Abstract

The need for flexibility in business process languages and tools has evolved over the past few decades, from totally rigid approaches, to totally flexible ones. The need to allow process designers to control this flexibility has risen due to the fact that, in the everyday practice, people do not wish for total flexibility. They rather prefer to be guided, even when they feel the need to change some part of business process. In this paper we propose CF4BPMN, a BPMN language extension to allow modeling and execution of controlled flexibility in business processes. Using this extension, process designers can express how a certain process element can or cannot be changed in execution time, taking into account their experience or other organizational restriction. Then, other process participants can visually learn and follow the advised changes onto a business process in a controlled manner.

Keywords: Business processes, BPMN, controlled flexibility, BPMN extension
1. Introduction

To address changes in environment, business processes need to be flexible. According to the recent study of Cognini et al. [19], business processes are subject of adaptation mainly due to external environment changes, outside the competence of the organizations, for example because of a change in the law. Others reasons that justify business process adaptations are changes in internal environment and in strategic goals, as well as in the approaches to reach these goals.

Since the early 1990’s, process flexibility has been a constant concern. Process flexibility denotes the ability to modify only those parts of a process that need to be changed while keeping other parts stable [17].

Although flexibility is considered essential so that processes can cope with expected (or unexpected) exceptions, in the everyday business practice, process participants do not wish for total flexibility, i.e., changing processes without any restrictions or guidance. Instead, designers would like to define which changes can be applied, as well as performers would like to follow advices previously modelled on which and how they can change the elements that compose business processes [3, 5]. This controlled flexibility can be defined as the ability to control which, where, how and by whom the elements that composed a business process can or cannot be changed.

Since its release in 2004, BPMN (Business Process Model and Notation) [15] is becoming the leader and a de facto standard in business process modeling [9]. Considering flexibility features in BPMN business processes, we can find in the literature some works that take advantage of standard BPMN elements, such as the ad-hoc sub-process BPMN element, to define more flexible business processes [23, 22]. In addition, some works propose to extend BPMN with rule-based languages to improve flexibility by design [13] and with versioning features to support flexibility by changes [1, 2]. The jBPM (http://www.jbpm.org) engine also supports flexibility by change, including the migration of running process instances.

However, none of the current proposals have the necessary expressivity and features to let the process designer to advise, and then to guide process performers in changing operations, with distinct degrees of enforcement and details.

In this paper, we propose CF4BPMN: a BPMN extension for controlled flexibility. The extension allows process designers to express, within a process model, how it is advised for a process participant to change a certain process element (including tasks, events, gateways, data objects and swimlanes, for instance), according to the well-known business process flexibility taxonomy of Regev et al. [18]. Then, process participants can visualize and learn from the modeled controlled flexibility constraints, and act accordingly, i.e., be guided in changing or not the process models and/or instances, taking into account the advised changes.

In the next section, we present related work about flexibility in general and particularly concerning BPMN processes. The BPMN extension for controlled flexibility is presented in section 3. Finally, section 4 concludes the paper and discusses future work.

2. Related Work

Among others, Regev et al. present in [18] a taxonomy of flexibility in business processes. They classify it according to three orthogonal (combinable) dimensions:

1. Abstraction level of change, that distinguishes where changes are to be made, i.e., if at the type or instance levels (or both). Changing the process model (type level) implies changing the defined standard way of working, as it will affect all instances created there forward. However, change can occur only for certain instances of a process (instance level), in order to accommodate exceptional situations;

2. Subject of change, representing which modelling elements are to be changed. For instance, considering BPMN, modelling elements would include 1) flow objects (events, activities, gateways); 2) data (data objects, data inputs and outputs, data stores); 3) connecting objects (sequence flows, message flows, associations and data associations); 4) swimlanes (pools and lanes); and 5) artifacts (groups and text annotations); and

