Instituto Alberto Luiz Coimbra de Pós-Graduação e Pesquisa de Engenharia COPPE UFRJ

BPMNT: A PROPOSAL FOR FLEXIBLE PROCESS TAILORING REPRESENTATION IN BPMN

Raquel Mainardi Pillat Basso

Tese de Doutorado apresentada ao Programa de Pós-graduação em Engenharia de Sistemas e Computação, COPPE, da Universidade Federal do Rio de Janeiro, como parte dos requisitos necessários à obtenção do título de Doutor em Engenharia de Sistemas e Computação.

Orientador: Toacy Cavalcante de Oliveira

Rio de Janeiro
Março de 2018

BPMNT: A PROPOSAL FOR FLEXIBLE PROCESS TAILORING REPRESENTATION IN BPMN

Raquel Mainardi Pillat

TESE SUBMETIDA AO CORPO DOCENTE DO INSTITUTO ALBERTO LUIZ COIMBRA DE PÓS-GRADUAÇÃO E PESQUISA DE ENGENHARIA (COPPE) DA UNIVERSIDADE FEDERAL DO RIO DE JANEIRO COMO PARTE DOS REQUISITOS NECESSÁRIOS PARA A OBTENÇÃO DO GRAU DE DOUTOR EM CIÊNCIAS EM ENGENHARIA DE SISTEMAS E COMPUTAÇÃO.

Examinada por:

RIO DE JANEIRO, RJ - BRASIL

MARÇO DE 2018

Basso, Raquel Mainardi Pillat

BPMNt: A Proposal for Flexible Process Tailoring Representation in BPMN/ Raquel Mainardi Pillat Basso. – Rio de Janeiro: UFRJ/COPPE, 2018.

XIII, 186 p.: il.; 29,7 cm.

Orientador: Toacy Cavalcante de Oliveira

Tese (doutorado) – UFRJ/ COPPE/ Programa de Engenharia de Sistemas e Computação, 2018.

Referências Bibliográficas: p. 157-166.

1. Process tailoring. 2. Process adaptation. 3. BPMN. I. Oliveira, Toacy Cavalcante de. II. Universidade Federal do Rio de Janeiro, COPPE, Programa de Engenharia de Sistemas e Computação. III. Título.

Para Fábio.

Agradecimentos

Em primeiro lugar, agradeço à Deus, por me acompanhar ao longo do caminho e permitir que eu chegasse até aqui.

A minha família, pelo apoio, amor e incentivo neste período em que estive distante. Em especial a meu marido, Fábio, que sempre me apoiou e foi crucial para a conclusão dessa tese. A minha mãe, Lúcia, meu pai, Luiz Clóvis, e minha irmã, Micheli.

Ao meu orientador, professor Toacy Oliveira, pelos direcionamentos, incentivo, e confiança. Obrigada por me ajudar a chegar até aqui.

Aos professores do PESC que participaram do meu aprendizado e formação.

À UFRJ como um todo, que deu todo suporte necessário, e ao CNPq, CAPES e PESC que contribuíram também com apoios financeiros.

Aos professores Cláudia Werner, Geraldo Xexéo, Renata de Araujo e Adriano Albuquerque por terem aceitado fazer parte desta banca.

Resumo da Tese apresentada à COPPE/UFRJ como parte dos requisitos necessários para a obtenção do grau de Doutor em Ciências (D.Sc.)

BPMNT: UMA PROPOSTA PARA REPRESENTAÇÃO DE ADAPTAÇÃO DE PROCESSO FLEXÍVEL EM BPMN

Raquel Mainardi Pillat Basso

Março/2018

Orientador: Toacy Cavalcante de Oliveira

Programa: Engenharia de Sistemas e Computação

BPMN (Business Process Model and Notation) é um padrão para modelagem de processos de negócio, que tem seu foco na representação do comportamento de processos. No entanto, ele pode também ser usado para representar o comportamento de processos de software, já que eles são um tipo de processo de negócio. Embora BPMN tem sido extensivamente usado para modelar processos em diferentes domínios, sua especificação padrão não possui nenhum mecanismo para apoiar usuários em atividades relacionadas à adaptação de processos. Pesquisas que estendem o padrão são baseadas em modelos complexos, que dificultam a análise e manutenção de modelos variantes, e não são apropriadas para domínios de aplicação onde variações de processo são difíceis de predizer, como em processos de desenvolvimento de software. Assim, nosso objetivo foi fornecer uma extensão para BPMN, chamada BPMN*t*, e mecanismos de suporte para especificar, de modo flexível, adaptações em processos modelados com esta linguagem. BPMN*t* deve também garantir a corretude de modelos adaptados e explicitamente capturar rastros de mudanças realizadas. Essa pesquisa teve como foco os domínios de Engenharia de Processos de Software e Gerenciamento de Processos de Negócio. Por fim, nós avaliamos a aplicabilidade da proposta para representar cenários de adaptação reais em ambos os domínios.

Abstract of Thesis presented to COPPE/UFRJ as a partial fulfillment of the requirements for the degree of Doctor of Science (D.Sc.)

BPMNT: A PROPOSAL FOR FLEXIBLE PROCESS TAILORING REPRESENTATION IN BPMN

Raquel Mainardi Pillat Basso

March/2018

Advisors: Toacy Cavalcante de Oliveira

Department: Computer Science and Systems Engineering

Business Process Model and Notation (BPMN) is a *de-facto* standard for business process modeling, which focuses on the representation of the process behavior. However, it can also succeed in representing the behavior of software processes, since they are a type of business process. Although BPMN has been extensively used for modeling processes in different domains, its standard specification does not have any mechanism to support users in activities related to process adaptation (tailoring). Moreover, researches extending BPMN are based on complex consolidated models, which hamper the analysis and maintenance of individual variant process models and are not appropriate for application domains in which process variations are difficult to predict, such as in software development processes. Thus, our objective was to provide a BPMN-compliant extension and associated mechanisms for specifying flexible process tailoring on models produced with this language while ensuring the correctness of adapted process models and explicitly capturing change traces. We have focused our research on the domains of Software Process Engineering (SPE) and Business Process Management (BPM). At last, we evaluated the applicability of the proposal for representing realistic tailoring scenarios in both domains.

Table of Contents

List of Figures

List of Tables

CHAPTER I

1. Introduction

1.1. Motivation

Nowadays, there is a consensus that managing processes (at different levels of abstraction) is essential for organizational performance and, for this reason, process-oriented approaches are already institutionalized in most organizations (SHARP and MCDERMOTT, 2009). In the context of an organization, a process defines how its activities are structured in a coordinated manner in order to reach business objectives (WESKE, 2007). However, processes are not static and often need to be adapted to specific contexts where they will be applied or improved due to organizational learning.

