Karlsruhe Reports in Informatics 2014,16
Edited by Karlsruhe Institute of Technology,
Faculty of Informatics
ISSN 2190-4782

On the Usability of a Break-the-Glass Annotation Language for Process Models

Silvia von Stackelberg, Klemens Böhm, Stefan Grabatin, Jürgen Wäsch

2014

KIT – University of the State of Baden-Wuerttemberg and National Research Center of the Helmholtz Association

On the Usability of a Break-the-Glass Annotation Language for Process Models

Silvia von Stackelberg[1], Klemens Böhm[1], Stefan Grabatin[1], Jürgen Wäsch[2]

[1] Karlsruhe Institute of Technology (KIT), 76131 Karlsruhe, Germany
[2] Hochschule Konstanz University of Applied Sciences, Germany

Abstract. Business process design for real-world applications often requires modelling languages of a certain complexity, interweaving several process perspectives, such as the data and the organizational one, and taking contextual process information into account. Ensuring that such languages are indeed usable is not trivial. This paper describes a usability study for a modelling language with those characteristics. More specifically, the so-called BTG! language features the security concept Break the Glass (BTG) that allows to specify exceptional access to data. We investigate usability characteristics of BTG! such as learnability, efficiency, errors, and satisfaction. Our study has helped to increase the usability of the language significantly. To illustrate, we have replaced the language terms with more intuitive ones and have extended the language with some frequently desired features. Our experiences can be applied to process modelling languages that are similarly comprehensive.

Keywords: Business process modelling, process perspectives, usability study

1 Introduction

Motivation: Business processes have a functional, a behavioral, a data, an organizational, and an operational perspective [3]. In addition, many applications call for support of further perspectives, such as security. However, various process modelling languages focus on the functional and the behavioral perspective/aspect, leaving aside other specific needs of the application. One such need is the necessity to specify access rights for process participants, to give an example. Extensions for process modelling languages [11] covering these other aspects have been proposed. However, specifying several aspects in combination largely continues to be an open issue. But this is important because aspects interleave with each other.

The following short example illustrates the complexity of such *hybrid specifications*: One might want to specify access rights to data for a role holder (i.e., referring to the organizational, the data and the security aspect) contingent on execution of a certain task (i.e., functional and behavioral perspective). – As hybrid specification languages are complex, studying their usability is an important issue. In other words, the questions are: How usable are languages covering several perspectives of business processes?, and: How to improve? There currently

is a lack of usability studies on process modelling languages interweaving several perspectives. Existing work in turn focuses on the functional and the behavioral aspects (c.f. Chapter 4). A usability study for one specific hybrid language is expected to not only be of interest to its designers, but also to those of other hybrid languages. This paper is such a study.

Usability studies typically address the issues *learnability*, *efficiency*, *memorability*, *errors*, and *satisfaction of users* [15]. Frequently, few probands allow to already detect important usability issues [15]. This renders usability tests an economic way to improve a product.

This paper focuses on a hybrid specification language called *BTG!* [21]. *BTG!* features an important security concept. The concept is Break the Glass (BTG), i.e., providing flexible data access in exceptional situations [2], [6]. BTG is a metaphor of breaking the physical glass which protects a fire alarm button against misuse. To illustrate, a fireman might require exceptional rights for accessing ground plan data of buildings for an emergency situation, which he cannot access in the regular case. To provide some control, his supervisor activates the exceptional situation by consenting to that access to the data. As a consequence, the fire brigade must inform the owner of the house on this exceptional data access later on (a so-called obligation in the BTG context). This example shows the complexity of the BTG concept: It requires flexible access control mechanisms, several role holders might be involved, and an exceptional access might require subsequent actions (obligations).

BTG! allows to specify powerful BTG features in business processes, tightly interwoven with the actual process model, in an abstract, descriptive way, without having to be familiar with its realization. Supporting BTG in business processes provides flexibility on authorizations for data access. Another important advantage is that process designers can interleave the security aspect with the execution context of a process. To illustrate, one can specify that an obligation must start after a particular task, and the system can take contextual information into account (e.g., the affiliation of a particular house owner) when executing the obligation. *BTG!* accounts for an important preference of process designers, namely to model constraints declaratively [22].

Challenges: Configuring and performing the usability study envisioned for *BTG!* is not trivial. This is because there do not exist any publications sharing experiences on how to perform a study for hybrid process specification languages. It also is challenging to select appropriate usability methods among the numerous ones that are well-known [15], in order to cover a broad set of relevant usability aspects of the language.

Contributions: This paper makes the following contributions:

Configuration of a usability study: We describe the design of a usability study for *BTG!* we have composed from well-known usability methods. Our study focuses on the usability criteria learnability, efficiency, errors, and satisfaction. We explain and justify our design decisions as well as the evaluation criteria we

have chosen. We do not compare *BTG!* with any competitors, mainly because we are not aware of any. By learning how to identify and to eliminate prominent usability problems in our specific case, the hope is to arrive at insights that are valuable in similar contexts as well.

Realization of the study: We have performed a usability study for *BTG!*, which consists of three workshop iterations (phases), with 13 participants. As the main goal has been to detect problems around *BTG!* itself, we perform a paper and pencil study. We have identified problems many participants had encountered and then have adapted the language as well as our teaching material accordingly. We describe some of these modifications and how they have improved the usability of the language.

Lessons learned: We describe our experiences on performing such a study (e.g., feedback of participants issued during tutorials and moderated discussions).

Paper overview: Section 2 reviews the annotation language. Section 3 describes our usability study. Section 4 discusses related work. Section 5 concludes.