3. Properties of change, denoting how a modelling element can be changed. We can consider four combinable properties of change:
a) Extent of change, denoting if change is only introduced to an already existing process model (incremental change), or if change abolishes the existing process model and creates a completely new one (revolutionary change). Often experts are required to do revolutionary changes to the whole or part of a process model;

b) Duration of change, which can represent temporary or permanent changes. Temporary changes are valid for a limited period of time, and permanent changes are valid until the next permanent change;

c) Swiftness of change, that expresses if changes are to be applied immediately to all family-related process models (types) or instances (also the running ones), or deferred only to new process models or instances of the changed process; and

d) Anticipation of change, that identifies if the change is planned or ad-hoc. Ad-hoc changes are often made to tolerate exceptional situations, and planned changes are often part of a process redesign.

From some of the most prominent works on workflow flexibility [20, 6], we highlight ADEPT and YAWL. The ADEPT project starts in 1995 and it is at the origin of the AristaFlow BPM suite [16]. ADEPT covers changes at both type (model) and instance abstraction levels, and allows process participants to choose between immediately apply changes to all running instances (with an appropriate migration strategy) or deferring them only to be applied to new instances. The YAWL environment has been extended with the Worklets Service, which provides sub-services to support flexibility in workflows (anticipation of change): a selection sub-service, which enables planned changes for process instances; and an exception handling sub-service, which provides facilities to handle both planned (expected) and ad-hoc (unexpected) process exceptions at runtime [14]. However, both systems use their own process modeling language and notation.

In the following, we use this taxonomy to systematize the state of the art on BPMN flexibility. For instance, in prescriptive modelling approaches, all the alternative workflow paths are included in the process model (or type, according the mentioned taxonomy). At runtime, the most appropriate execution path is selected for each process instance. Declarative or constraint-based modelling approaches are considered more flexible, as everything that does not violate the constraints is allowed [21]. In this case, more flexible process models have fewer constraints. In addition, many of the workflow patterns [22] can be seen as constructs to support flexibility at the model (type) abstraction level of change.

BPMN considers the Ad-Hoc Sub-Process object to support flexibility at the model (type) abstraction level of change, as well as ad-hoc anticipation of change and incremental extent of change. This object is a group of BPMN activities that can have no required sequence or other control-flow relationships. Here, the subject of change involves activity type modeling elements, as well as control-flow elements such as sequences or parallel flows. Ad-hoc sub-processes have a set of activities, but their control-flow and number of executions is determined by performers at the instance abstraction level of change [15]. In [23], the authors use BPMN ad-hoc sub-processes to design flexible clinical workflows.

Following a constraint-based modelling approach, in [13], the authors take advantage of their rule-enhanced business process modeling language (rBPMN) to support flexibility at the model (type) abstraction level of change. They use rules to define many alternative paths in BPMN processes in a compact way.

In [1] and [2], the authors present a BPMN extension to support process models’ versioning. The jBPM, a BPMN engine from the JBoss™ company, supports changing models during runtime, including the migration of running instances. This is an example of process flexibility at both model (type) and instance abstraction levels of change, with an immediate swiftness of change.

Despite the importance that flexibility has in business processes, in practice, both designers and performers do not want total flexibility, without any restrictions. Designers want to define which changes can be applied and performers would like some guidance at least [3, 5, 11]. In [6, 4], the authors present an approach that combines BPMN processes, rules, events and workflow adaptation patterns, which supports flexibility but also goes a step further by defining which changes can be applied, in which situations and how. In a more systematic way, Martinho et al. propose a framework for modelling controlled flexibility in software processes [12, 10]. They propose the Controlled Flexibility Language and they extend the Unified Modelling Architecture (UMA) metamodel [7] with its language constructs. To provide it with controlled flexibility, in this paper we extend BPMN with the Controlled Flexibility Language constructs, as we detail in the next section.
3. The CF4BPMN controlled flexibility extension

Extending BPMN to support controlled flexibility includes defining a domain model containing the main concepts of controlled flexibility language, as well an extension model as described by the native BPMN extension mechanisms. In this section we define both these models and include also an example of our CF4BPMN extension applied to a BPMN diagram.

