve such situations. They enable selected subjects to break or override the standard access control policies of an information system in a controlled manner. However, modeling support for break-glass policies is largely missing. In this paper, we present an approach to provide modeling support for break-glass policies in the context of process-related RBAC models. In particular, we provide a UML2 extension that allows for the integrated modeling of processes and break-glass policies. Additional constraints are formally specified via OCL. We also implemented our approach as an extension to the BusinessActivity library and runtime engine. The source code of our implementation is available for download.

1 Introduction

Process modeling languages provide primitives and construction rules to model the sequence of tasks performed in a business process. Corresponding access control models specify which subjects are authorized to perform the tasks that are included in the business processes (see, e.g., [Str10, SM11, WBK03]). However, most process diagrams and corresponding access control models only visualize standard task sequences and do not consider exceptional situations, such as emergency scenarios, where no authorized subject is available to execute a particular task (see, e.g., [RvdAH06, WRR07]). This results from the fact that proper modeling support for emergency scenarios is largely missing [vdARD07].

In many organizational environments some critical tasks exist which – in exceptional cases – must be performed by a subject although he/she is usually not authorized to perform these tasks. For example, a junior physician shall be able to perform certain tasks of a senior physician in case of emergency. *Break-glass policies* can be used to flexibly handle this kind of exceptional situations by breaking or overriding the standard access controls in a controlled manner (see, e.g., [BP09, BPW10, FCCA$^+$06, MCMD11]). The term "break-glass" is a metaphor referring to the act of breaking the glass to pull a fire alarm. Accordingly, process-related break glass policies define override rules for subjects to allow the execution of certain tasks in exceptional cases. Applying a break-glass policy implies that the resulting task executions need to be carefully recorded for later audit and review. Typically, a special review process is triggered to monitor such break-glass executions.

In order to model process-related break-glass policies, we need an approach that integrates the break-glass concept into a modeling language. However, standard process modeling languages, such as BPMN [OMG11a] or UML Activity diagrams [OMG11b], do not provide native language constructs to model break-glass policies. In current practice, the lack of native modeling support for exception handling mechanisms can result in very complex diagrams depicting all possible exceptional execution paths and scenarios (see, e.g., [vdARD07]). In this paper, we therefore present a break-glass extension for UML that can help to specify graphical break-glass models. We define a domain-specific extension for the Unified Modeling Language (UML) [OMG11b] for modeling process-related break-glass models. In particular, we integrate our break-glass models into the BusinessActivity extension that supports the definition of process-related RBAC models [SM11]. Moreover, we also implemented the corresponding break-glass concepts as an extension to the BusinessActivity library and runtime engine (see [SM10, SM11]). The source code of our implementation is available for download[1].

The remainder of this paper is structured as follows. In Section 2, we give an overview of the *BusinessActivities* extension and motivate the need for integrating break-glass policies into business process models. Section 3 introduces our extension for modeling break-glass models via extended UML2 Activity diagrams. Moreover, we formally define the semantics of our newly introduced modeling elements via OCL constraints. Next, Section 4 presents an example business process model including break-glass policies. Section 5 discusses our approach in comparison to related work and Section 6 concludes the paper.

2 Background and Motivation

2.1 BusinessActivities: Modeling Support for Process-Related RBAC Models

In recent years, role-based access control (RBAC) [FKC07, SCFY96] has developed into a de facto standard for access control in both, research and industry. In RBAC, roles correspond to different job-positions and scopes of duty within a particular organization or information system [Str10]. Access permissions are assigned to roles according to the tasks this role has to accomplish, and subjects (e.g., human users) are assigned to roles. Thereby, each subject acquires all permissions that are necessary to fulfill its duties. Several extensions for RBAC exist for different application domains.

The *BusinessActivities* extension was designed for the integrated modeling of business processes and access control concepts by providing modeling support for process-related RBAC models [SM11]. Figure 1 shows an example of a simple medical examination process modeled as a BusinessActivity. The process starts when a patient arrives at the hospital. Subsequently, the “Medical examination” task (t_1) is conducted to reach a med-

[1] `http://wi.wu.ac.at/home/mark/BusinessActivities/library.html`

ical diagnosis. Next, the "Determine treatment options" task (t_2) is executed to devise an appropriate treatment plan. This treatment plan has to be confirmed by a second physician (t_3). In case the treatment plan includes errors or is incomplete, it must be revised before it is resubmitted for confirmation. Finally, the "Medical treatment" task (t_4) is performed. Each of the tasks (e.g., medical examination) is typically associated with certain access permissions (e.g., to read the patient record). Therefore, *subjects* participating in this workflow must be authorized to perform the tasks needed to complete the process (see, e.g., [GMPT01, OP03]).

