Karlsruhe Reports in Informatics 2011,38
Edited by Karlsruhe Institute of Technology,
Faculty of Informatics
ISSN 2190-4782

Embedding 'Break the Glass' into Business Process Models

Silvia von Stackelberg, Klemens Böhm and Matthias Bracht

2011

Fakultät für **Informatik**

Embedding 'Break the Glass' into Business Process Models

Silvia von Stackelberg, Klemens Böhm and Matthias Bracht

Karlsruhe Institute of Technology (KIT), 76131 Karlsruhe, Germany

Abstract. Break the Glass (BTG) is an important feature for authorization infrastructures, as it provides flexible access control in unforeseen cases and emergencies. But current realizations have two drawbacks: (1) they neglect the need to manage authorization steps and (2) they do not take immediate process context into account. Our approach in turn embedds BTG functionality into business processes (BPs): the steps to decide for Breaking the Glass and the obligations compensating a BTG access for data are parts of the BPs. To support process designers in embedding BTG steps and obligations, we introduce an expressive annotation language for specifying BTG tasks for BP models. In particular, our language allows process designers to (1) take BP context into account, and (2) specify security constraints for responsible role holders performing BTG tasks. Using our approach, one can efficiently specify and use context-aware BTG functionality for BPs.

Keywords: Security in business processes; business process modelling; annotation language; immediate business process context

1 Introduction

Problem statement. Security mechanisms are important for Business Process Management (BPM). For instance, authorization constraints specify which roles may perform a task or access certain data. However, such mechanisms sometimes are too rigid, and more flexibility is needed. To illustrate, emergencies (e.g., in E-health) and disaster management necessitate rights to access data in exceptional situations. Thus, a trade-off between security on the one hand and flexibility on the other hand needs to be facilitated.

A seemingly straight forward way to realize this is to furnish temporary roles with predefined rights, so-called *super users*, which must be used in emergency cases only. This however requires the management of emergency accounts, including notifications to users and maintenance of authorizations after usage to avoid misuse. Another approach to provide flexibility is to give existing roles the necessary rights in particular situations. The so-called *Break the Glass (BTG)* principle allows users to overcome access denials in exceptional cases [1]. The designer specifies in advance who, in exceptional cases, will have access rights he normally does not have. In line with [6], the prerequisites to "break the glass" from the application perspective are: (1) regular access is denied, (2) BTG access is foreseen for the exceptional case, (3) a user explicitly asks for access, (4) optionally, another user has to agree to this access. We call the sequence of steps when users ask for exceptional access *BTG steps* in the following. Next, *obligations* typically are part of BTG, i.e., operations that compensate[1] for the security violations. Obligations can be triggered immediately after breaking the glass (synchronously) or later (asynchronously).

*Example 1 (***E-Employment).** The aim of the Dutch "Accreditation of Prior Learning" (APL) [12] is to generate a certified personal competence profile, to be used for job applications. In the regular case, the CV of a candidate participating in APL needs the approval of an assessor before it may be sent to a company as part of an application. But a job seeker might sometimes wish to apply without such an approval. In this case, a candidate submits his unapproved CV, i.e., the candidate breaks the glass. This results in the obligation that the assessor has to be informed. Later, when approval is issued, the approved CV is sent to the company another time. Here, the point of time when sending the CV again depends on the context of a business process (BP), i.e., when the approval takes place.

We envision to integrate BTG functionality into BPMS. This is new and challenging, because existing approaches providing authorization infrastructures for BTG (e.g., [1], [6], [11] and [7]), do not cover the following aspects: (1) Modelling BTG steps and obligations as part of the BP and executing them. (2) Considering BP-specific features, BP context in particular.

Regarding (1), related work leaves the execution of BTG steps and obligations to the application and views them as black boxes. However, a BTG access typically consists of several steps. The same holds for obligations. This asks for mechanisms to embed BTG steps and obligations into the BP, since the modelling of such steps and their execution is exactly the purpose of BPMSs.

The development of context-aware systems is a challenging research area. Most approaches take environmental context into account. *Immediate process context* in turn is information that characterizes the process itself. It refers to the execution state of a process instance, such as the state of tasks, associated entities, or objects to be accessed [18]. Regarding (2), combining immediate BP context with BTG functionality has several advantages, as we will explain in Section 2. But because existing work is generic, it does not take immediate BP context for BTG realizations into account.

Goals and Challenges: Our overall goal is to integrate BTG functionality into the BP and to have it executed by a BPMS. By doing so, we take BP context into account. To accomplish this, we envision the following steps:

– *G1: Facilitating the embedding of BTG steps and obligations into BPs.* Without any support for the embedding, process designers have to model both

[1] We use the term *compensation* to react for a BTG access. As a data access cannot be undone, it is at least mitigated by compensating actions.

the process logic and the security constraints for BTG functionality by hand. This requires profound security knowledge and thus is error prone; and it is time-consuming. Thus, there should be support at the process-modelling level. By using annotations for process models, designers can use the modelling primitives they are used to specify restrictions for the process. For example, [9], [17], and [21] specify security constraints for process models by means of annotations. We develop an annotation language for BP models representing BTG functionality.

– *G2: Context-aware annotation language.* As BP context is important for BTG functionality, the annotation language has to provide support for the coupling of BTG tasks with contextual information.
– *G3: Design and realization of an infrastructure supporting context-aware BTG.* To perform BTG, we make use of a secure BPMS [13]. Supporting BTG requires extensions for the transformation, for the management of BP context, and for authorizations for BTG tasks: The *transformation* results in an extended process model enriched with canned, generic process fragments representing BTG steps and obligations as well as in BP security policies in XACML [5] with embedded BTG authorization rules. The *context manager* component has to capture and to provide missing BP-context information, such as information on subjects, to the BP Engine. Further, authorization components have to decide on access requests for BTG.

As these issues are broad, we address G1 and G2 in this report (and leave the remaining issue to a future publication). It is challenging to already achieve Goals G1 and G2, for the following reasons:

– As the embedding of BTG functionality into BP is new, the design of an expressive annotation language asks for a systematic requirements analysis to exactly identify the expressiveness needed for representing BTG tasks.
– This results in the specification of a comprehensive set of expressions to represent BTG functionality (e.g., who is authorized for particular BTG tasks, and when).
– Existing approaches do not feature the coupling of immediate BP context with BTG functionality. To support this, we develop a representation of BP context so that process designers can easily use it for the specification of BTG functionality.