2 The BTG! Annotation Language

We first describe the general concept of the annotation language for process models. We then introduce the language constructs by means of examples.

2.1 Interleaving Security and Application Functionality

The rationale behind *BTG!* is as follows: Application-independent process fragments, which are available in a repository, carry out security-relevant BTG tasks. Such a fragment reflects the control flow of BTG tasks (e.g., to ask for BTG, to agree, to access data). Now think of an application designer who wants to extend his/her application processes with such BTG functionality. To do so, in a nutshell, he/she comes up with parameter values that specify the embedding of the fragments into the application process and their configuration, such as who is authorized to ask for BTG, or which conditions must hold to enable BTG. In other words, this specification is declarative. See [20] for more details.

2.2 Example Annotation Language

We now summarize the concepts of *BTG!* by means of examples.

Example Process. Think of a process for final exams at universities. There are three roles: lecturer, student, and clerk of a service unit. First, a student takes an examination, and a lecturer assesses the performance of the students. If the student passes the exam, the lecturer confirms the exam result (task *confirmation*) and sends it to the service unit. Otherwise, the student has to

repeat the exam. The clerk in turn generates a certificate and sends it to the student. A student might need exceptional access to his exam results right after the lecturer has confirmed that the exam has been passed, to start the job search immediately. We now show how using *BTG!* can support "Breaking the Glass" in this context.

Core Language Features. A BTG annotation refers to a process task. This means that BTG functionality, i.e., a given process fragment, is supposed to take place after the execution of this task. An annnotation starts with the keyword ≪`BTG:` and ends with ≫. It contains expressions from a predefined vocabulary of the form `parameter = "value"`. *BTG!* has mandatory and optional expressions. Process designers must specify which rights `rights="$right-types"` the system provides for which data objects `objects="$objectnames"` for the exceptional case. Symbol $ denotes a variable. In its simplest variant, a BTG description, in the form of an annotation of a process task, is as follows; this one means that one has the right to read the assessment of an exam:

```
<<BTG: objects= "exam-assessment"
       rights= "read">>
```

Role Holders for BTG Tasks. When the system carries out BTG functionality, certain process participants perform security-relevant tasks in addition to the application tasks. To give authorizations for those tasks (e.g., asking for exceptional access, allowing for access, performing obligations to repair the glass), *BTG!* allows the specification of security roles, namely *BTG Accessor*, *BTG Activator*, and *Compensator*. In our example, a student as holder of role *BTG Accessor* should have exceptional access to the data and thus have the authorization to ask for BTG. In the introductory example, this has been the fire man. The lecturer as *BTG Activator* has to agree and hence must be authorized to do this activation task. This has been the supervisor of the fire man in the introduction. A *Compensator* (the lecturer) is responsible for "repairing the broken glass" by informing the clerk by email. In the example, a process designer would assign process roles to the security roles as follows:

```
<<BTG: ...  BTGAccessorRole= "student"
            BTGActivatorRole= "lecturer" >>
```

BTG! foresees default settings for the role specifications as follows: If there is no specification for `BTGAccessor`, role holders of the BTG-annotated activity get the rights for exceptional access. If `BTGAccessor` is specified but `BTGActivator` is not, `BTGAccessor` also is in charge of activating (i.e., deciding on) the exceptional access.

Authentication Requirements for Role Holders. *BTG!* facilitates the specification of authentication requirements for security-critical roles. For instance, the expression `AuthnBTGActivator-attr= "$avps, idp=$idp-address"` specifies that authentication is required for a *BTG Activator*. The parameter

`AuthnBTGActivator-attr` specifies information the system uses to authenticate the holder of role *BTG Activator*. One can specify $avps, i.e., a list of `($attribute,$value)`- pairs and, optionally, an Identity Provider (IdP). An attribute is a property used for this authentication. A value specifies an identifier for this attribute. For *BTG Accessor* and *Compensator*, the syntax is accordingly.

In our scenario, the student has to identify himself with the attribute matriculation number (mn), with the value for s-card-mn given on his student ID card. This leads to:

```
<<BTG: ... BTGAccessorRole= "student"
       AuthnBTGAccessor-attr="(mn,s-card-mn),idp=some.univ.edu">>
```

Conditions on the Process Execution Context. In the example so far, a BTG annotation means that the process facilitates BTG functionality unconditionally right after the annotated process element. To specify other cases, *BTG!* allows to formulate conditions for BTG by taking the functional, behavioral, organisational, operational, and data aspect of process instances into account (c.f. [21]). These conditions are evaluated before an exceptional access is granted.

To illustrate, an exceptional access in our example scenario can only be granted if the task *confirmation* has been executed, and the person who has confirmed the grade is a professor. This is expressed as follows:

```
<<BTG: ...  exec= "performer('confirmation') == professor"
            start= "executed('confirmation')" ... >>
```

Process designers can formulate conditions on the context by specifying expressions using the parameters `start` and `exec`. The intention is to represent *when* conditions on the context must hold for BTG to be enabled. The parameter `exec` specifies conditions for allowing BTG (i.e., executing the fragment) that must hold right after the annotated process element. In contrast, the parameter `start` allows to represent that the application allows BTG as soon as the condition is fulfilled. Thus, `start` typically specifies a condition which will be fulfilled during process execution, and which implies a delay for facilitating BTG. Regarding the realization, the choice of these parameters determines the way how the Business Process Management System (BPMS) embeds BTG functionality into the application process, namely in parallel to the process (with `start`) or in sequence (with `exec`).