In Figure 1, members of the Junior Physician role are permitted to perform the tasks t_1, t_2, and t_4. Task t_3 ("Confirm treatment") can only be performed by subjects assigned to the Senior Physician role. Furthermore, the Junior Physician role is defined as junior-role of the Senior Physician role via a role-to-role assignment ("rrAssign"). In RBAC, a senior-role inherits all permissions of its junior-roles.

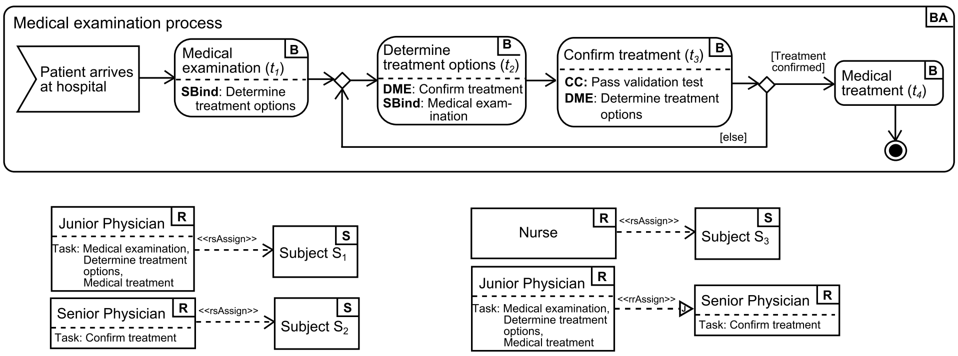

Figure 1: Simple medical examination process

In addition, the BusinessActivities extension supports the definition of different types of entailment constraints. A *task-based entailment constraint* places some restriction on the subjects who can perform a $task_x$ given that a certain subject has performed $task_y$. Thus, task-based entailment constraints have an impact on the combination of subjects and roles who are allowed (or required) to execute particular tasks (see, e.g., [RHE05, SSMB11, SM10, SM11, TCG04, WA06, WSM08]). Examples of entailment constraints include static mutual exclusion (SME), dynamic mutual exclusion (DME), subject-binding (SB), and role-binding (RB) constraints. A SME constraint defines that two statically mutual exclusive tasks must never be assigned to the same subject. In turn, DME tasks can be assigned to the same role, but within the same process instance they must be executed by different subjects. A SB constraint defines that two bound tasks must be performed by the same individual within the same process instance. A RB constraint defines that bound tasks must be performed by members of the same role, but not necessarily by the same individual.

In the example process shown in Figure 1, we define a subject-binding between the tasks t_1 and t_2 to ensure that the same physician who performed the examination in the "Medical examination" task also evaluates appropriate medical treatment options. This subject-binding is indicated via *SBind* entries in the corresponding task symbols (see Figure 1). In addition, we define a DME constraint on the tasks t_2 and t_3 to enforce the four-eyes-principle on medical examinations. Thus, for each medical examination the "Determine treatment options" and the "Confirm treatment" tasks must always be conducted by two different individuals. This is an essential quality and safety measure in hospitals to guard against mistakes and malpractice.

Moreover, in an IT-supported workflow, *context constraints* can be defined as a means to consider context information in access control decisions (see, e.g. [BBF01, SN04]). Typical examples for context constraints in organizational settings regard the temporal or spatial context of task execution, user-specific attributes, or the task execution history of a user (see, e.g., [CCB08]). In this paper, context constraints define that certain contextual attributes must meet certain predefined conditions to permit the execution of a specific task [SN04, SWS12]. In the example process, a context constraint (CC) can be defined on the "Confirm treatment" task which specifies several conditions that must be met in order to successfully validate the medical treatment plan (see Figure 1).

2.2 Motivation for Modeling Break-Glass Policies

Let us consider three potential *emergency scenarios* for the medical examination process shown in Figure 1:

(1) In case of emergency, no senior physician is available. However, only senior physicians are allowed to perform the "Confirm treatment" task (see Figure 1). To be able to start the "Medical treatment", a break-glass policy can authorize junior physicians to perform the "Confirm treatment" in case of emergency.

(2) Only one authorized subject is available to perform the tasks "Determine treatment options" and "Confirm treatment". However, due to a DME constraint these two tasks must be executed by different subjects. A break-glass policy can authorize the only available subject to override the DME constraint in case of emergency.

(3) If no physician is available to perform the "Medical treatment" task in case of emergency, subject s_3 – who is assigned to the nurse role – is allowed to execute this task. However, all other members of the nurse role do not have the necessary skills to perform this task. Therefore, a break-glass policy directly authorizes subject s_3 to perform the "Medical treatment" in a break-glass scenario.