Contributions: Following the ”security by design” principle, we have developed new concepts to embed BTG functionality at the BP modelling layer. ”Embedding” means that potential BTG steps and obligations are integrated into a BP, and BP context is taken into account.

In particular, we make the following contributions:

– *Motivation.* We list advantages of using immediate context information for BTG functionality. As context can be any information, we provide expressions for the specification of context relevant for an application.

- *Specification of an annotation language allowing to represent BTG steps and obligations.* In particular, it allows to specify actors involved in BTG steps and BP-context-specific constraints for BTG options. Process designers can use these constructs instead of modelling BTG functionality explicitly. Further, they can use annotations for BTG together with existing security-annotations for BPMN [15], e.g., [9], [17], and [21]. We specify generic process fragments the BPMS has to execute in order to fulfill the specifications contained in annotations. These process fragments represent tasks of actors involved in BTG steps and obligations.
- *Evaluation of our approach.* Our evaluation is twofold: We first analyze the gain in time of embedding BTG functionality into BP applications in general, as opposed to programming BTG steps and obligations by hand. We then analyze the benefit of using an annotation language compared to the usage of conventional BP modelling primitives.

By following our approach, one can smoothly embed BTG functionality into BPMS.

Paper structure: We motivate context-aware BTG functionality in Section 2. Section 3 lists requirements. Section 4 features an example. Section 5 describes the annotation terms for BP models. Section 6 evaluates our approach, Section 7 discusses related work, and Section 8 concludes.

2 Motivation for contextual BTG Functionality

Immediate BP context relevant for BTG can be information on the core BP aspects regarding the functional, behavioral, organizational, operational, and data information that is sufficient for the execution of process instances. We borrow these categories from [18]. – Coupling BTG functionality with BP context has the following advantages:

(1) BTG steps may comprise BP-context-specific constraints: BTG steps might be restricted by control-flow constraints on BP schemas. For example, in the APL scenario, there might be several approval steps for a CV. A candidate might be allowed to send his CV as part of a BTG action only if the assessor has already approved some part of it (e.g., education, job positions). Such a constraint can be expressed by referring to the *execution state* of a BP instance. However, existing BTG approaches do not allow to specify this.

(2) Specification of constraints for obligations: Constraints for the execution of obligations can be specified in the same way as for BTG steps. In the APL scenario, there might be one or several assessors involved in an approval. An example of a constraint is to send an email only when several assessors have been involved in the accreditation.

(3) Specification of BP context for obligation parameters: In general, obligations are parameterized. For example, an obligation might say that an individual who accesses a data object in parallel to a BTG access on this data must be informed about the respective BTG action. In our example, the system must pass the email address of that individual to the application executing the obligation.

By using BP-context information on *associated entities*, e.g., the individuals who currently access the same data, the system might determine the receivers of the email automatically.

(4) Triggering asynchronous obligations: Synchronous obligations are triggered immediately after the BTG action. Asynchronous obligations are triggered at an absolute or a relative point in time. In the APL example, the *execution state of a BP* determines when an obligation takes place (i.e., send the CV again after the task 'approval' is finished). Being able to refer to execution states of tasks gives way to asynchronous obligations.

As these advantages are essential it is important to combine BTG functionality with BP context.

3 Requirements for Annotation Language

To analyze the expressiveness of a language allowing to specify BTG functionality, we have studied BTG use cases in two different real-world application scenarios, an E-employment and an E-health application. Following these analyses, a BTG-annotation-language must support the following aspects:

R1: *Security constraints for BTG users:* BTG functionality has to be provided in a controlled way, i.e., it must be specified at design time who shall obtain the BTG rights, namely to break the glass, to access data, and to repair the glass. Thus, the BTG vocabulary must distinguish different types of users involved in a BTG action. The vocabulary must allow to specify authorization and authentication constraints for these users. The latter ones possibly differ from the regular application case.

R2: *BP-context-specific constraints:* It must be possible to specify the start time for BTG steps or obligations, i.e., *when* the tasks have to be performed, by taking the BP context into account. Example 1 has motivated this. Further, it should be possible to represent conditions for the execution of BTG steps and obligations, i.e., *whether* tasks have to be performed. To illustrate, one might specify that an obligation is needed only if an approval takes longer than 2 weeks (temporal condition), or if an external assessor has worked on the approval (causal condition). In particular, asynchronous obligations that rely on BP-context-specific conditions must be possible. The specification of BP context must be user-friendly [8]. This means that process designers should not have to deal with the BP-engine-internal representation of BP context, but should be able to specify BP context at the abstraction level of BP models.

R3: *Parameters for obligations:* It must be possible to specify BP-context-specific parameters of obligations (e.g., associated entities), cf. Example 1.

R4: *Data objects.* As BTG concerns exceptional access to data, the BTG vocabulary has to provide options for specifying data objects to be accessed. We assume that process designers have specified access rules for data objects for the regular case.

R1-R3 are issues that are currently open. Solutions exist for R4, but they need to be integrated into the language envisioned.

4 E-health Scenario

Until now, we have illustrated the use of context-aware BTG functionality for the E-employment domain. Our example has shown that exceptional access for data is needed for particular situations. To illustrate that our concepts have indeed a broad usage, we now switch to examples from another domain, namely E-health. This domain has strict regulations for emergency handling. The healthcare scenario will serve as our running example throughout this paper. To show the usage of BTG, we describe the regular case as starting point, and use annotations for BTG functionality later on.

Example 2 (**E-health).** Figure 1 shows a BP model of the visit of a patient to a physician, which we deem self-explanatory. We assume that health-record data is stored externally (see pool ”HRS” for health-record storage in Figure 1). Only authorized and authenticated users should have access to this data.