Process adaptation is a topic widely discussed in the literature, with extensive range of application domains (e.g., engineering, health, business management, software development, etc.). As a wide-range concept, the term process adaptation (and its equivalents) varies in meaning from one research community to another. In this thesis, we consider process adaptation from the perspective of two research communities, *Business Process Management* (BPM) and *Software Process Engineering* (SPE). In the context of a software organization, the software development process represents its main business process, since it prescribes the activities that must be carried out when creating and maintaining software products to reach business objectives. Therefore, software development processes (software process, for short) are also a type of business process (HENDERSON, 1994) (BENDRAOU and GERVAIS, 2007) (CAMPOS and OLIVEIRA, 2013) and we argue that it is beneficial for both areas, BPM and SPE, to share noteworthy technologies, techniques or tools.

Within the community of *Business Process Management* (BPM), adaptation is often considered a specific type of flexibility, concept of broad scope that represents definitely a key concern (VAN DER AALST, 2013). According to SCHONENBERG *et al.* (2007) *flexibility* reflects the ability of a process to deal with foreseen and unforeseen changes, by varying or adapting those parts of the business process that are affected by them. Adaptation, in turn, deals with unforeseen changes (e.g., special situations that occur rarely). Different taxonomies have been proposed in the technical literature of BPM aiming to precisely define typical application scenarios and nomenclatures related to process flexibility/adaptation (e.g., SCHONENBERG *et al.,* 2007; REICHERT and WEBER, 2012). However, currently there is not a consensus on it. We can only state that the concept of adaptation in BPM includes changes performed at both design- and run-time. Design-time adaptation leads to a variant process model that is intended to be executed in a particular organizational setting. Hence, it affects all instances of the business process executed in this setting. In contrast, run-time adaptations are unique and affect only one process instance. Such adaptations are not intended to modify the executed process model itself, beyond its effects on the process instance where the decision is applied (LA ROSA, 2017).

On the other hand, in the *Software Process Engineering* (SPE) domain, process adaptation is a concept typically employed to refer to changes or adjustments performed at design-time, usually referred as *process tailoring*. In this context, adaptation can be considered as a reuse technique (YOON, MIN and BAE, 2001), in which a new process is created from an existing one by adjusting its definition to meet specific needs of a given organization or project (GINSBERG and QUINN, 1995) (PEDREIRA *et al.*, 2007). However, as GINSBERG and QUINN (1995) have already claimed in their seminal document, "*tailoring is not a one-time event, but a repeated, ongoing analysis*" that integrates a process improvement program, suggesting that tailoring is continuously necessary in order to improve organization's processes. Still, tailoring can also be defined in terms of modifications that emerge from the monitoring of executions of a process, providing feedback on its definition (FERRATT and MAI, 2010) (SANTOS, OLIVEIRA and ABREU, 2015). In summary, process tailoring is an important activity for establishing and improving processes in software organizations (MARTÍNEZ-RUIZ *et al.*, 2012) and such a perspective on process adaptation can also be applied to business processes in general.

In this thesis we are especially interested in process adaptation at design-time. In order to clearly set up our scope under investigation, in this research we consider process adaptation from the perspective of the SPE community, such as described above, but apply such a concept to business processes in general (not only for software processes). Thus, we consider process adaptation any activity in which a process is derived from an existing one by refining and/or modifying its definition in order to meet specific needs for a given environment/context or incorporate evolution improvements. In this way, the terms adaptation and tailoring are considered synonymous in this research. However, unlike many solutions for process tailoring from the literature, our research has not the purpose of managing or constraining the way as processes can be adapted. Process engineers are allowed to freely adapt processes according to each current need.

Moreover, although process adaptation involves aspects from several existing disciplines, including organizational science, information science, computer science, and sociology (WESKE, 2007), in this thesis we focus on adaptation of business processes to support the operational level of an organization. In particular, our interest is on methodological and technological issues related to process adaptation, whereas managerial issues are outside the scope of this thesis.

A first step towards process adaptation within any organization is to explicitly represent its processes through *process models*. A process model aims to capture the different ways in which a case (i.e., process instance) can be handled. Process models are especially useful to analyze, understand, and improve the processes they describe (VAN DER AALST, 2013). In this sense, BPM systems provide important computational support. These systems are driven by process models to enact (execute) and manage operational business processes (VAN DER AALST, TER HOFSTEDE and WESKE, 2003), covering the scope of an entire process lifecycle. Thus, the explicit representation of business processes and the adoption of BPM systems is certainly an important step toward increased awareness on organization's activities and will allow improving its reactivity to changes (COGNINI *et al.*, 2014).

Business processes are usually described in terms of activities (and subprocesses) ordered according to causal dependencies. The *control-flow perspective* (modeling the ordering of activities) is often the backbone of a process model (VAN DER AALST, 2013). Although other perspectives, including the *resource perspective* (modeling roles, organizational units, authorizations, etc.), the *data perspective* (modeling creation and use of data, forms, etc.), and the *functional perspective* (describing activities and related applications), are also important for comprehensive process models, it is common to find business process models where these perspectives are not represented. For this reason, the solution presented in this thesis focuses on the control-flow perspective of process models.

Various *Process Modeling Languages* (PMLs) exist to represent business processes, e.g., Petri nets (HACK, 1976), BPMN (OMG, 2011), UML (OMG, 2015), and EPCs (MENDLING, 2008). However, BPMN 2.0 has been standing out as a leading technology since it is an ISO (ISO, 2013) and OMG standard (meta-model and notation) for business process modeling. Nowadays, BPMN is the business process notation most used among BPM practitioners[1] (HARMON, 2016) and with the highest number of available tools. BPMN models can be interpreted and manipulated by both technical and non-technical personnel, reducing the chance of erroneous knowledge transfer (OMG, 2011). Moreover, BPMN can also express executable models (since its 2.0 version) that are automatically interpreted by BPM systems. In fact, systems such as Camunda[2], Flowable[3], and BonitaSoft[4] are able to deliver an integrated environment where users can design and run BPMN models. At last, BPMN also has available transformations to other notations, such as Petri Nets, which allow the use of tools for formal verification.

Unlike formal languages such as Petri Nets (HACK, 1976), YAWL (VAN DER AALST and TER HOFSTEDE, 2005), and ADEPT (REICHERT *et al.*, 2005) that the semantics is based on mathematical theories (VERGIDIS, TIWARI and MAJEED, 2008) (preventing any kind of ambiguity), BPMN has precise syntax but semantics given in natural language. However, according to VAN DER AALST (2013), users in practice often have problems using formal languages due to the rigorous semantics and low-level nature. They typically prefer to use higher-level languages such as BPMN.

1.2. Problem Statement

Although BPMN has been extensively used for modeling business processes since its launch as an OMG standard in 2008, its current specification (OMG, 2011) still does not have any mechanism to support users in activities related to process adaptation (tailoring). Likewise, the most prominent BPM systems based on this technology also do not provide support for such activity.