Standard UML elements do not provide modeling support for process-related break-glass policies. For example, in Figure 1, these emergency scenarios would have to be included into the same process model. Apparently, this would result in a very complex diagram (see, e.g., [vdARD07]). Alternatively, each of the scenarios either needs to be modeled in a separate process model, or the break-glass information is included as a comment into the UML activity diagram. This simple example already shows that it is difficult to describe all connections and implications of process-related break-glass policies in a textual manner. For this reason, we define a UML extension for the integrated modeling of process-related break-glass policies.

3 A UML Extension for Process-Related Break-Glass Policies

The Unified Modeling Language (UML) [OMG11b] is a de facto standard for the specification of information systems. Modeling support for break-glass policies via a standard notation can help to bridge the communication gap between software engineers, security experts, experts of the application domain, and other stakeholders (see, e.g., [MJ10]). Our domain-specific modeling extension for break-glass policies serves as an enabler to document and communicate how certain emergency scenarios can be handled in a business process.

UML2 Activity models offer a process modeling language that allows to model the control and object flows between different actions. The main element of an Activity diagram is an Activity. Its behavior is defined by a decomposition into different Actions. An UML2 Activity thus models a process while the Actions included in the Activity are used to model tasks (for details on UML2 Activity models, see [OMG11b]). However, sometimes diagrams can not provide all relevant aspects of a specification. Therefore, there is a need to define additional constraints about the modeling elements. The Object Constraint Language (OCL) provides a formal language that enables the definition of constraints on UML models [OMG12]. We apply the OCL to define additional break-glass specific constraints for our UML extension. In particular, the OCL invariants defined in Section 3.2 ensure the consistency and correctness of UML models using our new modeling elements.

The UML standard basically provides two options to adapt its metamodel to a specific area of application [OMG11b]: a) defining a UML profile specification using stereotypes, tag definitions, and constraints. A UML profile must not change the UML metamodel but can only extend existing UML meta-classes for special domains. Thus, UML profiles are not a first-class extension mechanism (see [OMG11b, page 660]); b) extending the UML metamodel, which allows for the definition of new elements with customized semantics. In this paper, we apply the second option because the newly defined modeling elements for break-glass policies require new semantics which are not available in the UML metamodel. Thus, we introduce the *BreakGlassBusinessActivities* extension for the UML metamodel which is designed for modeling process-related break-glass policies (see Section 3.1). In particular, we extend the *BusinessActivities* package [SM11], which provides UML modeling support for process-related RBAC models. We also implemented the extended metamodel presented in Section 3.1 as well as the corresponding constraints provided in Section 3.2 as a break-glass extension to the BusinessActivity library and runtime engine[1] (see [SM10, SM11]).

3.1 Metamodel overview

A *BusinessActivity* [SM11] is a specialized UML Activity (see Figure 2). A *BusinessAction* corresponds to a task and comprises all permissions to perform the task. *Roles* and *Subjects* are linked to BusinessActions. For a detailed discussion on how mutual exclu-

[1] Available at `http://wi.wu.ac.at/home/mark/BusinessActivities/library.html`

sion, binding, and context constraints are integrated into the BusinessActivities extension, see [SM11, SWS12].