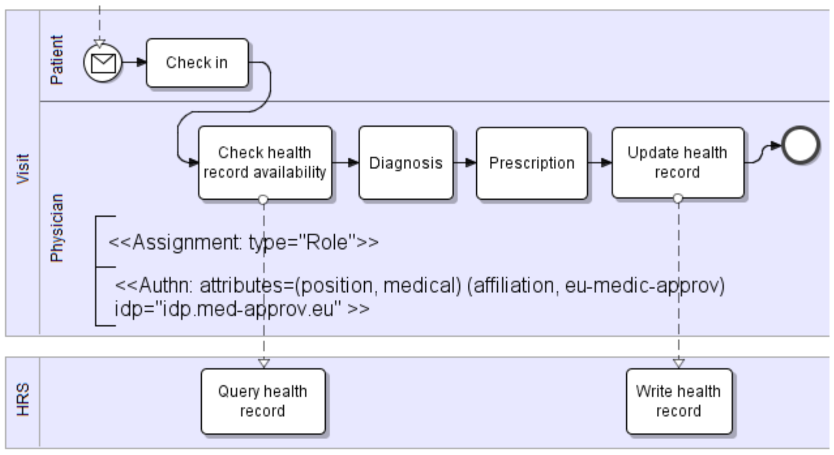

Visit
Patient
Physician
Check in
Check health record availability
Diagnosis
Prescription
Update health record
<<Assignment: type="Role">>
<<Authn: attributes=(position, medical) (affiliation, eu-medic-approv) idp="idp.med-approv.eu" >>
HRS
Query health record
Write health record

Fig. 1. Process Model for E-health Scenario

We now illustrate security constraints for tasks for the regular case.

Example 3 (**E-health (cont.)).** There are two security annotations, of different type, namely authorization and authentication, in our example process model in Figure 1. To express such constraints, we use the language from [9]. Both annotations refer to all activities of the lane ”Physician”. The term ≪*Assignment: type=”Role”* ≫ means that holders of role ”physician” are authorized to perform tasks of this lane. The authentication term ≪*Authn: attributes=(position, medical) (affiliation, eu-medic-approv) idp=”idp.med-approv.eu”*≫ requires individuals to have the position ”medical” (e.g., must have passed an exam) and a European approval to work as medical practitioner. The Identity Provider (IdP) specified has to approve these attributes.

The activities "Check health-record availability" and "Update health record" of the lane "physician" ask for access to data. The first one requires a read access, the second one needs a write access. Only authorized persons are able to access this data, such as the family doctor of a patient. Data access policies (e.g., sticky policies) specify this.

Example 4 (**E-health (cont.)**). In a life-threatening situation, other members of the medical staff might need access to the data. This can be realized by *Breaking the Glass*, i.e., physicians which are not authorized in the regular case access the record in a controlled way. In a real-world scenario, "Breaking the glass" results in many obligations, such as auditing the data access and informing the familiy doctor, among others. We discuss exemplarily one obligation in the following:

O1: At the end of the treatment process, the physician has to write a report and send it to the family doctor.
This obligation is asynchronous because it refers to a later point in time.

5 Design

In this section we describe how we embed options for breaking the glass into BP models. We first describe the different BTG roles and motivate annotating process models with BTG functionality. We then say how we represent BP context. Finally, we describe the syntax and the semantics of the annotation language and give a concrete example.

5.1 BTG roles

We introduce roles having BTG rights in the following. In line with [2], we distinguish three types of users involved in a BTG option: the first type are users who have the right to break the glass, i.e., users who activate a BTG case (e.g., patients who decide). We call the corresponding role *BTG Activator Role*. Second, the *BTG Access Role* are users who have the right to access a resource if BTG has happened (physicians in our example). In practice, it is possible that process participants have both rights. Third, the *BTG Compensator Role* are users who are allowed to perform obligations in the BTG case.

5.2 Embedding BTG functionality into business processes

From the perspective of the components enforcing authorizations for data access, the application has to perform several tasks for a BTG option and the compensating obligations [6]: First, the authorization system must deny a regular request, but offer the possibility for a BTG access to the application. In the next step, the holder of *BTG Access Role* must explicitly ask for access under BTG conditions. Further, a holder of *BTG Activator Role* has to agree for breaking the glass, enabling a holder of *BTG Access Role* to access the data.

A straightforward way to handle these steps would be to code them as an external application. But this requires a coupling with the authorization infrastructure to manage the various roles. Breaking the glass usually leads to obligations to be executed. These obligations can be complex applications. Further, contextual constraints can relate to obligations, and other role holders might be involved. But with this design alternative, developers have to constrain application logic with BP context by hand.

Design Decision: Our idea is to integrate those steps into the application process. Thus, the BP Engine controls their execution. A BPMS equipped with an authorization infrastructure can enforce the authorizations.

A subsequent question now is how to embed BTG steps and obligations into the BP application. We see several alternatives, with two extremes: to represent them within the process model or to dynamically adapt process instances at runtime if needed (ad-hoc adaptation).

With the first extreme, process designers embed any options for breaking the glass in the BP model, using conventional modelling primitives. Process events and gateways can represent these options for exception handling. This means that a process instance can perform any BTG case or not. This is likely to lead to very complex BP models, because a single BTG case already consists of a sequence of tasks and might have many corresponding obligations. However, BTG functionality is only needed in exceptional cases, and whether it is needed is known only at runtime. This observation leads to the second extreme, namely to enable ad-hoc changes at runtime, meaning that process instances deviate from the specified process model. This can be interpreted as a case of exception handling, and it affects only single process instances. Process designers have to specify allowed deviations in advance. This approach requires a BP Engine that is capable to deal with ad-hoc changes at runtime. Currently, there is only little support in BP Engines for this (e.g., by the AristaFlow BPM Suite [3]). If platform-independence is an issue, a solution currently cannot rely on these features.

Design Decision: Our approach is a middle ground. We let process designers specify BTG options in a BP model[2] with specific annotation vocabulary. The BPMS transforms these annotations by extending the BP schema with canned process fragments and executes them as part of the BP. By means of annotation terms, process designers specify who will have the various BTG rights for data and the constraints for enabling BTG.

5.3 Formalizing BP Context

It is the task of the Engine to manage BP context. As BP context is important for BTG, we need a way to represent it in the BP model. One way for process designers is to specify constraints for BP context relevant for BTG by using

[2] In line with our security-annotation language, we represent BTG options for BP models in BPMN. But our approach is sufficiently general. It can be applied to other representations for BP models as well.

the internal representation of the BP Engine. But this is error-prone and time-consuming, since process designers typically are not familiar with the internal representations.

Design Decision: We propose a vocabulary to represent BP context in BTG annotations. We formalize BP context on the abstraction level of BP models by introducing functions for tasks and data that return values representing the BP context. These functions enable the specification of associated entities, tasks, and data objects to be used for the representation of temporal and causal BP-context constraints in the annotations. Our representation of BP context also reflects that tasks can be executed many times. The BPMS transforms these specifications into representations the BP Engine can handle. This addresses R2.