We refer to conditions such as `"performer('confirmation')== professor"` and `"executed('confirmation')"` in `start` and `exec` expressions as business-process-context constraint *BP-CC*. To support formulating these, *BTG!* provides functions such as `owner`, `executed`, `performer`, `data-access`, `start-time-access`, with specific arguments. These functions enable the specification of associated entities, tasks, and data objects to be used for the representation of temporal and causal constraints. Formally, a BP-CC is as follows:

```
BP-CC:: function | function s-Op value | BP-CC c-Op BP-CC
```

Here, `s-Op` and `c-Op` are operators provided by the language: `s-Op` are simple ones, e.g., set operators, and `c-Op` are complex ones, e.g., boolean operators. A BP-CC returns a boolean value.

Obligations. *BTG!* enables the specification of one or more obligation to repair the glass. The expression `Obligations="$obligation-IDs"` is an assignment of a list of obligation IDs, specifying process fragments to be executed. In the university scenario, a lecturer as *Compensator* has to send an e-mail to the clerk of the service unit after an exceptional data access. To express this, a process designer first has to label the obligation as part of the BTG annotation. Next, he/she must specify the obligation parameters in a separate annotation:

```
<<BTG: ... Obligations= "O1">>
<< Obligation:ID= "O1"
    OGCompensator= "Lecturer"
    pattern= "Send Email"
    OGParameters= "((From, Lecturer),(To, Service Unit),
    (Subject, Degree),(Body, has been confirmed))"       >>
```

Note that the execution of an obligation does not necessarily have to take place right after the glass has been broken.

3 Analyzing the Usability of BTG!

Our goal has been to find and to eliminate usability problems (i.e., pitfalls) of *BTG!*. In general, there exist two variants of usability testing: comparing against objectives and against alternatives. In line with our goal, we follow the first option. This also is common practice in usability studies for software [15]. In the following, we first review requirements deduced from classical usability criteria, and we describe the configuration of our usability study and the test methods we have selected. Next, we describe the realization of the study, lessons learned and open issues.

3.1 Overview

Requirements: We follow the approved way to quantify usability by taking the classical usability criteria into account, namely *learnability*, *efficiency*, *memorability*, *errors*, and *satisfaction* [15]. To do so, we first interpret these criteria with respect to *BTG!*, which we list as requirements in the following.

R1: Learnability. In our context, learnability means that process designers should be able to learn *BTG!* with little time effort, and they should be satisfied with their learning process. After some hours of intensive learning, one should be able to annotate BTG cases, possibly by using assisting material.

R2: Efficiency. BTG! is efficient if process designers can use *BTG!* very quickly after they have been trained using it.

R3: Errors. Users should make only few errors when using *BTG!*, and users should not make critical errors at all. Syntactical errors are less critical than semantic ones. This is because one might avoid them by using an appropriate tool.

R4: Satisfaction. Process designers should feel comfortable and confident and should be satisfied by using *BTG!*.

R5: Memorability. Participants should be able to recapitulate their learned knowledge quickly and to employ at least simple language concepts after not having used *BTG!* for a while. A good time for tests on memorability is when its usability regarding the other criteria is high. This study aims at solving these problems first. In consequence, respective tests are future work and should be performed between three and six months later.

Settings. We follow text-book knowledge on usability engineering, as follows.

Tutorial material. We have developed tutorial slides to train *BTG!*. We were aware of the fact that the quality of the material as well as the way of presenting it affect learnability. An alternative to a tutorial would have been to ask participants to study publications on *BTG!*. However, scientific publications are rarely written didactically. Further, it is difficult to monitor the learning process of participants who are reading. – We also prepared a two-page user manual summarizing the syntax of *BTG!*.

Feedback. Our study consists of three phases, and in each phase we have evaluated and improved *BTG!* and adapted the training material. However, we did not let users decide on modifications of *BTG!*, because of its complexity. Further, it is often difficult to find a consent among users' opinions, cf. [15].

Low costs. We aimed at an economic run of the study, i.e., having a good tradeoff of costs versus accuracy. In line with the literature [15], we work with relatively few participants (13 in total). An alternative would have been to run empirical studies, but at significantly higher costs. To keep costs low, we also decided against hiring external process designers.

Profile of participants. We selected participants with knowledge in process modelling, who ideally had some experience in using security annotations for process models. Literature also suggests to have heterogeneous sets of participants for the different test iterations. We follow this recommendation.

Paper and pencil tests. We decided for a paper and pencil test, to focus on *BTG!* and to eliminate effects due to the quality of tools used.

Test Methods Selected. Established usability techniques are *asking* participants (e.g., interviews, questionnaires) or *observing* them when doing certain tasks (e.g., by measuring the performance and error rates). We now say how we have addressed the usability criteria.

Asking Particpiants. To measure the *user satisfaction*, participants had to fill out a questionnaire. Many sophisticated questionnaires for evaluating user satisfaction exist. Among these tools, we decided for the System Usability Scale (SUS) [1] questionnaire because it is widely used, its validity and reliability have been demonstrated, and it is relatively short (ten item questions). We adapted a German version [17] of the SUS questionnaire to our needs. Regarding *interviews*, we decided not to interview participants explicitly. One goal was to support an atmosphere in which participants did not feel observed. We however had *moderated discussions* at the end of a workshop.