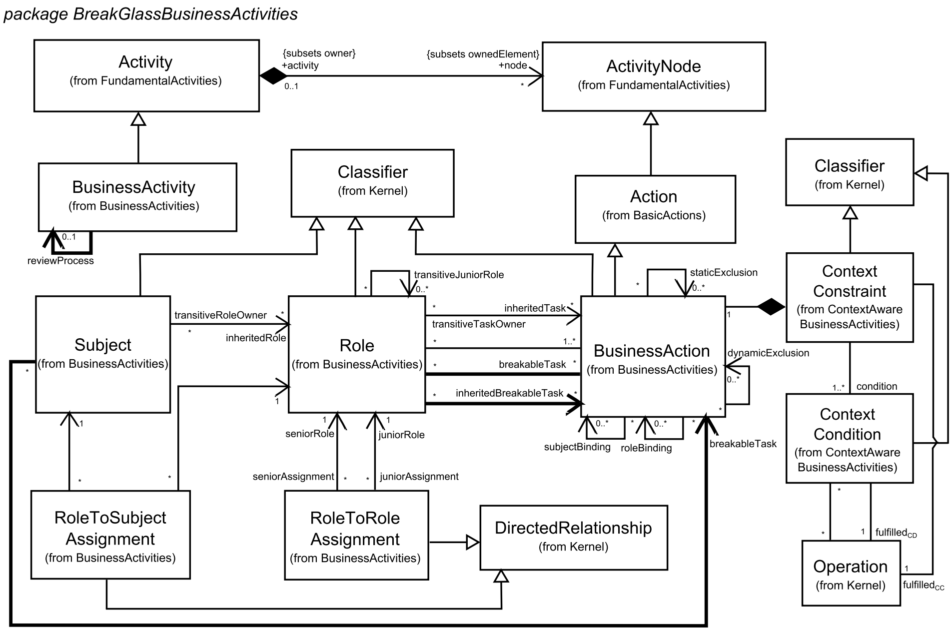

Figure 2: UML metamodel extension for process-related break-glass RBAC models

To support the definition of break-glass policies in business process models, we specify that certain *breakable tasks* can be performed by subjects who are usually not allowed to execute these tasks. For this purpose, override rules regulate that members of a certain role are permitted to perform a certain task in case of emergency (*breakable-by-role* override). In addition to role-based break-glass rules, our approach enables the definition of subject-specific break-glass rules, i.e. only a certain subject is authorized to execute a task in case of emergency (*breakable-by-subject* override). Breakable-by-subject override rules are used in cases where only certain individuals have all necessary competencies to perform the breakable task. Each break-glass execution will be recorded and subsequently be monitored via a corresponding review process.

For integrating break-glass policies into the UML metamodel, we introduce the following new relations: Each Role can include *breakableTasks* and *inheritedBreakableTasks*, which are inherited from its junior-roles. These two relationships can be used to visualize that members of a certain role are authorized to perform the assigned tasks only in case of emergency. Similarly, each Subject can be related to *breakableTasks* to show that a particular subject is permitted to perform these tasks in case of emergency. Figure 3 illustrates presentation options to visualize the breakable-by-role and breakable-by-subject override relations via “Breakable” entries. Note that these relations are formally defined through our UML metamodel extension and therefore exist independent of their actual graphical representation. Moreover, each BusinessActivity is related to a *reviewProcess* (see below).

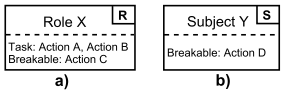

Figure 3: Visualizing (a) breakable-by-role and (b) breakable-by-subject override relations

Each instance of a BusinessAction and the corresponding BusinessActivity instance are marked as *broken* if the BusinessAction has been executed by a subject via a break-glass override assignment (see OCL constraints 1 and 2 in Section 3.2). For each broken BusinessActivity, there has to exist a corresponding reviewProcess (see Figure 2 and OCL constraint 3). A particular reviewProcess can be assigned to an arbitrary number (one or more) of BusinessActivity processes in an organization. Roles and subjects can own a task either regularly or via a break-glass override assignment (see OCL constraints 4 and 5). Moreover, in a break-glass scenario, the corresponding entailment constraints do not have to be fulfilled (see Constraints 1 and 2 as well as OCL constraints 7, 8, 9, 10).

3.2 OCL constraints

A structural UML model cannot capture certain types of domain-specific constraints which are relevant for describing a target domain. Thus, additional constraints can be defined, for example, by using a constraint expression language, such as the OCL [OMG12]. Below, we use OCL invariants to define the semantics by encoding break-glass specific constraints.

OCL Constraint 1 *Each BusinessAction defines an attribute called "broken" stating if a certain BusinessAction instance is executed via a break-glass override assignment:*

```
context BusinessAction inv:
self.instanceSpecification->forAll(i |
   i.slot->exists(b |
      b.definingFeature.name = broken ))
```

OCL Constraint 2 *Each BusinessActivity defines an attribute called "broken" stating if a certain BusinessActivity instance includes at least one broken BusinessAction:*

```
context BusinessActivity inv:
self.instanceSpecification->forAll(i |
   i.slot->exists(b |
      b.definingFeature.name = broken ))
```

OCL Constraint 3 *For each broken BusinessActivity instance, there has to exist a corresponding reviewProcess:*

```
context BusinessActivity inv:
self.instanceSpecification->forAll(i |
   if i.slot->exists(b |
      b.definingFeature.name = broken and b.value = true)
   then self.reviewProcess->notEmpty()
   else true endif)
```

OCL Constraint 4 *Each role is allowed to own a task either regularly or via a break-glass override assignment. To separate regular task ownerships from break-glass task ownerships, we need to ensure that no BusinessAction is assigned to a certain role via both mappings:*

```
context Role
inv: self.businessAction->forAll(b |
       self.breakableTask->select(bbr |
         bbr.name = b.name)->isEmpty())
inv: self.businessAction->forAll(b |
       self.inheritedBreakableTask->select(bbri |
         bbri.name = b.name)->isEmpty())
inv: self.inheritedTask->forAll(bi |
       self.breakableTask->select(bbr |
         bbr.name = bi.name)->isEmpty())
inv: self.inheritedTask->forAll(bi |
       self.inheritedBreakableTask->select(bbri |
         bbri.name = bi.name)->isEmpty())
```