We now define the syntax and semantics of BP contraints.

Definition 1 (Syntax BP Constraint) *A BP-context constraint (BP-CC) has the following* syntax*:* [3]

`BP-CC` := `S-BP-CC` | `C-BP-CC`
`S-BP-CC` := `function(argument)` | `function(argument)` $f - op$ `value`
`C-BP-CC` := `S-BP-CC` *c—op* `S-BP-CC` | `C-BP-CC` *c—op* `S-BP-CC` | `function(S-BP-CC)`
`f-op` := $> | < | <= | >= | == | \in | \notin$
`c-op` := $\wedge | \vee | == | \neq$
`function` := `data-user` | `owner` | `start-time-access` | `end-time-access` |
`performer` | `data-access` | `start-time-exec` | `end-time-exec`
`argument` := `s-argument` | `c-argument`
`s-argument` := `data-object` | `activity`
`c-argument` := `s-argument, number`

Simple BP-context constraints `S-BP-CC` are either expressions of a `function` with an `argument` or a `function` with an `argument`, an operator `f-op` and a `value`. Complex BP-context constraints `C-BP-CC` are combinations of simple BP-constraints by operators `c-op`, such as logical ones, or nested functions. Simple arguments `s-argument` are `data-object` or `activity`. A complex argument `c-argument` contains a simple argument and a number. The functions `data-user`, `owner`, `start-time-access`, `end-time-access` operate on the simple argument `data-object`, or on the complex argument `(data-object, number)`. The functions `performer`, `data-access`, `start-time-exec`, and `end-time-exec` have the simple argument `activity`, or the complex argument `(activity, number)`.

Definition 2 (Semantics BP Constraint) *The meaning of a BP-context constraint depends on the return value of* `function` *as follows:*

– *The functions* `performer(task)`*,* `data-user(object)`*,* `owner(object)` *return subjects related to a BP instance, namely the individual performing a*

[3] the notation "|" is the logical "XOR"

task instance, the individual accessing a data object, and the data owner respectively.

- *The functions* `start-time-exec(task)` *and* `end-time-exec(task)` *return the start and end times of the execution of a task.*
- *The functions* `start-time-access(object)` *and* `end-time-access(object)` *return the start time and end time of access to a data object.*
- `data-access(task)` *returns the set of data objects accessed by a task.*
- *The value* `number` *in a complex argument specifies the number of executions or data accesses to be considered. The result of a function on a complex argument is a set of values.*

We account on loops in process execution. To illustrate, several subjects perform an activity, or many subjects access a data object over time. We represent this by the value `number` of complex arguments. A specification "`number=1`" means that only the latest execution (or data access respectively) is of interest, while "`number=inf`" specifies that the entire execution history is considered.

This set of functions is sufficient for the applications we have studied. Using these functions, process designers can represent a comprehensive set of BP-context-constraints within annotations for BTG steps and obligations. In particular, they can specify temporal and causal constraints regarding the start time of BTG steps and obligations as well as constraints on their execution.

Example 5 (**BP Context Constraints).** To express that BTG is only allowed for adults, we specify a causal constraint on the execution for BTG steps by `Exec=performer(activity-ID).age` $\geq$ `18`. We specify an asynchronous start time for an obligation that depends on the execution time of an activity by `Start` = `end-time-exec(activity-ID)`. To express that an obligation has to be executed if several performers are involved (see APL example in Section 2), we formulate `Exec` = `performer(activity-ID-1)` $\neq$ `performer(activity-ID-2)`.

5.4 Specification of BTG steps for BP models

We now describe our annotation language for BTG steps. It features the specification of security aspects (exceptional access to data, authorizations for tasks, authentications) and of BP-context constraints for BTG steps. Process designers annotate tasks of a BP model with BTG options, because BTG data access happens during the execution of tasks.

Definition 3 (Syntax of BTG Annotation Language) *An annotation term to describe BTG steps with security constraints has the following syntax:*

```
≪BTG: "object=list($objectname)"
"right=list($right-type)"
{optional-assignments} ≫
```
with
```
right-type := read | update
optional-assignments := BTGActivatorSpec ? BTGAccessorSpec ?
```

```
BTGStartCondition ? BTGExecCondition ? BTGObligations ?  BTGInsertMode
BTGActivatorSpec := BTGActivatorRole {BTGActivatorAuthn}
BTGAccessorSpec := BTGAccessorRole {BTGAccessorAuthn}
BTGActivatorRole := "BTGActivator=$rolename" | "BTGActivator=$username"
BTGAccessorRole := "BTGAccessor=$rolename" | "BTGAccessor=$username"
BTGActivatorAuthn := "AuthnBTGActivator-attr = list(attribute, value)
{idp=$idp-address}"
BTGAccessorAuthn := "AuthnBTGAccessor-attr = list(attribute, value)
{idp=$idp-address}"
BTGStartCondition := "Start=$BP-CC"
BTGExecCondition := "Exec=$BP-CC"
BTGObligations := "Obligations=list($obligation-ID)"
BTGInsertMode := "Insert=seq | par"
```

A BTG annotation, starting with "≪BTG:", contains a set of assignments (any statements indicated by "....=..." above) for a specified vocabulary, and ends with "≫". The assignments for `object` and `right` are obligatory. Further, there are `optional-assignments`, as listed[4].

To represent authorizations for BTG steps, we need two out of three BTG roles, namely *BTG Activator Role* and *BTG Access Role*. This accounts for R1. Process designers assign role holders or users which typically perform tasks of the regular application case to BTG roles. This means that they have the right to perform BTG tasks. This avoids the maintenance of particular BTG roles, as discussed in the introduction. The specifications for *BTG Activator* and *BTG Accessor* can have optional refinements by `BTGActivatorAuthn` or `BTGAccessorAuthn`. The brackets "{...}" denote this option. One can specify `BTGStartCondition` by `Start`, and a `BTGExecCondition` by `Exec`. To represent obligations, we proceed as follows: One BTG action can have many obligations, and obligation specifications can be complex. Thus we specify the list of related obligations in the BTG annotation and annotate each obligation separately for the BP model. Insert can be specified by "seq" or "par".