[1] BPMN is used by 64% of respondents in survey published by BPTrends
[2] https://camunda.org/
[3] www.flowable.org/
[4] https://www.bonitasoft.com/

When using BPMN and its current BPM systems, *process variants*, i.e., process models that pursue the same or similar business objective (e.g., product sale, car maintenance, or software development), are usually defined and maintained in separate process models without any connection with each other (HALLERBACH, BAUER and REICHERT, 2009). This solution typically results in highly redundant model data because process variants are identical or similar in most parts. When considering the large number of variants that generally occur in practice, this approach leads to significant modeling and maintenance efforts. Particularly, efforts for maintaining and changing process variants become high since process changes (e.g., due to new or changed legal regulations) have to be separately accomplished for each individual variant model, which is both time-consuming and error-prone (HALLERBACH, BAUER and REICHERT, 2009).

Consequently, organizations may have repositories containing many variations of the same process model for different departments or products without any relation established between them. Moreover, due to continuous improvement practices and organizational learning, processes may also change over time resulting in different versions. Due to the lack of appropriated support for process reuse, including the activity of adaptation, process models may be built from scratch without reusing existing models (VAN DER AALST, 2013). As a result, even more process models need to coexist, further hampering model management. In such contexts, techniques are needed to keep track of process variants, understand their common points and differences, and co-evolve them over time (DIJKMAN, ROSA and REIJERS, 2012).

In this sense, we have found BPMN-based approaches aiming to support *process variability* modeling (REICHERT and WEBER, 2012). Variability can be found in many domains and requires processes to be handled differently, resulting in different process variants, depending on the given context (GOTTSCHALK *et al.*, 2009) (HALLERBACH, BAUER and REICHERT, 2010) (REICHERT and WEBER, 2012). Process variants typically share the same core process whereas the concrete course of action changes from one variant to another.

In general, variability approaches represent at design-time all possible process variants for a given domain (i.e., a process family) into a *configurable process model*. This model may be customized for a particular setting by hiding (i.e., bypassing) or blocking (i.e., inhibiting) certain fragments of the configurable model (GOTTSCHALK, VAN DER AALST and JANSEN-VULLERS, 2007). In this way, the desired behavior is selected.

However, although variability modeling through configuration may facilitate planned process reuse (requiring only the selection of alternative process fragments from a configurable model), it is generally suitable in well-defined domains, where all alterations are previously known (REICHERT and WEBER, 2012). Adaptations to generate specific process variants at design-time are usually limited to addition and removal of process fragments defined in advance. In this way, variations limited to a given set of options and in specific process parts can prevent new or changing needs of an organization or department from being properly addressed in time. Moreover, there are application domains in which process variations are difficult to predict, such as in software development processes. This type of process is influenced by several complex factors that are still poorly understood (KALUS and KUHRMANN, 2013) (CLARKE and O'CONNOR, 2012).

Still, there are use cases in which unanticipated changes are required in order to improve or evolve a business process (VAN DER AALST and JABLONSKI, 2000) (REICHERT and WEBER, 2012). In such cases, adaptations are planned at design-time to meet *new* needs (unanticipated), extending or modifying existing process models. These adaptations can be driven by changes in the business, technological environment, and legal context (VAN DER AALST and JABLONSKI, 2000), as well as by performance or quality issues related to the process model (WEBER *et al.*, 2011). Another motivation is organizational learning. In this last case, changes are motivated by optimization/improvement opportunities or misalignments between real-world processes and those ones represented by process models (REICHERT and WEBER, 2012).

According to REICHERT and WEBER (2012), evolution changes may be incremental, only requiring small changes in the process model as for continuous process improvements (PANDE, NEUMAN and CAVANAGH, 2000), or be revolutionary, requiring radical changes as in the case of process innovation or re-engineering (HAMMER and CHAMPY, 2003). In general, such changes are applied on a single process model, affecting only its new instances. However, in some practical scenarios, evolution changes should have effect on other process models as well. For example, when changing a reference process model can be desirable to propagate such changes to its variant process models. Such a situation is usually referred as *co-*

evolution. In this research field, a large body of knowledge exists and has been surveyed (HEBIG, KHELLADI and BENDRAOU, 2017) (HERRMANNSDÖRFER and WACHSMUTH, 2014) (PAIGE, MATRAGKAS and ROSE, 2016). However, providing such types of solution (i.e., for model co-evolution) is out of the scope of this thesis. We are only concerned with aspects related to the suitable representation of adaptations, which can also occur in such scenarios.

Finally, in the context of BPM initiatives it is also common to find use cases in which a business process model needs to be adapted from a general business specification (which focuses on the concepts and rules relevant to business analysts) to technical-level specifications (which detail tasks and flows as well as add technical exceptions) in order to make clear issues related to the implementation of the process (BRANCO *et al.*, 2014) (KÜSTER et al., 2016). The final aim of this technical-level specification is to obtain an executable process model that can be directly enacted into a BPM system. Typically, these representations of the same process in different levels of abstraction are created and maintained in different process models to effectively separate concerns and to convey the right information (with proper level of abstraction) to groups of stakeholders (BRANCO *et al.*, 2014). Such process models can be conveniently modeled in BPMN 2 (OMG, 2011), since it supports appropriated modeling concepts to both business and IT-level concerns. The derivation of technical-level process models from business-level process models is referred in the literature of the area as Business-IT *refinement* (BRANCO *et al.*, 2014).

Despite the importance of flexible adaptations for BPM contexts, we have not identified researches proposing extensions of BPMN for explicitly specifying this type of variation on its process models. In other words, BPMN still lacks a flexible and comprehensive mechanism to address process adaptation (beyond the variability modeling). An important aspect related to this type of solution involves the correctness of the produced model. That is, adaptation solutions should provide some mechanism to prevent the specification of incorrect models.

An aspect on correctness of process models is related to their structural correctness (ROSA *et al.*, 2017). In this case, a correct process model must not contain flow breaks, i.e., disconnected process nodes. Another aspect related to correctness is the well-formedness of the process model concerning the modeling language used to build them. BPMN is a semantically rich modeling language. While, for example, a UML activity diagram has around 20 different modeling constructs, a BPMN process model diagram has around 100 different modeling constructs, including 51 event types, 8 gateway types, 7 task types, etc. (CORREIA and ABREU, 2012). If process designers are allowed to freely specify/adapt models by combining such a plethora of modeling constructs, incorrect models can easily be produced (CORREIA and ABREU, 2015). Even if only a subset of these elements is taken into account, it is still important to impose well-formedness rules when specifying BPMN process models for reducing the sources of modeling malformation (CORREIA and ABREU, 2015). As mentioned above, since the amount of BPMN process elements and their possibilities of combination are huge, automated support for process tailoring becomes very important, and to this end it is necessary to specify a set of rules compliant to this language. In this thesis, we consider process model correctness concerning the two mentioned aspects, i.e., structural correctness and well-formedness.