OCL Constraint 5 *Each subject is allowed to own a task either regularly (via its role memberships) or via a break-glass override assignment. To separate regular task ownerships from breakable task ownerships, we need to ensure that no BusinessAction is assigned to a certain subject via both mappings:*

```
context Subject
inv: self.roleToSubjectAssignment->forAll(rsa |
       rsa.role.businessAction->forAll(b |
         self.breakableTask->select(bbs |
           bbs.name = b.name)->isEmpty()))
inv: self.roleToSubjectAssignment->forAll(rsa |
       rsa.role.inheritedTask->forAll(bi |
         self.breakableTask->select(bbs |
         bbs.name = bi.name)->isEmpty()))
inv: self.inheritedRole->forAll(ri |
       ri.businessAction->forAll(b |
         self.breakableTask->select(bbs |
           bbs.name = b.name)->isEmpty()))
inv: self.inheritedRole->forAll(ri |
       ri.inheritedTask->forAll(bi |
         self.breakableTask->select(bbs |
           bbs.name = bi.name)->isEmpty()))
```

OCL Constraint 6 *Each role inherits the breakable tasks assigned to its junior-roles (i.e. breakable tasks assigned to junior-roles are indirectly/transitively assigned to the corresponding senior-roles):*

```
context Role
inv: self.seniorAssignment->forAll(sa |
       sa.juniorRole.breakableTask->forAll(bbr |
         self. inheritedBreakableTask->exists(ibbr | ibbr.name = bbr.name)) and
       sa.juniorRole.inheritedBreakableTask->forAll(jb |
         self.inheritedBreakableTask.exists->(ibbr | ibbr.name = jb.name)))
inv: self.inheritedBreakableTask->forAll(ibbr |
       self.seniorAssignment->exists(sa |
         sa.juniorRole.breakableTask->exists(bbr | bbr.name = ibbr.name) or
         sa.juniorRole.inheritedBreakableTask->exists(jb | jb.name = ibbr.name)))
```

OCL Constraint 7 *For all broken BusinessAction instances, the executing subjects of corresponding SME tasks do not have to be different:*

```
context BusinessAction inv:
self.instanceSpecification->forAll(b |
  b.slot->select(s |
    s.definingFeature.name=broken
    if (s.value = true) then
      self.staticExclusion->forAll(sme |
        sme.instanceSpecification->forAll(i |
          b.slot->forAll(bs |
            i.slot->forAll(is |
              if bs.definingFeature.name=executingSubject
                and is.definingFeature.name=executingSubject
                then (bs.value = is.value) or not (bs.value = is.value)
              else true endif ))))
     else true endif ))
```

OCL Constraint 8 *For all broken BusinessAction instances, the executing subjects of DME tasks do not have to be different:*

```
context BusinessAction inv:
self.instanceSpecification->forAll(b |
  b.slot->select(s |
    s.definingFeature.name=broken
    if (s.value = true) then
      self.dynamicExclusion->forAll(dme |
        dme.instanceSpecification->forAll(i |
          b.slot->forAll(bs |
            i.slot->forAll(is |
              if bs.definingFeature.name=executingSubject
                and is.definingFeature.name=executingSubject
                then (bs.value = is.value) or not (bs.value = is.value)
              else true endif ))))
     else true endif ))
```

OCL Constraint 9 *For all broken BusinessAction instances, the executing role of role-bound tasks does not have to be the same:*

```
context BusinessAction inv:
self.instanceSpecification->forAll(b |
  b.slot->select(s |
    s.definingFeature.name=broken
    if (s.value = true) then
      self.roleBinding->forAll(rbt |
        rbt.instanceSpecification->forAll(i |
          b.slot->forAll(bs |
            i.slot->forAll(is |
              if bs.definingFeature.name=executingSubject
                and is.definingFeature.name=executingSubject
                then (bs.value = is.value) or not (bs.value = is.value)
              else true endif ))))
     else true endif ))
```

OCL Constraint 10 *For all broken BusinessAction instances, the executing subject of subject-bound tasks does not have to be the same:*

```
context BusinessAction inv:
self.instanceSpecification->forAll(b |
  b.slot->select(s |
    s.definingFeature.name=broken
    if (s.value = true) then
      self.subjectBinding->forAll(sbt |
        sbt.instanceSpecification->forAll(i |
          b.slot->forAll(bs |
            i.slot->forAll(is |
              if bs.definingFeature.name=executingSubject
                and is.definingFeature.name=executingSubject
                then (bs.value = is.value) or not (bs.value = is.value)
              else true endif ))))
     else true endif ))
```

Moreover, the following two constraints must be satisfied which cannot be expressed in OCL (see [OMG11b]):

Constraint 1 *For all broken BusinessAction instances, context constraints do not have to be fulfilled. Therefore, the fulfilled$_{CC}$ Operations do not have to evaluate to true.*

Constraint 2 *For all broken BusinessAction instances, context conditions do not have to be fulfilled. Therefore, the fulfilled$_{CD}$ Operations do not have to evaluate to true.*

4 Perspectives for UML Break-Glass Models

We suggest to use three complementary perspectives to model process-related break-glass RBAC models. This is because capturing all aspects within the process model will presumably overload it. Figure 4a shows the *process perspective* of the medical examination process (see Section 2).