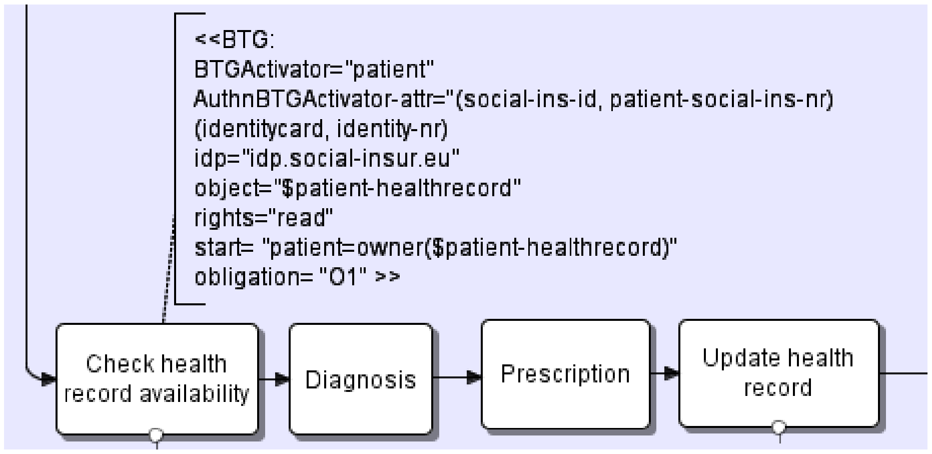
Definition 4 (Semantics of BTG Annotation Language) *The parameter* `right` *specifies the nature of the access to a data* `object` *for which the glass can be broken. The* `right` *can be to* `read` *or to* `update` *a data object.* `BTGActivatorSpec` *and* `BTGAccessorSpec` *determine the roles for* `BTG Accessor` *and* `BTG Activator` *as well as the authentication requirements* `BTGActivator Authn` *and* `BTGAccessor Authn` *for their holders. The parameters* `AuthnBTG Activator-attr` *(and respectively* `Accessor-attr`*) specify attributes the system uses for the authentication of these two different role holders (or users). One can specify attribute-value-pairs and optionally an IdP.* `BTGStartCondition` *specifies constraints on the start of BTG steps and* `BTGExecCondition` *constraints on the execution. Both can contain temporal or causal constraints. We assume that BTG steps usually have to be executed immediately, but we provide some flexibility for process designers by means of parameter* `Start`*. It states when the BTG steps have*

[4] the notation "?" is the logical "OR", and the notation "|" is the logical "XOR"

to be executed, and which constraint must hold at this point in time. In contrast, `Exec` *specifies constraints that must hold for executing BTG steps in general. By specifying* `Exec`*, BTG steps can only take place at some time during the process execution if the constraints are fulfilled. Further,* `obligations` *specifies a list of obligation-IDs which have to be executed when the glass has been broken. To provide some flexibility to process designers, they can control the insert mode for the fragment by specifying* `Insert`*. The value "seq" means that the obligation is inserted in sequence to the regular process, and "par" means that the process fragment runs in parallel. The default value is "seq".*

If there is no specification for `BTGAccessorRole`, role holders of the BTG-annotated activity get the access rights. If `BTGAccessorRole` is specified but `BTGActivatorRole` is not, `BTGAccessorRole` is used for `BTGActivatorRole` instead. The specification of `AuthnBTGActivator-attr` is in line with our annotation term for authentications [9]. We now illustrate a BTG annotation in our running example.

Example 6 (**E-health (cont.).** To enable BTG functionality for health records of patients, we make use of the BTG-annotation term for activity "Check health-record availability". Figure 2 graphs the annotation.



```
<<BTG:
BTGActivator="patient"
AuthnBTGActivator-attr="(social-ins-id, patient-social-ins-nr)
(identitycard, identity-nr)
idp="idp.social-insur.eu"
object="$patient-healthrecord"
rights="read"
start= "patient=owner($patient-healthrecord)"
obligation= "O1" >>
```

Fig. 2. Annotation of Activity "Check health-record availability"

The meaning of the annotation is as follows: In the regular case, patients do not have to perform tasks. To provide the "break the glass" option, a process designer assigns patients whose health record might be accessed to *BTG Activator*. In this case, the patients have to agree to break the glass. As this is a security-relevant task, the process designer asks for authentication for the *BTG Activator*. In other words, the patient now has to authentify himself, and this needs to be modelled. The authentication specification says that the IdP must authenticate a patient by the AuthnBTGActivate-attributes social insurance number and identity-card number. The objects to be accessed are health records of the patients. We set "read" access rights for the BTG case. The `BTGStartCondition` (`Start=...`) specifies that the glass can only be broken if the patient is the owner of the data object.

A BTG annotation means that the BPMS has to provide options for BTG steps, as described in Section 5.2. To embed these steps into the BP, we rely on the fundamental technique of process fragments, enabling to re-use parts of process structures. Our understanding of a process fragment is similar to [20]: We see process fragments as independent process structures, which are "underspecified" at design time. Using our approach, executable BPs require specifications for BP-context constraints as well as for security (data access, authorizations and authentications for specific role holders of the application process), which is different to [20]. Thus, underspecified means in our context that process fragments are completely modelled (i.e., they contain connected tasks), but do not contain these specifications. By doing so, process fragments are generic and can be used in any BP model.

Figure 3 shows the process fragment for BTG steps. It represents the execution order for BTG tasks as well as conditions on BP-context constraints. The annotated role assignments specify authorizations for BTG role holders. Holders of role BTG Accessor can perform activity "ask-for-BTG" and activity "data access". Role holders of BTG Activator have to execute the activity "agree-BTG". To ease presentation, we have omitted user interactions and interactions with security components.