Moreover, another important aspect related to tailoring is change traceability. Traceability implies keeping track of the relationships between different artifacts involved in any development process so that this traceability information can help in the evolution of such artifacts over time (VARA *et al.*, 2014). As tailoring is applied, different process models are produced from a base process model and, at any time, this base process can need modifications aiming its improvement. In this scenario, it may be necessary to propagate updates of the base process to variant processes in order to keep them consistent with the original process (DIJKMAN, ROSA and REIJERS, 2012) (KUHRMANN *et al.*, 2016). A prerequisite for propagating (manually or automatically) such modifications to variant processes is to track the changes previously performed (DIJKMAN, ROSA and REIJERS, 2012) (KUHRMANN *et al.*, 2016).

Thus, the main research question that guided this thesis was: *How to extend BPMN to support flexible process tailoring in different application scenarios while ensuring the correctness of tailored process models as well as explicitly capturing change traces?*

1.3. Goals

Motivated by the lack of support for process tailoring in BPMN, we propose a meta-model extension and associated infrastructure to address process adaptation especially designed to this technology. As stated by AYORA *et al.* (2015), "*it might be* *more suitable to focus on a well established process modeling language (e.g., standardized languages) as well as to develop adaptation techniques optimized for this language. In particular, this would facilitate its industrial adoption and evaluation*".

Our BPMN extension, named BPMN*t* (BPMN + tailoring), (and support mechanisms) aims at specifying flexible process tailoring in different application scenarios, ensuring the correctness of tailored process models and explicitly capturing change traces.

Thus, this thesis research has the following specific goals:

(1) ***Specify a conceptual representation of tailoring compliant with the BPMN standard:*** Tailoring operations must be conceptually represented and associated to the BPMN process meta-model in order to facilitate the specification and management of adaptations on models of the language. This goal involves introducing tailoring concepts in BPMN, related to basic and high-level operations, by using its standard extension mechanism.

(2) ***Define a change traceability mechanism:*** Tailoring operations applied to a variant process will be recorded into the variant model itself as extension information. Moreover, configuration parameters of operations will be used as traceability links connecting new variant process elements to adapted base process elements. These links will be created and kept in order to facilitate the identification of adaptations (changes) after tailoring.

(3) ***Define a catalog of BPMN tailoring operations:*** The catalog must include a complete set of basic operations as well as the main adaptation and refinement patterns for BPM.

(4) ***Specify rules ensuring the well-formedness of tailored models regarding the BPMN specification:*** Tailoring operations must have pre- and post-conditions associated. Pre-conditions aim at verifying if an operation can be applied on a given process element or set of elements and post-conditions aim at adjusting the resulting process model after the tailoring for ensuring its validity concerning constraints of the BPMN standard.

(5) ***Implement a tool support:*** A prototype must be implemented to evaluate and validate the proposed approach, allowing adaptation of BPMN-based process models through the tailoring operations proposed in this research.

1.4. Research Methodology

This thesis followed the research stages shown on the left part of Figure 1, which are based on the methodology proposed by PEFFERS *et al.* (2007). Such stages are: (1) Identify the problem and define objectives of the solution; (2) Design and development of the solution; (3) Demonstration; (4) Evaluation; and (5) Communication. Figure 1 also shows our main research tasks related to each stage.

We followed these stages in an iterative and incremental way, more specifically in two iterations. In the first iteration of the methodology we focused our research on the Software Process Engineering (SPE) domain, which was initially our target domain. In this way, we identified a problem, defined objectives to solve it, designed, developed, demonstrated, evaluated, and communicated the solution named SPEM-based BPMN*t*, which has been intended for representing software process tailoring. In the second iteration, we followed these same research stages, but now focusing on the Business Process Management (BPM) domain. This focus change was a recommendation of the researchers that evaluated our thesis proposal during the Doctoral Qualification Exam. Thus, the second iteration of the methodology resulted in the solution named Pattern-based BPMN*t*, which has been intended for representing business process tailoring in general.

Therefore, our proposal for dealing with adaptations on BPMN process models involved two application domains, SPE and BPM. Initially, we formulated our research problem considering the domain of SPE (i.e., dealing with adaptations in workflow-based software process models). This problem was identified from a literature review on process adaptation in SPE and then we defined requirements for a possible solution (i.e., SPEM-based tailoring support as an extension for the BPMN modeling language). The relevance of this initial research proposal was checked by submitting its description for peer review in an international conference in the area of software processes, in which it was accepted for publication as a short paper (PILLAT *et al.*, 2012).

Then, we designed and developed an extension of the BPMN meta-model for specifying process tailoring inspired in a widespread technology in the SPE domain (the OMG standard SPEM). This extension had software processes as its target domain and is referred in this thesis as SPEM-based BPMN*t* (depicted in Chapter 5). After, we demonstrated the solution by implementing a support prototype and evaluated its applicability for representing real tailoring scenarios through a study in the SPE domain. At last, this solution was published in a journal of the area (PILLAT *et al.*, 2015).

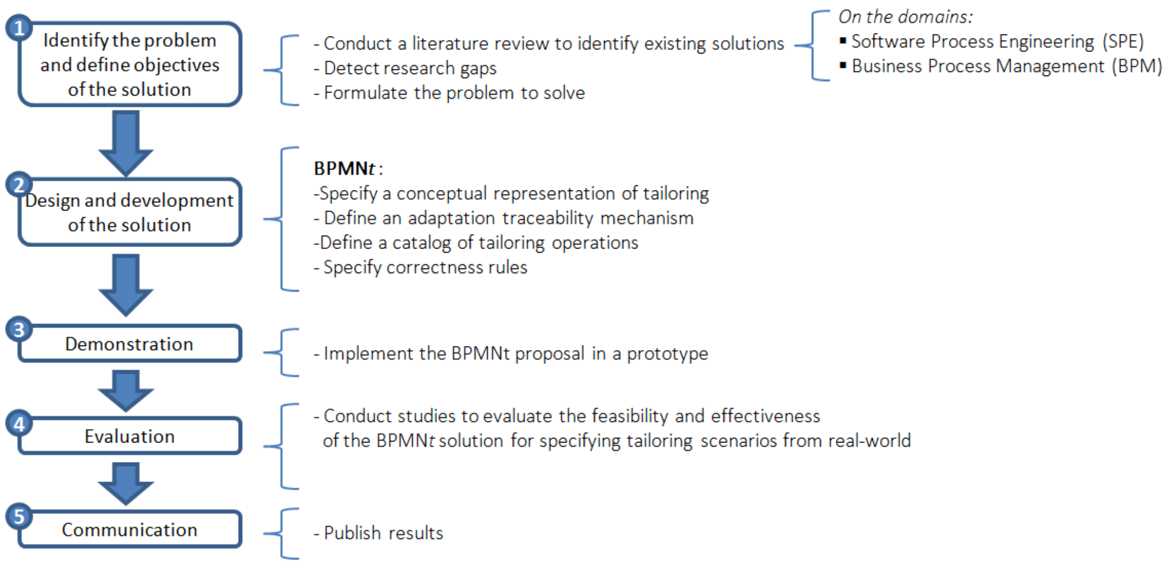
In the second iteration on the stages of the proposed methodology, we reformulated our research problem considering the domain of BPM, but focusing especially on the context of BPMN (i.e., how to support flexible process tailoring in BPMN while ensuring correctness of tailored models), which is the base technology of our proposal. We conducted an *ad-hoc* literature review on process adaptation in the general context of BPM and performed a *structured*[5] literature review on BPMN-based adaptation approaches, which are our main related works. In order to check the relevance of our more recent research proposal considering the domain of BPM, we have again submitted its description for peer review in an international conference, in which it was accepted for publication (PILLAT and OLIVEIRA, 2016).