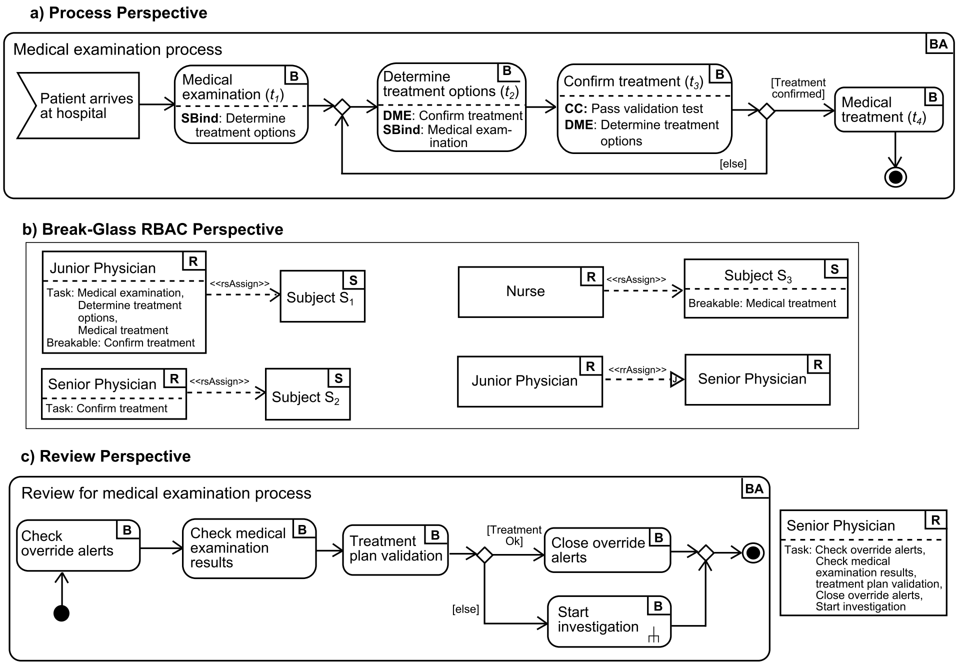

Figure 4: Example for process-related break-glass RBAC models

The *Break-Glass RBAC perspective* is exemplified in Figure 4b illustrating task-to-role, role-to-subject, and role-to-role assignments. For example, subject s_1 is assigned to the Junior Physician role. Corresponding notation symbols are described in detail in [SM11]. Moreover, this perspective provides a detailed view on which role or subject is allowed to perform a break-glass override. For example, we define a breakable-by-role override relation between the Junior Physician role and the "Confirm treatment" task in Figure 4b. Thus, in a break-glass scenario, members of the junior physician role are able to perform the "Confirm treatment" task. Moreover, a breakable-by-subject override is defined on subject s_3 and the "Medical treatment" task, because nurse s_3 has all necessary skills to perform the medical treatment in an emergency case.

Finally, the *review perspective* illustrates the review process which is triggered each time after a break-glass override is executed (see Section 3). An example review process for the medical examination process is shown in Figure 4c. In particular, a physician (who was not involved in the medical examination process) is appointed to perform the following tasks: After checking the override alerts for a particular process, the physician checks the medical examination results and validates the medical treatment plan. If the treatment plan is successfully validated, the override alerts are closed. Otherwise, an investigation process is started.

5 Discussion and Related Work

Standard process modeling languages, such as BPMN [OMG11a] or UML Activity diagrams [OMG11b], do not provide native language constructs to model break-glass policies. Previous work by van der Aalst et al. [vdARD07] has identified a general lack of process modeling language capabilities to adequately model emergency scenarios. Due to missing modeling support, organizations need to specify break-glass policies via informal textual descriptions, or the modeling of various break-glass scenarios results in complex and confusing diagrams. Such work-arounds, however, easily result in consistency, maintainability, and communication problems. The separation between the regular process model perspective and the break-glass view has been suggested in [vdARD07] in order to clearly distinguish between both scenarios. Our approach provides integrated modeling support for break-glass policies in a process-related RBAC context. Moreover, we propose three different modeling perspectives, where each of these perspectives focuses on different aspects of integrated break-glass models. By defining an extension to the UML2 standard and specifying OCL constraints for our newly introduced modeling elements, our extension can also be integrated with UML-based software tools.