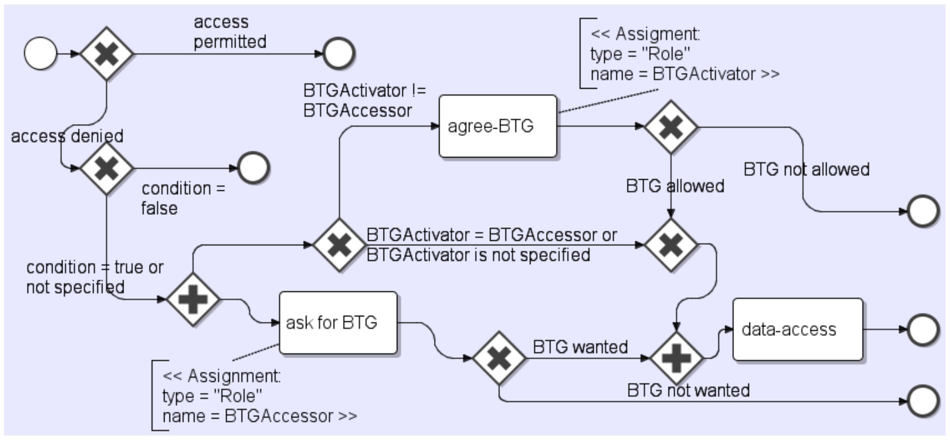

access permitted
access denied
condition = false
condition = true or not specified
BTGActivator != BTGAccessor
agree-BTG
<< Assigment: type = "Role" name = BTGActivator >>
BTG allowed
BTG not allowed
BTGActivator = BTGAccessor or BTGActivator is not specified
ask for BTG
<< Assignment: type = "Role" name = BTGAccessor >>
BTG wanted
BTG not wanted
data-access

Fig. 3. Process Fragment for BTG steps

Regarding the transformation, the system substitutes in a first step each BTG annotation with a BTG-process fragment. It specifies the extended BP model by substituting, for example, the generic BTG roles by role information contained in the annotations. In the last step, it generates the access-control policy.

5.5 Specification of Obligations

We distinguish between obligation patterns and application-specific obligations. Obligation patterns are sequences of tasks that are likely to be required in many applications. Typical obligation patterns are: send email, send sms, or log data access. Application-specific obligations have little potential for re-use, such as the obligation that an assessor has to approve a CV again.

We propose to handle obligation patterns in the same way as BTG steps. This means that we provide a vocabulary to describe obligation patterns and represent them as language primitives.

An obligation annotation consists of a sequence of assignments to specify the obligation. Process designers can either annotate an activity with both, the option for BTG and the related obligations, or annotate other activities that the BPMS executes later with obligations. The meaning is as follows. The execution time of an annotated activity determines the earliest execution time of an obligation, that is after the execution of the annotated activity.

Definition 5 (Syntax Obligation) *The syntax of obligation annotation terms is as follows:*

```
≪Obligation:
"id=$obligation-ID"
"pattern=$obligationpattern"
{optional-assignments} ≫
```
with
```
optional-assignments := CompensatorSpec ? OParameter ? CompStartCondition ?
CompExecCondition
"OParameter=list(($parametername,$parametervalue))"
CompensatorSpec := CompensatorRole {CompAuthn}
CompensatorRole := "CompRole=$rolename" | "CompRole=$username"
CompAuthn := "AuthnComp-attr = list((attribute, value)) idp=$idp-address"
CompStartCondition := "Start=$BP-CC|$time"
CompExecCondition := "Exec=$BP-CC"
```

Assignments for the parameters `id` and `pattern` are obligatory. The specifications for `CompensatorSpec`, `OParameter`, `CompStartCondition`, and `CompExecCondition` are optional[5].

Definition 6 (Semantics of Obligation) *The semantics of annotation terms for obligations is as follows:*

[5] the notation "?" is the logical "OR"

The `id` *of the obligation is the reference to the BTG annotation. The* `pattern` *parameter specifies the obligation type. We currently support* "SendEmail" *and* "AuditAccess"*. The value of the optional* `OParameter` *gives parameters for the execution of the obligation, in the form of (name, value) pairs.* `CompensatorSpec` *specifies a* `Compensator Role`*, which can be role holders or users allowed to perform the obligation.* `CompensatorSpec` *can optionally contain authentication specifications* `CompAuthn`*. The optional parameter* `CompCondition` *specifies constraints on the start or the execution of an obligation. It can be a BP-context constraint or an absolute or relative value for time.*

Some explanations regarding the rationale behind the previous definitions are in place. `OParameter` depends on the obligation type. We allow the parameters *from*, *to*, *subject*, *body*, and *attachment* for an obligation type *"SendEmail"*. *"AuditAccess"* specifies an audit policy by the parameter *auditpolicy*, and the *start* and *end* time of the auditing. The audit policy describes the objects to be audited. We can determine the parameter values from the BP context. A specification of `CompRole` is required if humans have to perform the obligation. If no humans are involved, and the BPMS executes the obligation automatically (e.g., it performs a logging), `CompRole` does not have to be specified. In line with the authentication requirements for *BTG Accessor* and *BTG Activator*, attribute-value pairs and an IdP can be specified (`AuthnComp-attr`) for these users.

If the parameter `CompStartCondition` is not specified, this means that the obligation has to be executed immediately after the task annotated by an obligation. Otherwise, the parameter `CompStartCondition` determines the start time for executing an obligation. `CompExecCondition` gives BP context constraints on the execution of the obligation. To illustrate, the specification "`Exec = owner(object-ID).age < 18`" says that the obligation has to be executed only if the data owner is under age.

Example 7 (**E-health (cont.)**). In the example scenario, "Breaking the glass" results in one obligation. We now specify O1 in our annotation language. The obligation says: "At the end of the treatment process, the physician has to write a report and send it to the family doctor." Additionally, the recipients of the report have to be specified. O1 in our obligation vocabulary is:

≪Obligation: id="O1"
pattern="SendEmail"
OParameter=((from, "$performer(update-health-record).email"),
(to, "$familyDoctor.owner($patient-healthrecord).email"),
(subject,"Patient Report: owner($patient-healthrecord)"),
(body,"Find attached a report describing my treatment."),
(attachment,"$owner($patient-healthrecord).ReportFile"))
"Start=end-time-exec(update-health-record")" ≫

The constraint `Start= end-time-exec(update-health-record)` represents an asynchronous BP-context condition on the obligation start time. Several parameters for the obligation call, such as "from", "to", are listed as parameter name/value pairs.

Accordingly, we represent obligation patterns as process fragments. Figure 4 shows the process fragment, representing an obligation for sending an Email.