Then, we designed and developed an extension of BPMN meta-model for specifying process tailoring based on high-level operations, which aim at ensuring the correctness of the adapted model. These operations were derived from *adaptation patterns* and *refinement patterns* in BPM, which are recognized researches in this field. This extension had business processes in general (including still software processes) as its application domain and is referred in this thesis as Pattern-based BPMN*t* (depicted in Chapter 6). After, we demonstrated the proposal by extending our previous support prototype and evaluated its applicability for representing real tailoring scenarios through studies in both domains (BPM and SPE). This complete proposal has not yet been published, but we are currently working on its description in an article. Therefore, during the second iteration of the methodology we expanded our application scope for business processes and solved limitations identified in our first proposal (i.e., SPEM-based BPMN*t*).

1.5. Outline

This thesis is organized in eight chapters. *Chapter 1* presented our motivation, problem, goals and research methodology.

[5] We called the review of *structured* and not *systematic* because it has been conducted uniquely by the researcher of this thesis.


1
Identify the problem and define objectives of the solution
- Conduct a literature review to identify existing solutions
- Detect research gaps
- Formulate the problem to solve
On the domains:
▪ Software Process Engineering (SPE)
▪ Business Process Management (BPM)
2
Design and development of the solution
BPMNt:
-Specify a conceptual representation of tailoring
- Define an adaptation traceability mechanism
-Define a catalog of tailoring operations
- Specify correctness rules
3
Demonstration
- Implement the BPMNt proposal in a prototype
4
Evaluation
- Conduct studies to evaluate the feasibility and effectiveness of the BPMNt solution for specifying tailoring scenarios from real-world
5
Communication
- Publish results
Figure 1: Stages of the research methodology (left) and related tasks (right)

Chapter 2 introduces concepts related to the Software Process Engineering (SPE) and Business Process Management (BPM) domains as well as their main technologies. This chapter also introduces concepts of process adaptation in both the domains (SPE and BPM) and the well-formedness of BPMN models.

Chapter 3 presents the main related works to this research. It summarizes techniques for process adaptation management, presents a review of adaptation operations proposed in the literature, discusses on the main approaches for process adaptation based on BPMN and compares them by using a set of criteria.

Chapter 4 introduces our solution, named BPMN*t*, presenting its structure and providing an overview on tailoring operations. At the end of this chapter, we summarize the two BPMN extensions that compose our solution.

Chapter 5 presents the first part of our solution, which consists of a support for adapting BPMN process models based on SPEM. SPEM is an OMG standard for modeling of software processes. Such a support has been intended for representing adaptations in the behavior of software processes. Thus, Chapter 5 presents the conceptual representation, implementation and evaluation of this tailoring support.

Chapter 6 presents our BPMN extension and support mechanisms based on high-level tailoring operations, which has been derived from adaptation patterns in BPM. This chapter also presents a catalog of high-level operations for BPMN and demonstrates the application of these operations in different adaptation scenarios.

Chapter 7 presents three studies evaluating the proposed solution. These studies have been conducted based on real process adaptation data from different contexts in both domains, SPE and BPM, and aimed at evaluating the feasibility of the BPMN*t* solution.

Chapter 8 presents our final remarks about this thesis research, including our main contributions, limitations of the proposed solution, implications and perspectives for future development.

CHAPTER II

2. Background

2.1 Software Process Engineering (SPE)

2.1.1 Concepts and Definitions

Software Process Engineering refers to "*the total set of software engineering activities needed to transform user's requirements into software*" (HUMPHREY, 1989). According to LONCHAMP (1993), a ***software process*** can be defined as "*a set of steps, with sets of related artifacts, human and computerized resources, organizational structures and constraints, intended to produce and maintain the requested software deliverables*".

Software production is a highly creative task and many activities involved in a software process cannot be automated (ARMENISE *et al.*, 1993). According to BENDRAOU and GERVAIS (2007), software processes have some typical characteristics:

- They are complex;
- They are unpredictable since they depend on many people and circumstances;
- Not all activities are supported by automated tools;
- They depend on communication, coordination and cooperation within a predefined framework;
- Their success depends on the coordination of many roles;
- They may take a long time and are subject to changes during this time.

A ***software process lifecycle*** defines the engineering activities performed in this process and organizes them into different stages (FUGGETTA, 2000). Lifecycle activities are called *meta-activities* whereas the lifecycle itself is called *software meta-process* (DERNIAME *et al.*, 1999). Therefore, software processes are formed of two kinds of processes: the *Software Production Process*, which represents the process being actually performed by software developers and tools, and the *Meta-Process*, which consists of lifecycle activities. There are many proposals of software process lifecycle in the literature; the most traditional ones are described by PRESSMAN (2015). However, the lifecycle more related to assumptions of this thesis was proposed by REIS (2003). She considered the following lifecycle activities:

(1) *Provision of technology* provides support technologies for production of software and models (e.g., process modeling languages, reusable process models, and support tools for lifecycle stages).

(2) *Process requirement analysis* identifies requirements for the design of a new process or new requirements for an existing process.

(3) *Process design or modeling* defines the general and specific architecture of the process. In this stage, *Process Modeling Languages* (PMLs) are used to describe processes in *Process Models*.

(4) *Process instantiation* modifies the process specification created in the previous stage (process design) by adding information on time constraints and allocating people and resources for activities defined in the process.

(5) *Process simulation* allows verifying and validating defined processes before executing.

(6) *Process model execution* executes the instantiated process through tools that coordinate the software process in real world. Moreover, the progress of the process is monitored and relevant information is collected along the execution (in other lifecycles, these activities are usually considered in a separated stage named *Process Monitoring*).

(7) *Process evaluation* analyzes quantitative and qualitative information on the performance of process execution. The result of this stage can be used to improve the software process in a next cycle.

The definition of software processes and their representation in model is performed in initial stages of lifecycle. The modeling of software processes can have several purposes. Most important ones remain ensuring process understandability and communication between software developers. In addition, ARMENISE *et al.* (1993) adds the following objectives: process planning, analysis, measurement, configuration, reuse, execution, and improvement.