3.1. Conceptual Domain Model for Controlled Flexibility

Based on our previous work on concept maps and ontologies for controlled flexibility in software processes [10], we propose the Controlled Flexibility Language (CFL) whose UML metamodel is presented in Figure 1.

The metamodel is based on the notion of constraint of change (COfChange), which defines that a change to a certain business process element can be constrained either positively (if it is changed, it can or must be done in a certain way), or negatively (if a change occurs, it should not or cannot be done in a certain way). The CFL metamodel has three related substructures, denoted by different grey-shaded classes and their corresponding associations:

1. ConFlexElement and COfChange;
2. CFEExpression, CFLPart, CFLNameLiteral, CFLTypeLiteral and CFLValue; and
3. CFEExpression and the corresponding hierarchy of expressions, including the shared association between ExpressionDecorator and CFEExpression.

The first substructure illustrates a composite association between ConFlexElement and COfChange. The ConFlexElement on the right defines an abstract superclass for all possible process elements for which the process designer can define controlled flexibility. A controlled flexibility-enabled element is a process element...
which has, at least, one COfChange associated. The composite association also denotes that a constraint of change only exists within the context of a ConFlexElement. In turn, a COfChange has a one-to-one association with CFLExpression, defining that constraints of change must have exactly one expression.

The second substructure includes the definition and associations of the CFLExpression class. It represents a literal expression (string), which will contain all the semantic information of a COfChange. For example, we can consider a CFLExpression with a body attribute containing the following string:

\{tbAdvice:TBAdvice=recommended, absLevel:AbsLevel=model\}

The expression informs a process participant that it is recommended to change the model representation of the constrained ConFlexElement. We can observe that the expression follows a tuple-like format, and results from the composition of a variable number of CFLEPart elements (two for the example above). Each CFLEPart has a name literal (CFLNameLiteral), a type literal (CFLTypeLiteral) and a value (CFLValue). For the example above, the first CFLEPart instance is defined by a tbAdvice name literal, a TBAdvice type literal and the recommended value. A CFLExpression must have, at least, one CFLEPart, and a CFLEPart belongs only to one CFLExpression at a time. The CFLTypeLiteral of a part is optional (association with multiplicity of 0..1 to 1), and therefore the expression above can also be written in its simplest form:

\{tbAdvice=recommended, absLevel=model\}

The third substructure of the CFL constructs is composed by CFLExpression and the corresponding hierarchy of expressions, along with the shared association between ExpressionDecorator and CFLExpression. The structure applies the Decorator structural software pattern of Gamma et al. [8]. A constraint of change must be able to assume different degrees of detail in its definition. It can vary from having a single AdviceExpression, to a combination of CFLExpression elements, composing a constraint of change with more detail. The expression example above is itself a combination of a TBAdviceExpression with an AbsLevelExpression. The Decorator pattern from Gamma et al. [8] avoids the disadvantage of static subclassing. It enables a process engineer to combine several CFLExpressions of different types into a global one. This means that a mandatory AdviceExpression (leaf expression) can be decorated with other (non-leaf) expressions at runtime by the process engineer. The result is still a single CFLExpression object, which begins by being a single AdviceExpression, to which is later added responsibilities through the CFLExpression decorators, such as the AbsLevelExpression one. Back to the expression example above, its concrete implementation comprises a leaf TBAdviceExpression and an AbsLevelExpression decorator.

3.2. BPMN Extension

The BPMN 2.0 meta-model includes an extensibility mechanism to add non-standard attributes and elements to standard BPMN elements. A BPMN extension consists of four elements:

1. Extension - binds/imports an ExtensionDefinition and its attributes to a BPMN model definition;
2. ExtensionDefinition - defines and groups additional attributes that can be added to BPMN elements;
3. ExtensionAttributeDefinition - defines new attributes that can be added to BPMN elements; and
4. ExtensionAttributeValue - contains the attribute value.