In recent years, there has been much work on various aspects of (process-related) break-glass policies (see, e.g., [Pov00, FCCA$^+$06, AdVF$^+$10, MCMD11, FCF$^+$09]). In [BP09], a break-glass extension for SecureUML is provided. The resulting SecureUML break-glass policies can then be transformed into XACML. However, this approach does not consider break-glass decisions in connection with dynamic mutual exclusion or binding constraints. In [WBK03], Wainer et al. present an RBAC model for workflow systems that allows the controlled overriding of entailment constraints in case of emergency. To achieve this, each constraint is associated with a certain level of priority. On the other hand, roles hold override privileges according to their level of responsibility. A comprehensive overview of exception handling patterns – including resource reallocation – is provided in [RvdAH06].

To the best of our knowledge, this work represents the first attempt to address break-glass policies from a process modeling perspective. However, several other approaches exist that deal with process adaptations and process evolutions in order to flexibly handle different types of exceptions in process-aware information systems. For example, [RD98] provides a formal model to support dynamic structural changes of process instances. A set of change operations is defined that can be applied by users in order to modify a process instance execution path. In [WRR07], change patterns and change support features are identified and several process management systems are evaluated regarding their ability to support process changes. Exception handling via structural adaptations of process models are also considered in [RRMD09]. In particular, several correctness criteria and their application to specific process meta models are discussed. In comparison to our work, all of these approaches have in common that processes must be changed in order to handle exceptional situations. The main goal of our approach is to maintain the designed process flow, while ensuring that only authorized subjects are allowed to participate in a workflow.

6 Conclusion

The need for integrated modeling of business processes and certain kinds of exception handling mechanisms has been repeatedly identified in research and practice. In this paper, we focus on the integrated modeling of business processes and break-glass policies to define who is allowed to execute particular critical tasks in case of emergency. This is necessary because standard process modeling languages do not provide native language support for break-glass policies. However, providing suitable modeling primitives for break-glass policies is especially important to support the controlled overriding of access rights in information systems.

In particular, we presented a domain-specific modeling extension for the UML2. We also used the Object Constraint Language to formally define break-glass specific knowledge which cannot be captured via the metamodel extension. Thus, our approach can be applied to supplement other UML-based approaches and can be integrated into UML-based software tools. Moreover, we implemented our approach as a break-glass extension for the BusinessActivity library and runtime engine.

References

[AdVF+10] C. A. Ardagna, S. D. C. di Vimercati, S. Foresti, T. W. Grandison, S. Jajodia, and P. Samarati. Access control for smarter healthcare using policy spaces. *Computers & Security*, 29(8), 2010.

[BBF01] Elisa Bertino, Piero Andrea Bonatti, and Elena Ferrari. TRBAC: A temporal role-based access control model. *ACM Transactions on Information and System Security (TISSEC)*, 4(3), 2001.

[BP09] Achim D. Brucker and Helmut Petritsch. Extending Access Control Models with Break-Glass. In *Proceedings of the 14th ACM symposium on Access control models and technologies (SACMAT)*, 2009.

[BPW10] Achim D. Brucker, Helmut Petritsch, and Stefan G. Weber. Attribute-Based Encryption with Break-glass. In *Proc. of the Workshop In Information Security Theory And Practice (WISTP)*, 2010.

[CCB08] Frederic Cuppens and Nora Cuppens-Boulahia. Modeling contextual security policies. *International Journal of Information Security*, 7(4), July 2008.

[FCCA+06] A Ferreira, R Cruz-Correia, L Antunes, P Farinha, E Oliveira-Palhares, D W. Chadwick, and A Costa-Pereira. How to Break Access Control in a Controlled Manner. In *Proceedings of the 19th IEEE Symposium on Computer-Based Medical Systems*, 2006.

[FCF+09] Ana Ferreira, David Chadwick, Pedro Farinha, Ricardo Correia, Gansen Zao, Rui Chilro, and Luis Antunes. How to Securely Break into RBAC: The BTG-RBAC Model. In *Proceedings of the 2009 Annual Computer Security Applications Conference*, December 2009.

[FKC07] David F. Ferraiolo, D. Richard Kuhn, and Ramaswamy Chandramouli. *Role-Based Access Control*. Artech House, second edition edition, 2007.

[GMPT01] Christos K. Georgiadis, Ioannis Mavridis, George Pangalos, and Roshan K. Thomas. Flexible Team-Based Access Control Using Contexts. In *Proceedings of the sixth ACM symposium on Access control models and technologies (SACMAT)*, May 2001.

[MCMD11] Srdjan Marinovic, Robert Craven, Jiefei Ma, and Naranker Dulay. Rumpole: A Flexible Break-Glass Access Control Model. In *Proceedings of the 16th ACM symposium on Access control models and technologies (SACMAT)*, June 2011.

[MJ10] Haramlambos Mouratidis and Jan Jürjens. From Goal-Driven Security Requirements Engineering to Secure Design. *International Journal of Intelligent Systems*, 25(8), 2010.