Regarding ***core process elements***, in essence ***software process models*** should represent *activities* that have to be accomplished to achieve the process objectives (e.g., develop and test a functionality); *roles* of people in the process (e.g., software analyst and project manager); *artifacts* to be created and maintained (e.g., requirement specification documents, implementation documents, and test cases); and *tools* to be used (e.g., CASE tools and IDEs) (FUGGETTA, 2000) (LONCHAMP, 1993).

2.1.2 Software Process Modeling Languages (SPMLs)

Researchers have created a number of languages and modeling formalisms, often called ***Software Process Modeling Languages*** (SPMLs), which make possible to represent in a precise and comprehensive way a number of software process features and perspectives. There are many different types of PMLs, but a detailed discussion of existing approaches can be found in the literature review conducted by GARCÍA-BORGOÑÓN *et al.* (2014).

The authors have provided taxonomy for SPMLs considering the base technology used in their development. They have structured the proposals in three groups according to the results of the review: (1) *Grammar-based SPMLs* focus on programming languages (e.g., graph theory, Petri Nets, and rules); (2) *UML-based SPMLs*; and (3) *Metamodel-based SPMLs or DSLs* that are mainly derived from SPEM (in different versions).

Figure 2 outlines the correspondence among each SPML obtained from the literature review and the aforementioned groups. *Grammar-based SPMLs* are shown with no background color or thick frame, *UML-based SPMLs* are shown with background color and *Metamodel-based SPMLs* are shown with a thick frame. The authors concluded that UML has been considered a suitable base technology, since it constitutes a standard in Software Engineering. However, its weakness is the inability to execute processes due to the lack of formality. Considering the temporal view in Figure 2, new SPMLs trend to use meta-models as base technology, mainly SPEM 2.0 (OMG, 2008) that is an OMG standard for software process modeling. However, the SPEM specification also does not support process execution. It suggests the use of some external formalism for modeling precise process behavior. The authors of the literature review also stated that Grammar-based SPMLs, which focus on process execution and formality, are complex, inflexible and difficult to understand (GARCÍA-BORGOÑÓN *et al.*, 2014).

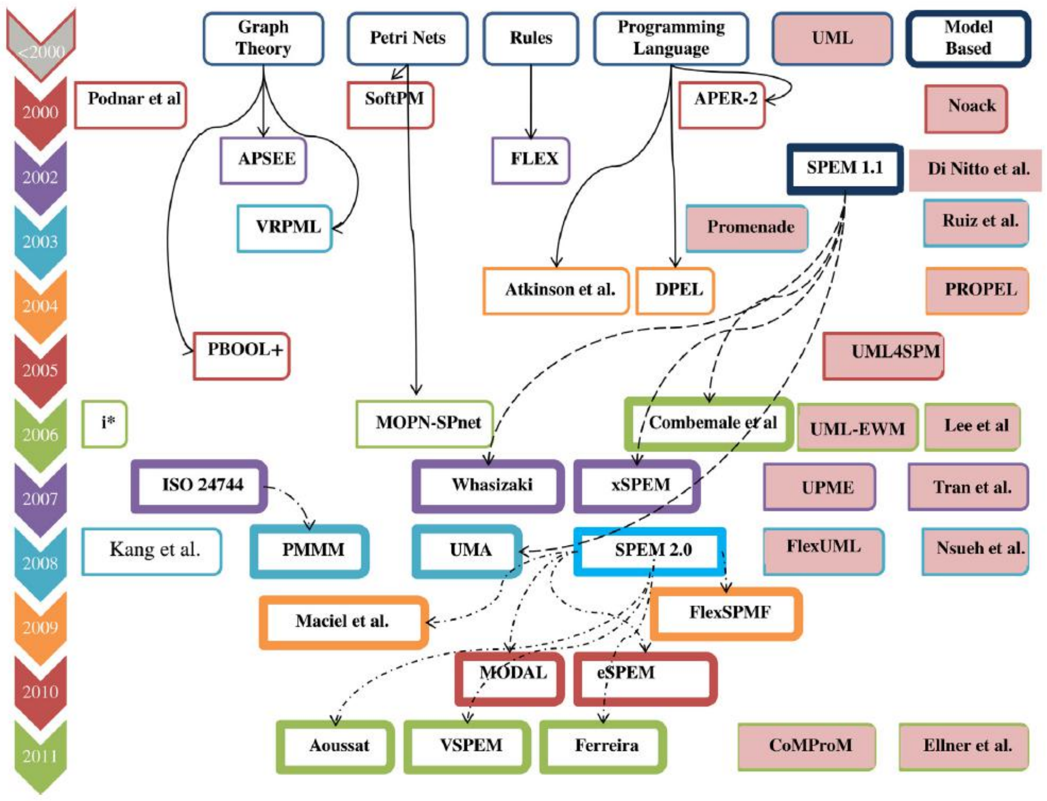


Figure 2. Base technology and relations of existing SPMLs (GARCÍA-BORGOÑÓN *et al.*, 2014)

They finalized the paper claiming "*a proposal that may allow establishing both, a modeling and execution environment, maintaining suitable levels of understandability, would result in an important alternative in this area*" of software process modeling (GARCÍA-BORGOÑÓN *et al.*, 2014).

2.1.3 Core Software Process Elements in SPEM 2.0

SPEM 2.0 is currently the most widespread and popular SPML to represent software processes (KUHRMANN *et al.*, 2013) (RUIZ-RUBE *et al.*, 2013). It is frequently used in academia for exploration and prototyping and comprises reference processes that are applied in practice (KUHRMANN *et al.*, 2013). Thus, in this section we briefly present how core software process elements are represented in the SPEM 2.0 meta-model.

SPEM 2.0 (Software & Systems Process Engineering Meta-Model) (OMG, 2008) is a meta-model based on MOF 2.0 (Meta Object Facility) and a UML2 profile for the specification of software processes. The SPEM meta-model is organized in 7 packages, but we will focus on the *Process Structure* package because it contains the basic structural elements for defining development processes, including *activities*, *workproducts* and *roles*. This package also supports organizing process elements hierarchically and defines the mechanism for tailoring process elements, which will be explained further in this document.

Figure 3 illustrates the main meta-classes found in the *Process Structure* package. The abstract meta-class *ProcessElement* represents any element that is part of a SPEM process whereas *BreakdownElement* is an abstract generalization for any type of Process Element that is part of a breakdown structure. Any concrete subclass of *BreakdownElement* can be “placed inside” an *Activity* (via the *nestedBreakdownElement* association) to become part of a breakdown of Activities. Note that Activities are also Breakdown Elements themselves and therefore can be nested inside other activities. Thus, an *Activity* is a *WorkBreakdownElement* (element that represents work) which defines basic units of work within a process as well as a process itself. Generally, activities are assigned to specific performers represented by *RoleUse* and can rely on input artifacts or produce output artifacts represented by *WorkProductUse.*

SPEM 2.0 contains a precedence control mechanism that supports sequencing activities through relationships of type *WorkSequence*. Such a relationship links two Work Breakdown Elements in which the execution of the first depends on the start or finish of the second. The specific type of *WorkSequence* relationship is defined by the *WorkSequenceKind* enumeration (see Figure 3). However, SPEM does not provide resources for reactive control, i.e., it does not allow the specification of conditions or events in response to which activities are to be executed (OMG, 2008). Therefore, the precedence control mechanism of SPEM is very limited to represent process advanced behavior.