Observation. We collected the written solutions by participants for their tasks. These solutions is the most relevant data we have collected for the analysis. By giving tutorials, we were able to actively observe the *learnability* of *BTG!*. Our tutorial consists of several topics of *BTG!* (e.g., motivation, specification of BTG roles, obligations, etc.). Participants had to solve small tasks during the tutorial, so-called training tasks. We had doubts whether users are able to learn *BTG!* in short time. Thus, we measured the training time and analyzed errors that had occurred when doing the tasks. After the tutorial, we asked participants to solve complex annotation tasks (so-called expert tasks) for various scenarios, to study the *efficiency*. Indicators for low efficiency are, for example, unnecessarily complex language constructs. We also wanted to learn which constructs participants do not use, although they would reduce the annotation effort. Further, we tried to identify reasons for language problems from questions issued. We analyzed the *errors* for both the training as well as the expert tasks. This is to detect pitfalls for learning as well as for using *BTG!*. We have protocolled participant questions and comments issued during the tutorial, documented their learning behavior, and the atmosphere during the workshops. Thus, we explicitly developed a presence tutorial, and not an online-based one. We decided against any recording of the workshops, to keep the environment as natural as possible.

As our focus was to make improvements, we did not measure usability criteria (on a scale). Instead, we decided for a macro-analysis of observed data, i.e., address usability issues for problem categories. By doing so, we were able to consider relationships between usability issues rather than evaluating them in isolation.

3.2 Design and Preparation of Workshops

The core of the study have been three usability test phases, each consisting of the steps preparation, running of workshop, and evaluation and improvement of language and learning material. A workshop consists of a tutorial for *BTG!*, followed by expert tasks, filling out questionnaires, and a moderated final discussion.

Development of Tutorial Material. The tutorial features the University scenario from Section 2.2 with more examples for the language constructs. We have striven to activate participants by letting them solve *training tasks* for each concept. We further have presented a solution for each training task and have motivated participants to ask questions, if any. We have scheduled the tutorial for 1.5 hours with around 80 slides [10].

Expert Tasks. To test how trained users perform with the language, we have developed *expert tasks* for three realistic scenarios [10]. To this end, we have provided process models to the participants. These scenarios feature exceptional situations with specific security needs where BTG annotations are convenient for the specification, but are no emergency cases. The first scenario (S-ET-1)

aims at buying and issuing a preliminary season ticket for public transport. In this context, a sales agent requires exceptional access to customer data to issue a ticket. The second scenario (S-ET-2) is to request a book from a university library which is already borrowed by someone else. Iff the loaner of the book agrees, the library provides the requester of the book access to the contact data of the loaner. The third one (S-ET-3) deals with a loan request at a bank, and features BTG iff nobody has decided on the loan request within a time period. By sketching the annotation tasks for these scenarios we aimed to cover all features of the BTG language. To work on these expert tasks, we scheduled around 45 minutes altogether.

Classification of Errors. In line with literature, we have classified usability issues into five categories: Category 1 refers to minor syntactic problems, issues of Category 2 ask for syntactic or semantic clarifications. Category 3 refers to major syntactic and semantic problems, and Category 4 contains conceptual misunderstandings of the language which might necessitate major revisions. Category 5 is the wish list of participants regarding additional features.

Questionnaire. With the questionnaire we have asked for user satisfaction regarding the BTG annotation language [10]. To identify pitfalls and dislikes for particular language constructs, we have added four questions asking for the most severe difficulties, feature wishes, proposals for improvements, etc. Participants were encouraged to write free text answers. We have set aside 15 minutes to complete the questionnaire.

Selecting and Incentivizing Participants. We have decided to acquire end users, i.e., process designers, at our institution which match our user profile, instead of hiring external process designers. The rationale has been that external experts may have only little motivation to participate, the costs for their participation are high, and there are few incentives for doing good work. We looked for participants with solid computer science skills, who are familiar with process modelling techniques, and are able to abstract. We do not explicitly ask for specific programming skills, and participants must not know *BTG!*.

We now say how we have selected participants for the study.

Phase 1: Four computer scientists of our research group have participated in the tutorial. All participants have met our profile, and no particular incentives were necessary. We have scheduled this group for Phase 1 because we expected fruitful discussions from it.

Phase 2: For the second phase, we have recruited six computer science or information systems students, being in the last year of their master program. Related work reports that such individuals have been good test persons [9],[8]. To ensure that participants have the required knowledge, we have selected only students who had already passed their exam on Business Process Management at our university. Some of them had some knowledge on the concepts of a generic security annotation language, but *BTG!* was new for all of them. We have paid them – according to regulations for student payment at our university – 10 Euro

per hour for participating in the workshop, which was scheduled for three hours altogether. We have announced a bonus of 10 Euro for the participant issuing the most constructive feedback.

Phase 3: Three experts on business processes have participated in this phase. All of them were familiar with the Business Process Model and Notation (BPMN) language and some security annotation languages, but they were not or only marginally familiar with *BTG!*. We have scheduled this expert group for the last phase, because these persons might identify problems which non-experts do not see.

Organization of Workshops. We did not have any particular needs regarding technical equipment, such as usability labs. To have a comfortable atmosphere we have provided some drinks and snacks for the participants.

3.3 Study Results

Behavior Rules for Workshops. We have aimed for a good atmosphere so that participants did not feel under pressure. Participants were allowed to ask questions any time, and we have taken notes on questions raised. We have told participants that there are no right or wrong answers. We also have said that their solutions might be valuable information for us.

We now describe the realization of the three phases.

Phase 1. All in all, we have identified many usability issues in this phase. Table 1 lists these issues. We now illustrate only some problems exemplarily. After having detected the issues, we have modified the language accordingly.

Syntax: In general, we observed some uncertainty when using the syntax. Some participants stated that it was too complicated (e.g., many quotation marks) or confusing (e.g., listing several items as a parameter). Based on this feedback we simplified the syntax.