Figure 2 presents the class diagram of BPMN extension. By associating a BPMN element with an ExtensionDefinition, every BPMN element which subclasses the BPMN BaseElement can be extended with additional attributes.

The name of the extension we propose, CF4BPMN, is the value of the ExtensionDefinition attribute name. The value of the mustUnderstand attribute of the Extension class is false - if the BPMN tool does not understand this extension, it can ignore it, losing the controlled flexibility support. The new CF4BPMN attributes are COfChange, CFLExpression, and CFLEPart, defined as ExtensionAttributeDefinitions.
3.3. BPMN Extension application example

Our extension includes also an altered graphical representation of the BPMN elements that are to be marked as controlled flexibility-enabled elements. This is achieved by marking these elements with a green-coloured background, and associating them an annotation element (also coloured), which will contain the CFExpression body text, as described in section 3.1. Figure 3 shows, as an example, the BPMN diagram of the Elaboration phase of the Unified Process (UP), as detailed in [24] with controlled flexibility-enabled elements.
In this example, we can observe the following four applications of the CF4BPMN extension:

1. Data object labeled “Use case model (briefly described), with the CFLExpression value of {tbAdvice=denied, absLevel=model}, meaning it that are denied (tbAdvice - text-based advice) any changes performed at the model (type) abstraction level of change;

2. Sequential Multi-Instance subprocess, with the CFLExpression value of {tbAdvice=discouraged, duration=temporary}, meaning that it is discouraged to make temporary changes to this subprocess, i.e., changes valid only for a limited period of time (referred to the duration property of change described in section 2).

3. Merge parallel gateway, with the CFLExpression value of {vbAdvice=70%, absLevel=instance, operation=synch_partial}, meaning that 70% (vbAdvice – value-based advice) of instances were changed in order to allow for a partial synchronization of incoming paths (i.e., the gateway allowed for the process to proceed for the next element with only one incoming path reaching it);

4. The lane labeled “Use case engineer”, with the CFLExpression value of {tbAdvice=allowed, absLevel=instance, swiftnessMode=immediate}, meaning that changes to whom is going to perform the Tasks in this lane are allowed, but only at the instance abstraction level of change, being immediately applied to all (running) instances of this element.

It is important to notice that our work assumes that flexibility is unconstrained (i.e. totally free) for any unmarked BPMN element. For instance, changes to the “Define architecture” Task in Figure 3 are completely allowed (not constrained by any expression).

The same way, any flexibility concept not referred within the CFLExpressions should assume a default value. For instance, the CFLExpression on the sequential multi-instance subprocess does not explicitly refer the abstraction level of change, meaning that the included advice may be assumed for both models and instances of this element’s abstraction level of change (default value for absLevel=model_instance).

4. Conclusions and future work

In this paper we proposed the CF4BPMN extension in order to enable controlled flexibility in business processes. Through this extension, process designers are able to express which, where and how certain elements of a business process can be changed. This allows for other process participants to be aware of the advised changes they can perform on a business process. The concepts behind these advised changes are taken from the business process flexibility taxonomy in [18], from which we derived an extension metamodel. The metamodel is based on the notion of constraints of change, which in turn include a tuple-like text-based expression stating the controlled flexibility to which a certain process element is subjected to.

In our extension to BPMN we propose the use of green-coloured process elements to distinguish them from those which have not controlled flexibility. We also use coloured annotations to convey the tuple-like constraint expressions defined in our metamodel, taking advantage of a well-established way to add extra information to BPMN diagrams.

Future work includes the implementation of CF4BPMN in a BPMN-compliant business process suite, comprising the necessary process design and execution features. We are also pursuing organizations where there is already an informal way of controlling flexibility in business processes. This will allow us to test our extension and to improve it taking into account additional requirements from real-world scenarios.

References