2.2 Business Process Management (BPM)

2.2.1 Concepts and Definitions

Business Process Management **(BPM)** refers to the set of methods, techniques and tools to support the design, enactment, management, analysis and improvement of business processes (VAN DER AALST *et al.*, 2003). In other words, BPM is concerned about all stages of lifecycle of a business process.

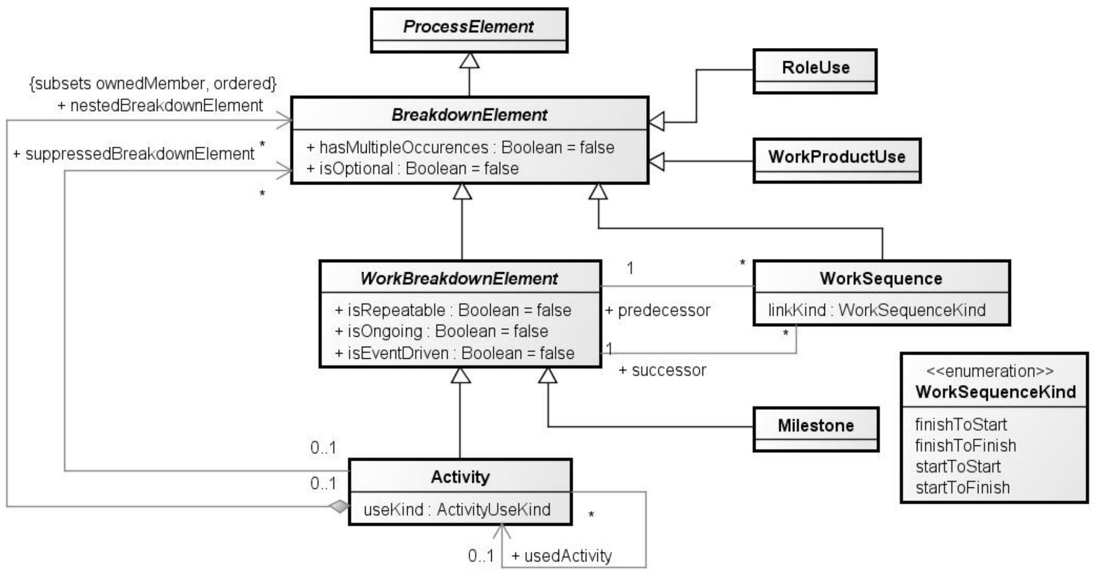

ProcessElement
RoleUse
{subsets ownedMember, ordered}
+ nestedBreakdownElement
BreakdownElement
+ hasMultipleOccurences : Boolean = false
+ isOptional : Boolean = false
+ suppressedBreakdownElement
WorkProductUse
WorkBreakdownElement
+ isRepeatable : Boolean = false
+ isOngoing : Boolean = false
+ isEventDriven : Boolean = false
+ predecessor
+ successor
WorkSequence
linkKind : WorkSequenceKind
<<enumeration>>
WorkSequenceKind
finishToStart
finishToFinish
startToStart
startToFinish
Milestone
Activity
useKind : ActivityUseKind
+ usedActivity
0..1
1
*

Figure 3. *Process Structure* package main meta-classes

DAVENPORT (1993) defined ***business process*** as "*a specific ordering of work activities across time and space, with a beginning and an end, and clearly defined inputs and outputs*". According to WESKE (2007), business process consists of a set of activities that are performed in coordination in an organizational and technical environment to realize a business goal. Typical examples of business processes are Purchasing, Manufacturing, Marketing, and Sales.

Business process activities can be performed by company's employees manually or with help of information systems. There are also activities that can be enacted automatically by information systems, without any human involvement (WESKE, 2007).

Business processes are usually modeled as workflows, i.e., flows of activities. The formal representation of these processes by means of a ***Business Process Modeling Language*** **(BPML)** allows the simulation, execution, monitoring and improvement of an organization's workflow. The output workflow of the business process modeling is known as ***Business Process Diagram***. It uses a network of graphical elements from a BPML to represent flows of activities.

In practice, a range of business to IT-oriented stakeholders create and use Business Process Diagrams for specific purposes (BRANCO, 2014). Each model must be appropriate for its target audience and purpose, having adequate level of detail (BRANCO, 2014). This goal is generally achieved by creating separate models, each one focused on a particular set of stakeholders and purposes. Typically, business processes are modeled in three abstraction levels: *Business* specification, *Technical* specification, and *Executable* specification (BRANCO *et al.*, 2014).

Regarding ***core process elements***, in essence ***business process models*** should contain work *activities*, *roles* of people in the process, *artifacts* to be created and consumed, and support *tools* (BENDRAOU and GERVAIS, 2007).

2.2.2 Business Process Modeling Languages (BPMLs)

Well-known BPMLs can be coarsely divided into (CORREIA, 2014): (1) *semi-formal approaches*, which focus on graphical modeling but also provide technical backgrounds, such as Business Process Model and Notation (BPMN), Event-Driven Process Chain (EPC), and Yet Another Workflow Language (YAWL); and (2) *formal approaches*, which are grounded in different algebraic theories and target simulation and execution of business processes, such as Business Process Execution Language (BPEL), Petri Nets, and Communicating Sequential Processes (CSP).

In this context, BPMN 2.0 (OMG, 2011) (ISO, 2013) has been standing out as a leading technology because:

(1) BPMN is an OMG and ISO standard for business process modeling;

(2) BPMN is one of the most recent BPMLs, so it is grounded on the experience of earlier BPMLs, which ontologically makes it one of the most complete BPMLs (RECKER *et al.*, 2005);

(3) BPMN is nowadays the business process notation most used among BPM practitioners;

(4) It is the BPML with more available BPM tools;

(5) BPMN models can be interpreted and manipulated by both technical and non-technical personnel, reducing the chance of erroneous knowledge transfer (ISO, 2013);

(6) BPM tools can automatically run BPMN models. In fact, tools such as Activiti BPM (ACTIVITI, 2016) are capable of delivering an integrated environment where users can design and run BPMN models;

(7) BPMN also has transformations to other notations, such as Petri Nets (DIJKMAN *et al.*, 2008), which allow the use of tools for formal verification.