Semantics: To illustrate some usability issues at this level, we briefly describe the expert task for the 'University library' scenario (S-ET-2) first. For this scenario, participants had to specify BTG to give the requester of a book exceptional access to contact data of the current loaner of the book. The loaner has to agree to this access. This asks for assigning the BTG roles `Accessor` and `Activator` to requesters and loaners, for specifying the object to be accessed (the contact data of the loaner) and the access right for reading them. We further required the requester to authenticate himself (i.e., authentication for `Accessor`) by means of a student registration number (srn), using the University Identity Provider (IdP) 'idp.some-univ.edu'. Further, we asked for an obligation to inform the head of the library.

The following annotation would have been a correct solution:

```
<<BTG:objects= "loaner-data"
rights= "read"
BTGAccessor= "requester"
AuthnBTGAccessor-attr= "(srn, requ.srn),idp=idp.some-univ.edu"
BTGActivator= "loaner"
Obligations= "O1">>
```

Phase	Usability Issue	Category
1	Syntax for representation of lists (in part. authentication)	1
1	Syntax for variables	1
1	Order of specifications in annotation terms	1
1	Nonambiguous specification of rights	2
1	Default values for BTG roles	2
1	Usage of authentication parameters	2
1	(Assignment of) Compensator Role	3
1	(Assignment of) BTG Roles (Accessor, Activator)	3
1	Semantics of 'start' and 'exec' constraints	3
1	Semantics of authentication parameters	3
1	Usage of BP-CC	4
1	Declarative vs. functional notations for BTG and BP-CC	4
1	Possibility to use BTG role names in BP-CC	5
1	Supporting multiple Activators, Accessors and Compensators	5
1	More powerful rights (e.g., create and delete)	5
1	Specification of relationships between Obligations	5
1	Additional functions for BP-CC	5
1	Supporting BTG for exceptional performing of tasks	5

Table 1: Categorized Usability Issues of Phase 1

Authentication and Authorization: Participants felt uncomfortable when describing authentications. They were not familiar with attribute-value pairs an IdP asks for. Some were confused by separating authentication and authorization specifications for one role, see expressions for `AuthnBTGAccessor-attr` and for `BTGAccessor`. Participants also found complex authorization specifications (`AuthnBTGAccessor-attr="(srn,requ.srn),idp=idp.some-univ.edu"` in the example) too complicated. We reacted to these issues as follows: First, we improved our tutorial, to better explain how to use IdPs. Second, we modified the parameter names `BTGAccessor` and `AuthnBTGAccessor-attr` to `accessor.role` and `accessor.authn` respectively. We have chosen these parameter names because we explicitly decided against using a compact, more deeply structured expression for both authorization and authentication in combination. Third, we simplified the syntactical representation for authentication parameters, resulting in `accessor.authn:(srn, card, idp.some-univ.edu)` for Scenario S-ET-2.

Business Process Context Constraints: We observed usability problems when specifying BP-CC. Participants expressed their dissatisfaction with the mixture of declarative and functional aspects in this context. To recapitulate, the language requires an expression `start= $BP-CC` or `exec= $BP-CC`. The `$BP-CC` in turn requires pre-defined functions, e.g., `performer('$task-name')`. We now illustrate this language aspect by means of the specification we asked for in the scenario 'Purchasing a season ticket' (S-ET-1): BTG is only allowed if the customer is an adult, and the purchase date must be within the first five days of the month. Next to others, the process model for this scenario contains the tasks *buy season ticket* (performed by a customer) and *prepare ticket contract* (performed by an agent).

The following specification would have been a correct solution:

```
<<BTG: objects= "customer-data"...
       exec= "start-time('prepare ticket contract').day <= 5
       ∧ performer('buy season ticket').age >= 18">>
```

To account for participant dislikes for combinations of declarative and functional aspects, we have evaluated several options to make the language constructs unique. Finally, we have decided against fundamentally changing the conceptual setup of BP-CC. The reason has been that both ways have their advantages and disadvantages: Declarative assignments are intuitive and easy to learn, and functional representations for BP-CC can easily handle many arguments. The later is important to identify process elements in iterations, e.g., loops. Declarative representations in turn would lead to specifications that are lengthy.

Another common pitfall for participants was to use role names for BP-CC specifications, but the language does not foresee this. *BTG!* allows only functions specifying role holders of process instances (e.g., `performer('$task-name')`. We have emphasized this in later tutorials.

Next, the names of the parameters `start` and `exec` were misleading: Participants associated with the name `start` a BP-CC with temporal conditions, e.g., within ten days. But the intended meaning is that `start`-expressions may be a delay for allowing BTG due to conditions that hold later. Accordingly, participants associated the parameter `exec` with conditional constraints. Because of this, participants were unaware of how both parameters imply the embedding of BTG fragments into the process model. Consequently, we replaced the parameter name `start` by `cond.anytime`, i.e., the system starts BTG as soon as the condition holds. We also renamed the parameter `exec` to `cond.immediate`, i.e., the condition must hold immediately after execution of the annotated process element. Further, we provided more details in the tutorial.

Desired features: We also reviewed the wish list of participants (see Category 5 in Table 1). In general, participants asked for new language features, such as providing BTG not only for accessing data, but also for exceptional execution of tasks. Another desired feature was to allow several roles to be assigned to a BTG role. But at this time, we decided against supporting any of these features, because we aimed at addressing usability problems first. We were aware of the fact that new features would make the language even more complex.

Due to intensive discussions, this workshop had taken longer than scheduled. All in all, we have analyzed any usability problem we have detected. In consequence, we have modified *BTG!* significantly by simplifying the syntax as well as the semantics.