[OMG11a] OMG. OMG Business Process Modeling Notation. available at: `http://www.omg.org/spec/BPMN/2.0/`, January 2011. Version 2.0, formal/2011-01-03, The Object Management Group.

[OMG11b] OMG. Unified Modeling Language (OMG UML): Superstructure. available at: `http://www.omg.org/spec/UML/2.4.1/`, August 2011. Version 2.4.1, formal/2011-08-05, The Object Management Group.

[OMG12] OMG. Object Constraint Language Specification. available at: `http://www.omg.org/spec/OCL/2.3.1/`, January 2012. Version 2.3.1, formal/2012-01-01, The Object Management Group.

[OP03] Sejong Oh and Seog Park. Task-Role-Based Access Control Model. *Information Systems*, 28(6), 2003.

[Pov00] Dean Povey. Optimistic Security: A New Access Control Paradigm. In *Proceedings of the 1999 workshop on New security paradigms (NSPW)*, 2000.

[RD98] Manfred Reichert and Peter Dadam. Adept_flex-Supporting Dynamic Changes of Workflows Without Losing Control. *J. Intell. Inf. Syst.*, 10(2), 1998.

[RHE05] Nick Russell, Arthur H. M. Ter Hofstede, and David Edmond. Workflow Resource Patterns: Identification, Representation and Tool Support. In *Proceedings of the 17th Conference on Advanced Information Systems Engineering (CAiSE)*. Springer, 2005.

[RRMD09] Manfred Reichert, Stefanie Rinderle-Ma, and Peter Dadam. Flexibility in Process-Aware Information Systems. In Kurt Jensen and Wil M. Aalst, editors, *Transactions on Petri Nets and Other Models of Concurrency II*, pages 115–135. Springer, Berlin, Heidelberg, 2009.

[RvdAH06] Nick Russell, Wil M.P. van der Aalst, and Arthur H. M. Ter Hofstede. Exception Handling Patterns in Process-Aware Information Systems. In *International Conference on Advanced Information Systems Engineering (CAiSE)*, 2006.

[SCFY96] Ravi Sandhu, Edward Coyne, Hal Feinstein, and Charles Youman. Role-Based Access Control Models. *IEEE Computer*, 29(2), 1996.

[SM10] Mark Strembeck and Jan 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)*. Springer, October 2010.

[SM11] Mark Strembeck and Jan Mendling. Modeling Process-related RBAC Models with Extended UML Activity Models. *Information and Software Technology*, 53(5), 2011.

[SN04] Mark Strembeck and Gustaf 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.

[SSMB11] Sigrid Schefer, Mark Strembeck, Jan Mendling, and Anne Baumgrass. Detecting and Resolving Conflicts of Mutual-Exclusion and Binding Constraints in a Business Process Context. In *OTM Conferences (1) 2011, Proc. of the 19th International Conference on Cooperative Information Systems (CoopIS)*. Springer, October 2011.

[Str10] Mark Strembeck. Scenario-Driven Role Engineering. *IEEE Security & Privacy*, 8(1), 2010.

[SWS12] Sigrid Schefer-Wenzl and Mark Strembeck. Modeling Context-Aware RBAC Models for Business Processes in Ubiquitous Computing Environments. In *Proc. of the 3rd International Conference on Mobile, Ubiquitous and Intelligent Computing (MUSIC), IEEE*, June 2012.

[TCG04] Kaijun Tan, Jason Crampton, and Carl A. Gunter. The Consistency of Task-Based Authorization Constraints in Workflow Systems. In *Proceedings of the 17th IEEE workshop on Computer Security Foundations*, June 2004.

[vdARD07] Wil M. P. van der Aalst, Michael Rosemann, and Marlon Dumas. Deadline-based Escalation in Process-Aware Information Systems. *Decis. Support Syst.*, 43(2), 2007.

[WA06] Janice Warner and Vijayalakshmi Atluri. Inter-instance authorization constraints for secure workflow management. In *Proceedings of the eleventh ACM symposium on Access control models and technologies (SACMAT)*, June 2006.

[WBK03] Jacques Wainer, Paulo Barthelmess, and Akhil Kumar. W-RBAC - A Workflow Security Model Incorporating Controlled Overriding of Constraints. *International Journal of Cooperative Information Systems (IJCIS)*, 12(4), 2003.

[WRR07] Barbara Weber, Stefanie Rinderle, and Manfred Reichert. Change Patterns and Change Support Features in Process-Aware Information Systems. In *International Conference on Advanced Information Systems Engineering (CAiSE)*, 2007.

[WSM08] Christian Wolter, Andreas Schaad, and Christoph Meinel. Task-Based Entailment Constraints for Basic Workflow Patterns. In *Proceedings of the 13th ACM symposium on Access control models and technologies (SACMAT)*, 2008.