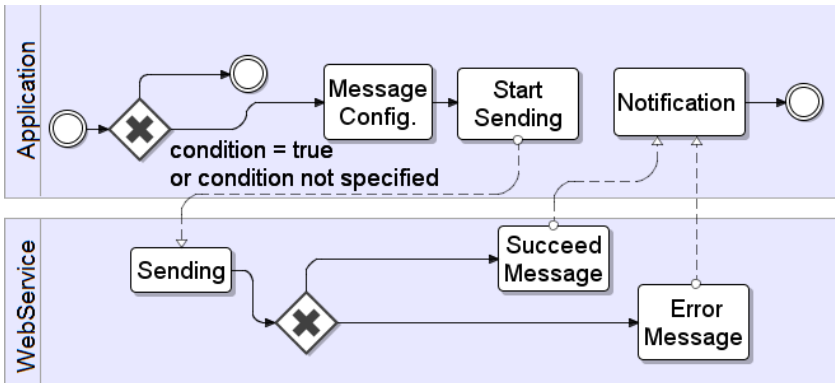


Fig. 4. Process Fragment for Sending Email

6 Evaluation

Our approach provides support for two levels: First, we provide an infrastructure for BTG functionality from the application perspective by embedding BTG steps and obligations into the BP model. The alternative would be to manage BTG steps and obligations isolated from the BP application logic. Second, we provide an annotation language for BTG functionality, enabling process designers to use our annotation language instead of embedding process fragments for BTG steps and obligations into the BP model by hand.

We now describe the initial evaluations of our approach. We analyze the benefit of our approach in terms of reduced effort (i.e., time needed) for these two levels of support. We first quantify the gain of a BPM-infrastructure for BTG functionality. Second, we quantify the effort required for selecting and specifying process fragments by hand, as opposed to using our annotation language.

6.1 Gain of embedding BTG into process models

With conventional approaches, *programmers* have to implement BTG steps and obligations as BP-external application functionality and have to couple this functionality with the process logic and the BP context. Regarding the functionality of BTG steps as shown in Figure 3, programmers have to implement an application that performs BTG steps, including user dialogues for role holders of BTG

Accessor and BTG Activator, and data access. Further, they must implement authorization and authentication functionality for role holders of these tasks.

Setup. We have chosen an experienced programmer skilled with role-based authorization concepts. We have trained him to BTG concepts in general and to our approach. In particular, we have explained the three BTG roles we distinguish and the steps to perform a BTG action. For this interactive, oral training, we have spent one hour altogether.

Results. The programmer has needed around two days to implement an application for BTG steps with the corresponding user interfaces (see column in the middle in Table 1). The effort for implementing obligations depends on their complexity. Our implementer has needed one day to realize the obligation to send an email (see the right column in Table 1).

Alternative	Required Time BTG Steps	Required Time Obligation
1 (programming)	2 days	1 day
2 (modelling fragments)	3 hours	2 hours

Table 1. Evaluation of Embedding BTG Functionality into BP

With our approach to embed BTG steps and obligations into BP models in turn we shift the modelling and configuring of BTG steps and obligations from the programmers to the *process designers*. The secure BPMS is responsible for the execution of BTG steps and obligations and provides authorization and authentication functionality, as well as the evaluation of BP-context-constraints. To model and specify a process fragment for BTG steps, a process designer has needed only around three hours (see Table 1). Thus, our approach reduces the effort to implement BTG steps and obligations significantly.

6.2 Gain of transformed annotations

Process designers who want to embed BTG functionality into process models have the following alternatives to do so: (1) Using our proposed annotation language and the transformation mechanisms, (2) embedding canned sub-processes by hand, (3) modelling BTG steps and obligations by hand. To evaluate the advantages of specifying BTG steps and obligations declaratively, we have compared the effort with the alternatives.

Setup. We have recruited an experienced process designer skilled in BPMN-process-modelling, including the usage of respective tools, and with a basic understanding of role-based authorization concepts. The training has been the same as for the programmer described above.

To rule out that the effort with our approach is lower because of learning effects, we have organized the tasks as follows: The process designer has started with the simplest alternative, namely to use annotations (Alternative 1). His second task has been to embed canned process fragments (Alternative 2), and the third task to model BTG steps by hand (Alternative 3).

Results. For Alternative 1, the process designer had to annotate the E-health process (see Figure 1) with our BTG vocabulary. The task was to represent BTG steps with the specification of an Accessor Role, an Activator Role, and with authentication requirements for both role holders. To assist him for this task, we have furnished him with a brief syntax and semantic description (one page, containing syntax and semantics for annotations). To evaluate Alternative 2, the process designer had to embed a process fragment from a file directory. Further, he had to specify authorization and authentication constraints for role holders of the process model by hand. For Alternative 3, the process designer had to model and configure three activities and eight gateways representing BTG steps, as shown in Figure 3. Further, we asked him to configure the process fragment by binding activities to web services and specifying gateways and the data flow.

Alternative	Required Time BTG	Required Time Oblig.
1 (annotation and transformation)	2 minutes	3 minutes
2 (embedding process fragments by hand)	5 minutes	6 minutes
3 (modelling fragments by hand)	10 minutes	8 minutes

Table 2. Evaluation of Annotation Language

Table 2 summarizes the results. If a process designer uses our annotation language and the transformation mechanisms, he needs around two minutes to specify BTG steps (see the middle column). If process designers plug re-usable process fragments into the process model manually (Alternative 2), the modelling effort is increased. However, one also has to specify authorization constraints for tasks as well as BP-context specific constraints for the process fragment by hand. A process designer needs 5 minutes to do so. Regarding Alternative 3, the modelling by hand, an experienced process designer needs around 10 minutes. The results for the obligation to send an email are similar (see right column in Table 2). This shows a significant gain with our annotation language.

7 Related Work

With respect to the integration of BTG into business processes, work from three research threads is of particular relevance. First, there is a relationship to providing access control policies for BTG. Second, we describe security-related research for BPs with respect to its context-awareness. Third, we analyze security annotation languages for BPs according to express immediate BP context.

BTG access control policies: Several approaches realize BTG by implementing access control policies ([1], [10], [6] and [11]). But they do not feature support for BP context, due to their generic nature, and do not address the management of BTG steps and obligations from the perspective of the application, as we do. Our approach in turn does not focus on a BTG-enabled access-control-policy language, but on an infrastructure for embedding BTG functionality. We manage BTG authorization functionality to some degree by tasks being part of the BP. To illustrate, the BPMS executes process branches with BTG options, instead of offering BTG options by the authorization component, as in [6]. Our approach makes access control rules for BTG easier.

Context-aware security support: Many approaches propose context-aware process management. We classify related work in the following by (a) the understanding of context, (b) the purpose, and (c) the used context model.