Phase 2. During the tutorial run, we had already gained the impression that the usability of *BTG!* is better, compared to Phase 1. This is because participants discussed issues less intensively, and they have asked fewer questions regarding the training tasks. When analyzing solutions, we observed a significant improvement over Phase 1. Table 2 shows the issues detected and their corresponding categories. Now many problems addressed in Phase 1 either were

not relevant any more at all, or participants had only little problems. Due to the lower error rate, we have put them into a lower problem category (e.g., usage of BP-CC, semantics of authentication parameters). The improvement shows that our modifications had been appropriate. Interestingly, participants have not been confused by using declarative and functional assignments any more, although we had not modified this characteristic of the language, only the respective syntax. Again, participants proposed new language features (see Category 5 in Table 2), a subset of the ones from Phase 1.

Phase	Usability Issue	Category
2	Semantics of authentication	2
2	Default values for rights	2
2	Assignment of BTG roles	2
2	Usage of BP-CC	3
2	Specification of relationships between Obligations	5
2	Supporting multiple Activators, Accessors and Compensators	5

Table 2: Categorized Usability Issues of Phase 2

Open usability issues and language modifications. We identified few usability problems in this phase. The most relevant ones were that participants still were slightly unconfident when using BP-CCs, and one participant was uncertain in assigning BTG roles appropriately. Further, the semantics of authentication was slightly unclear, and participants were not always familiar with the default settings for rights. Consequently, we did some minor changes on the language and the tutorial, addressing exactly these issues.

Desired features. Participants proposed new language features similar to the ones from Phase 1. For instance, they asked to allow several roles to be assigned to a BTG role. To illustrate, both loaners and heads of library might take over the role of an *BTGActivator* in the library scenario. In line with this, we changed the syntax of assigning process roles to the accessor to: `accessor.role: list($rolename)`. Accordingly, the other role assignments correspond to this syntax. We delayed the other desired features for future work. This is because their realization would require further investigation of the language and regarding its usability.

Phase 3. As the participants in this phase have been process modelling experts, we have run the tutorial slightly faster. First of all, many usability problems identified in Phase 1 and Phase 2 were not relevant any more. Participants performed very well in doing the tasks, and participants were generally satisfied with the language. To illustrate, they always used the parameters `cond.immediate` and `cond.anytime` correctly. Their few critical comments referred to some details regarding the specification of BP-CC. Table 3 shows the results from Phase 3. We now exemplarily explain two points in more detail.

Phase	Usability Issue	Category
3	Weak points of some BP-CC functions	2
3	Operator usage for BP-CC	2
3	Need for using functions in BP-CC unclear	3
3	Supporting conditions and relationships for BTG Role Assignments	5

Table 3: Categorized Usability Issuess of Phase 3

Need for functions in BP-CC. Regarding usability, our expert users have not been concerned that one must use pre-defined functions to specify performers in BP-CC. Indeed, using pre-defined functions for BP-CC is more verbose, but this restriction is on purpose, to distinguish performers of process instances and roles. To illustrate, the term `performer('$task-name')` with a pre-defined function stands for a distinct performer at process execution, while specifying only roles (e.g., `student`) might be ambiguous. This is because at process execution, there might be several performers of a particular role, student in our example. But it had slightly confused the participants that the language allows to use process roles when specifying a data `object`, but does not allow this for BP-CC (see Section 2.2). We decided to explain that there is no ambiguity for objects in the later case in future tutorials.

Desired features. Participants suggested new features. They proposed specifying conditions for BTG Role Assignments, e.g., to constrain the assignment if the *BTG Accessor* is an adult or not. They also proposed allowing to specify relationships between role assignments, such as separation of duties for *BTGAccessor* and *BTG Activator*.

User Satisfaction. We have already discussed implicit factors of user satisfaction, emerging from comments participants have issued. We now focus on results obtained by measuring satisfaction explicitly, using the SUS questionnaire [10]. Although - due to the small number of participants - the results do not give empirically solid insight, they do support some of our modifications of the language.

To analyze the results, we first have transformed the natural language scale of the questionnaire into numbers. For every question, 5.0 is optimal regarding usability, and 3.0 is the value in the middle of the scale. Table 4 lists the results for all phases (Q1: regular usage of *BTG!* likely, Q2: language too complex, Q3: convenience of using language, Q4: further support required, Q5: *BTG!* has required functionality, Q6: too many inconsistencies, Q7: easy to learn for process modellers, Q8: working with language too laborious, Q9: firm in it, Q10: learning effort).

Due to the heterogeneity of the participants in the three phases, results have to be taken with care. Most values measured in Phase 1 were slightly above the middle of the scale. We concluded that *BTG!* generally satisfies users but still needs improvements. While the low learning effort (Q10: 4.0) obtained by users

Phase	Average	Q1	Q2	Q3	Q4	Q5	Q6	Q7	Q8	Q9	Q10
1	3.18	3.8	3.0	3.3	3.0	2.8	2.5	3.8	3.3	2.5	4.0
2	3.75	4.0	4.2	3.7	2.7	3.2	4.0	4.3	4.2	3.2	4.0
3	3.73	3.7	3.3	3.3	4.3	4.3	3.0	4.3	3.3	3.7	4.0

Table 4: User Satisfaction of Phase 1, Phase 2, and Phase 3

is remarkable, inconsistencies (Q6: 2.5) and uncertainty in use (Q9: 2.5) unfolded problems we have tried to specifically address with our modifications.