Type of context. [18] gives a comprehensive classification of BP context. According to this classification, our work focuses on immediate context. We thereby also consider security and privacy aspects (e.g., authorized role holders, owner of data). The understanding of activity and object context in [16] is similar to our understanding of immediate context. Most recent work on context-aware BP (e.g., [11]) is confined to the context of the *environment* by taking properties such as the physical location of a user, the date, or the temperature, into account. We do not address environmental context.

Purpose. There is a broad range regarding the purposes to support context-aware concepts. We focus here on security aspects for BPs. [16] summarizes well-known approaches on BP-context-aware access control methods. To bind access rights to the execution time of tasks (strict least privilege), the approaches use immediate BP context (e.g., [16]). Further, the realization of Binding and Separation of Duties [4] requires immediate context information at runtime. [19] proposes a context-dependent assignment of actors to roles. Our purpose differs from the discussed approaches. We address context-aware embedding of BTG functionality.

Context Model. Several context models have been proposed in the literature. [8] describes context variables by a so-called context cube, representing the relevant internal context dimensions. [19] specifies a context tree by representing facets. [14] specifies contextual information by means of an ontology. We interpret a process meta-model as context model. This allows to consider process elements as well as their relations.

Security modelling languages: [9], [17], [21], among others, propose annotation languages to represent security constraints in BP models, but lack in the following two aspects: (1) None of them takes exceptional cases, in particular BTG, into account; (2) None of them provides features to represent BP context as part of the annotation language. Our language is the first to cover this.

To our knowledge, our approach is unique in that we embed BP-context-constraints into the BTG-annotation-language, and use this contextual information for the embedding of process fragments by taking authorization rules into account.

8 Conclusions

The "Break the Glass" concept facilitates controlled access to data in exceptional situations. To our knowledge, this article has been first to provide BTG functionality for business processes. As breaking the glass and compensating a BTG action require several tasks, BTG steps and obligations should be embedded in processes. We have shown that using BP context for BTG tasks is essential.

To disburden the process designer from modelling BTG steps and obligations by hand, we have proposed a vocabulary for annotating the process model with BTG functionality. In particular, we take BP context into account. This reduces the design effort significantly.

Acknowledgements: This research has received funding from the Seventh Framework Programme of the European Union (FP7/2007-2013) under grant agreement n^o 216287 (TAS^3 - Trusted Architecture for Securely Shared Services)

References

1. Brucker, A.D., Petritsch, H.: Extending Access Control Models with Break-glass. In: SACMAT. pp. 197–206 (2009)
2. Chadwick (Ed.), D.: Design of Identity Management, Authentication and Authorization Infrastructure, TAS3 Deliverable 7.1, Version 3.0.1 (2010)
3. Dadam, P., Reichert, M., Rinderle-Ma, S., Lanz, A., Pryss, R., Predeschly, M., Kolb, J., Ly, L.T., Jurisch, M., Kreher, U., Goeser, K.: From ADEPT to AristaFlow BPM Suite: A Research Vision has become Reality. In: ER-BPM. pp. 529–531. Springer (2009)
4. Elisa Bertino and Lorenzo Martino and Federica Paci and Anna Squicciarini: Security for Web Services and Service-Oriented Architectures. Springer (2010)
5. Erik Rissanen (ed.): eXtensible Access Control Markup Language (XACML) Version 3.0. Committee Specification 01 (August 2010), `http://docs.oasis-open.org/xacml/3.0/xacml-3.0-core-spec-cs-01-en.pdf`
6. Ferreira, A., Chadwick, D., Farinha, P., Correia, R., Zao, G., Chilro, R., Antunes, L.: How to securely break into RBAC: The BTG-RBAC model. In: ACSAC. pp. 23 –31 (2009)
7. Hafner, M., Memon, M., Alam, M.: Modeling and Enforcing Advanced Access Control Policies in Healthcare Systems with SECTET. In: Giese, H. (ed.) Models in Software Engineering, pp. 132–144. LNCS, Springer (2008)
8. Hallerbach, A., Bauer, T., Reichert, M.: Context-based configuration of process variants. In: TCoB. pp. 31–40 (2008)
9. J. Mülle, S. von Stackelberg, K. Böhm: Modelling and Transforming Security Constraints in Privacy-Aware Business Processes. In: SOCA. pp. 1–4 (2011)
10. Ja'far Alqatawna, Erik Rissanen, B.S.F.: Overriding of Access Control in XACML. In: POLICY. pp. 87–95 (2007)
11. Marinovic, S., Craven, R., Ma, J., Dulay, N.: Rumpole: A Flexible Break-glass Access Control Model. In: SACMAT. pp. 73–82 (2011)
12. Mülle (ed.), J.: Design of a semantic underpinned, secure and adaptable process management platform (1). TAS3 Deliverable 3.1, 1st Iteration (June 2009)
13. Müller, J., Böhm, K.: The Architecture of a Secure Business-Process-Management System in Service-Oriented Environments. In: ECOWS. pp. 49–56 (2011)

14. Najar, S., Saidani, O., Kirsch-Pinheiro, M., Souveyet, C., Nurcan, S.: Semantic representation of context models: a framework for analyzing and understanding. In: CIAO. pp. 6:1–6:10. ACM (2009)
15. Object Management Group: Business Process Model and Notation, V2.0. OMG Available Specification (January 2011), `http://www.omg.org/spec/BPMN/2.0/PDF`
16. Park, S.H., Eom, J.H., Chung, T.M.: A Study on Access Control Model for Context-Aware Workflow. In: INC, IMS and IDC. pp. 1526–1531 (2009)
17. Rodríguez, A., Fernández-Medina, E., Piattini, M.: A BPMN extension for the modeling of security requirements in business processes. Trans. Inf. Syst. – IEICE E90-D, 745–752 (March 2007)
18. Rosemann, M., Recker, J.C., Flender, C.: Contextualisation of business processes. Int. Journ. of Business Process Integration and Management 3(1), 47–60 (2008)
19. Saidani, O., Nurcan, S.: Context-awareness for adequate business process modelling. In: RCIS. pp. 177–186 (2009)
20. Schumm, D., Karastoyanova, D., Kopp, O., Leymann, F., Sonntag, M., Strauch, S.: Process Fragment Libraries for Easier and Faster Development of Process-based Applications. Journal of Systems Integration 2(1), 39–55 (2011)
21. Wolter, C., Schaad, A.: Modeling of Task-Based Authorization Constraints in BPMN. In: BPM. pp. 64–79. Springer (2007)