As we had modified *BTG!* significantly after Phase 1, many ratings are notably higher in Phase 2, indicating that the changes have improved the usability. The remarkably low complexity (Q2: 4.2) as well as simple usage of the language (Q8: 4.2) and the, once again, good learnability (Q10: 4.0) point towards a good general usability. However the results also showed some uncertainty (low values for Q4 and Q9). Thus our main focus after Phase 2 has been more clarity.

In Phase 3, the average of the values is approximately the same as for Phase 2. Further, we get a result of 4.3 on user satisfaction regarding the functionality (Q5). This is significantly higher than the values for Phase 1 (2.8) and for Phase 2 (3.2).

By comparing the satisfaction of Phase 1 with the average values of Phase 2 and Phase 3, we see that the language adaptations after Phase 1 have increased the overall satisfaction. The value for learnability has always been high. This confirms our design decision to run a presence tutorial with a step-wise training of BTG concepts, as well as to activate participants to perform training tasks during the tutorial.

Summary. By having organized a first workshop, we have identified various usability problems. Our modifications of *BTG!* however have led to significantly fewer problems. The positive feedback from of the last workshop with exerienced participants has shown that the usability of *BTG!* is good. In general, the concept of BP-CC have lead to the most usability problems in any phase. This has not been surprising after all, because this concept is complex, and *BTG!* provides a comprehensive vocabulary for functions.

All these points hold in the light of usability engineering, namely that a usability study always yields a snapshot of (selected) users for particular tasks for the configuration chosen [15].

3.4 Lessons Learned

We summarize some of our lessons learned when performing a usability study in the process-design context, and for improving a process-specification language which interleaves several aspects.

Performing a Usability Study. *Recruiting students as participants.* When planning the study, we were unsure about hiring students as participants. One reason for this concern was that our institution does restrict the amount of payment, making a participation financially less attractive than jobs in industry. However, all invited students agreed to participate. Secondly, we were aware that students might cancel their participation. Colleagues had a drop out rate of 15 to 20 percent in similar setting. This means that we had to calculate accordingly. Although students' participation was not obligatory for their studies (e.g., to get a bonus), all students were extremely reliable. We think that this is because all students were known to our research group and were interested in the topic. Thus, we propose to recruit students from carefully selected groups, as opposed to announcing such job offers widely and openly.

Collecting opinions. In our setting, we have observed an open discussion culture to be very productive in the first phase. However, these participants were not very motivated to fill out the questionnaire after long discussions. On the other hand, student participants did not discuss issues as openly as computer scientists of our institution. Thus, in such constellations we for our part have benefitted from using questionnaires. Consequently, the way how opinions are collected should be adapted to the target group with care.

Evaluation. As opposed to similar studies referred to in the section on related work which mainly were quantitative in nature, our study was a qualitative one. Both variants have been in a way useful in the past. We for our part have been able to make significant improvements and to detect many issues with the qualitative variant.

Few participants. In our constellation, few participants were sufficient to improve usability significantly. Interestingly, we were able to eliminate most problems in Phase 1. Consequently, we propose to put effort into the training material and the profiling of participants, as opposed to recruiting large numbers of them.

Heterogeneous participants. The first group made many helpful comments regarding general issues. This confirms our rationale to start with participants which are used to discuss issues regularly. The second group in turn gave good feedback on the modifications of the language after Phase 1. The expert group has been very helpful to detect specific problems. Consequently, we recommend to work with heterogeneous participants in similar studies.

Tutorial. One experience has been that a well-thought-out tutorial is very helpful to detect usability problems of a hybrid process annotation language. Some participants have even commented on the good, step-wise way to train the concepts of *BTG!* explicitly. We for our part conclude that it has been worth the effort for preparing the material. However, we also are aware of dependencies between quality of training material and learnability of *BTG!*.

Increased Usability. *Language re-design.* Important modifications of the language have been changes in its syntax and in the wording for expressions. We have not made any conceptual changes of the language. For instance, omitting a BTG role has been an option. By observing and reacting to the problems issued,

our well-reflected modifications of the syntax have increased the usability of the language significantly. This shows that syntactical issues should receive much attention when designing a hybrid language.

Proposals by Participants. Participants had many good ideas for new language features. Mainly because any new feature affects usability, we realized only few of them. To illustrate, supporting BTG for task execution would imply major extensions of *BTG!*. On the other hand, participants were satisfied with the new features we had introduced based on earlier suggestions. Here, the challenge is to find a good trade-off between feasability and more powerful features. We conclude for ourselves that suggestions for new features should be taken seriously, but without leaving aside potential consequences.

4 Related Work

A lot of related work aims at the design of graphical notations, or compares modelling paradigms, languages, or notations. Our intention in turn has been a stepwise improvement of the usability of an existing language. Next, while we address several usability criteria, related studies focus on specific usability issues, or they rely on specific restricted test methods, such as questionnaires [18]. Nevertheless, the experiences reported there have affected our study. We now describe these approaches, grouping them by their intension and goals.

Usability studies for design. [12] is a user study aiming to find appropriate icons/drawings representing security concepts for process modeling. Our focus is different, namely the usability of a language, not its visualization.

Usability studies for comparisons. To our knowledge, studies on the usability of process modelling languages and their modifications do not choose a non-empirical approach for improving a language, as we have done. Instead, they either compare realizations or focus on specific hypotheses with empirical results. [4] compares iterative and declarative modelling languages, but without usability tests. The respective study is [16], where the authors present hypotheses on strengths and weaknesses of imperative and declarative languages. In contrast to us, their setting is offline, i.e., unsupervised, and they use the results to validate hypotheses. Other research has focused on finding good labels for activities, a challenge similar to choosing the terminology for *BTG!*. [14] discusses activity labelling and supporting the understandability of process models with good icons. [13] is a usability study with a focus on textual labels. These studies focus on establishing guidelines for labelling, but do not actually improve existing ones.

Other work on the usability of BPMN introduces more intuitive icons and a simplified version of BPMN, e.g., there is only one gateway type, called SBPMN [5]. In their workshops the authors first introduced SBPMN, to enable participants to solve tasks with it. The motivation has not been to improve SBPMN but to collect evidence that "SBPMN is better than BPMN". [8] is another study on comparing notations of modelling languages (BPMN, EPC, UML, YAWL). [7] has investigated routing symbols in these languages. Both studies use a setup

similar to ours, including a tutorial and expert tasks. Their results were exclusively used for comparisons and conclusions on which notations or routing symbols serve which purpose best.

[7] compares the usability of different modeling languages. Like in our work, the authors define quality criteria for modelling languages, and how these criteria have to be applied. They do not conduct any user studies, nor are we aware of any study using their approach. Just like we have done, [19] consists of tutorial-based workshops with questionnaires and moderated discussions. They wanted to find out what professionals think about declarative process modelling.

5 Conclusions

A real-world process requires a comprehensive specification that interweaves several process aspects, such as the data and organizational one. Hybrid modelling languages support this, but their usage is significantly more involved than the one of using conventional languages which focus on only few aspects. This asks for usability studies aiming to do away or at least alleviate language issues. This paper has described the design and realization of a study for a hybrid language *BTG!* and has shown how we have improved its usability.

References

1. J. Brooke. SUS: A quick and dirty usability scale. `http://www.tbistafftraining.info/smartphones/documents/b5_during_the_trial_usability_scale_v1_09aug11.pdf`, 1996. accessed Dec. 2014.
2. A. D. Brucker and H. Petritsch. Extending Access Control Models with Break-glass. In *SACMAT*, 2009.
3. M. Dumas, M. L. Rosa, J. Mendling, and H. A. Reijers. *Fundamentals of Business Process Management*. Springer, 2013.
4. D. Fahland, D. Lübke, J. Mendling, H. Reijers, B. Weber, M. Weidlich, and S. Zugal. Declarative versus Imperative Process Modeling Languages: The Issue of Understandability. *Enterprise, Business-Process and Information Systems Modeling*, 29, 2009.
5. H. Fernández-Fernández, E. Palacios-González, V. Garcia-Diaz, B. C. P. Garcia-Bustelo, O. S. Martinez, and J. M. C. Lovelle. SBPMN - an easier business process modeling notation for business users. *Computer Standards & Interfaces*, 32, 2010.
6. A. Ferreira. *Modelling Access Control for Healthcare Information Systems: How to control access through policies, human processes and legislation.* PhD thesis, October 2010.
7. K. Figl, J. Mendling, and M. Strembeck. Towards a Usability Assessment of Process Modeling Languages. volume 554 of *EPK Workshop*, Nov. 2009.
8. K. Figl, J. Mendling, and M. Strembeck. The Influence of Notational Deficiencies on Process Model Comprehension. *J. AIS*, 14(6), 2013.
9. K. Figl, J. Recker, and J. Mendling. A study on the effects of routing symbol design on process model comprehension. *Decision Support Systems Journal*, 54(2), 2013.

10. S. Grabatin. Documents BTG Usability Study. http://dbis.ipd.kit.edu/2134.php. accessed Dec. 2014.
11. M. Leitner and S. Rinderle-Ma. A systematic review on security in Process-Aware Information Systems - Constitution, challenges, and future directions. *Information and Software Technology*, 56, March 2014.
12. M. Leitner, S. Schefer-Wenzl, S. Rinderle-Ma, and M. Strembeck. An Experimental Study on the Design and Modeling of Security Concepts in Business Processes. In *PoEM*, 2013.
13. J. Mendling, J. Recker, and H. A. Reijers. Activity labeling in process modeling: Empirical insights and recommendations. *Information Systems*, 35, 2010.
14. J. Mendling, J. Recker, and H. A. Reijers. On the Usage of Labels and Icons in Business Process Modeling. *IJISMD*, 1(2), 2010.
15. J. Nielsen. *Usability Engineering*. Morgan Kaufmann, 1994.
16. P. Pichler, B. Weber, S. Zugal, J. Pinggera, J. Mendling, and H. A. Reijers. Imperative versus Declarative Process Modeling Languages: An Empirical Investigation. In *BPM Workshops*. Springer, 2012.
17. M. Rauer. Quantitative Usablility-Analysen mit der System Usability Scale (SUS). `http://blog.seibert-media.net/2011/04/11/usablility-analysen-system-usability-scale-sus/`, april 2011. accessed Dec. 2014.
18. J. Recker. Continued use of process modeling grammars: the impact of individual difference factors. *EJIS*, 19(1), 2010.
19. H. A. Reijers, T. Slaats, and C. Stahl. Declarative Modeling-An Academic Dream or the Future for BPM? In *Business Process Management*, 2013.
20. S. von Stackelberg, K. Böhm, and M. Bracht. Embedding 'Break the Glass' into Business Process Models. Technical Report Faculity of Computer Science, KIT 2011,38, December 2011.
21. S. von Stackelberg, K. Böhm, and M. Bracht. Embedding 'Break the Glass' into Business Process Models. In *OTM Conferences (1)*, 2012.
22. B. Weber, H. A. Reijers, S. Zugal, and W. Wild. The Declarative Approach to Business Process Execution: An Empirical Test. In *CAiSE*, 2009.