In this thesis, we use BPMN 2.0 for representing and adapting behavior of software processes. Our choice of BPMN as modeling language has been especially motivated by its large number of associated tools and techniques as well as its comprehensive set of concepts for modeling of precise process behavior. Thus, the next section presents an overview of BPMN 2.0, its core process elements and how these elements relate to SPEM 2.0 core process elements. Here, SPEM 2.0 is used as reference for comparison because it is the main technology applied currently for representing software processes.

2.3 BPMN (Business Process Model and Notation)

BPMN (Business Process Model and Notation) (OMG, 2011) (ISO, 2013) is a standard meta-model and notation for the representation of business processes. Details on BPMN, such as the entire range of icons used to represent each aspect of a process, including alternative and exception flows, can be found in THOM and IOCHPE ([s.d.]).

Figure 4 shows the main meta-classes of BPMN process elements and their relationships. Such meta-classes are derived from the same abstract super class *BaseElement* at the top right in the figure. In the hierarchical level below this super class there are the meta-classes *FlowElementsContainer* and *FlowElement*, which are related by a composition relationship. The first represents a container or superset of flow elements that forms a BPMN process, whereas the second represents the process flow elements themselves such as tasks, events, gateways, sequence flows, and data objects. Such meta-classes together allow the specification of processes in a hierarchical structure.

Figure 5 shows the most commonly used elements of the BPMN process notation. BPMN meta-classes that instantiate these notation elements are identified in the figure by textual annotations connected to each graphical element.

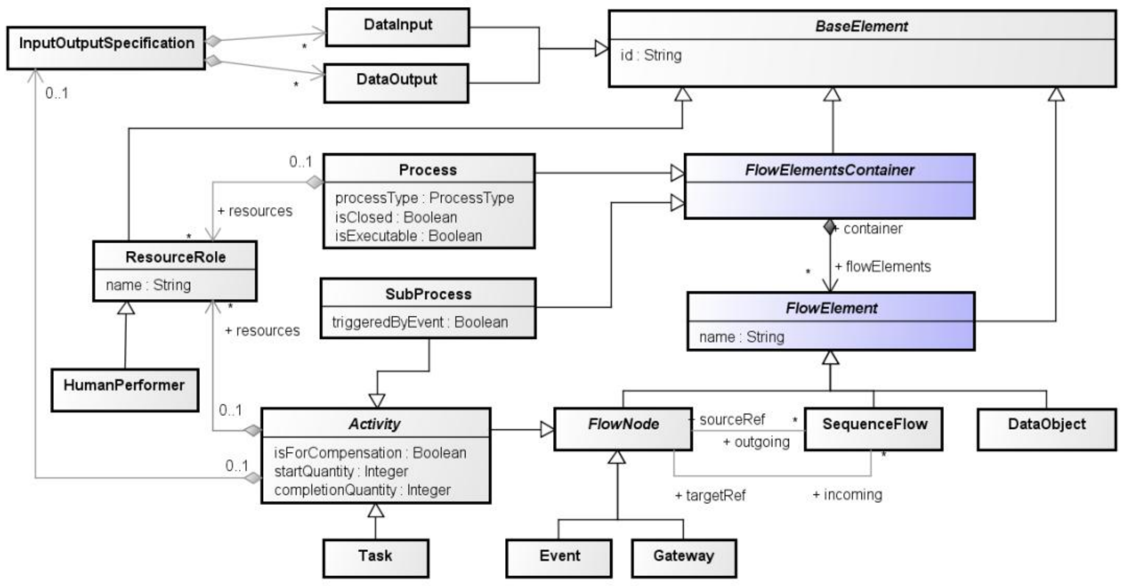


Figure 4. BPMN 2.0 meta-class structure for the core process elements

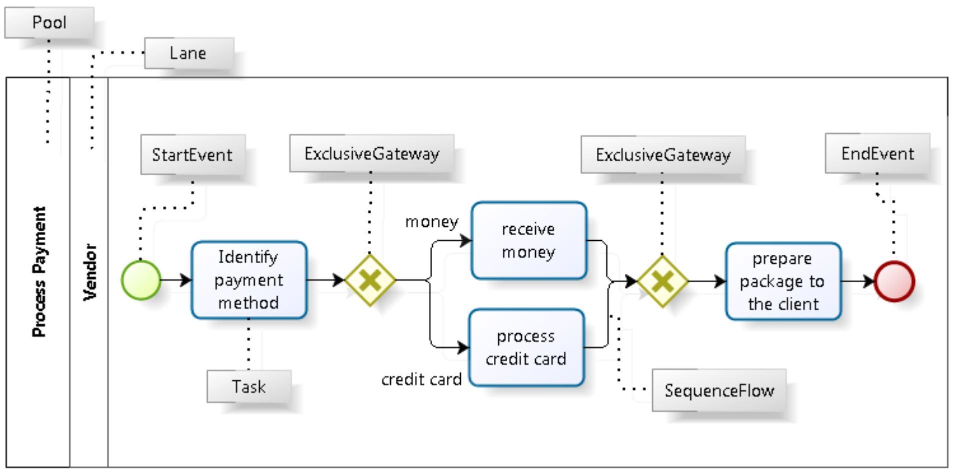


Figure 5. Core process elements of the BPMN notation and their related meta-classes

2.3.1 Correspondence with SPEM Core Process Elements

In order to show that BPMN can be used to specify software process, we first identify correspondences between SPEM core process elements and BPMN elements. Table 1 summarizes the mapping between the meta-models considering the core process elements.

Table 1. Correspondence of SPEM Core Process Elements with BPMN Core Process Elements

SPEM 2.0 Meta-Class	Equivalent BPMN 2.0 Meta-Class(es)
Activity	*Process*
	SubProcess
	Task

RoleUse	*Lane*
	HumanPerformer
WorkProductUse	*DataObject*
	DataInput
	DataOutput

As explained previously, the SPEM meta-class *Activity* is used to represent not only basic units of work (atomic tasks) within a process, but also a process itself. BPMN also has a meta-class called *Activity* (see Figure 4) that represents basic or composed work units within a process, but it cannot represent a whole process. Moreover, this meta-class is abstract and cannot be instantiated. In order to create a process activity instance, its sub-classes *Task* or *SubProcess* must be used. Thus, in BPMN the behavior of the SPEM meta-class *Activity* is provided by three different concrete meta-classes: *Task*, *SubProcess* and *Process*. The first one represents an atomic activity (indivisible) within a process whereas the second represents a set of activities. *Subprocess* defines an embedded process that must be contained within another. Both meta-classes are a type of *FlowElement* and share the same shape in the BPMN notation. However, the meta-class *SubProcess* is also a type of *FlowElementsContainer*. Finally, *Process* is a type of *FlowElementsContainer* used to reference a set of elements that composes a global and reusable process. Unlike the previous meta-classes, *Process* does not have a specific graphical object in the BPMN notation, since it is a set of graphical objects. Nevertheless, it is common to use the notation element *Pool* to represent a meta-class element *Process*, such as shown in the BPMN model in Figure 5 (an element *Pool* is used to represent the process "*Process Payment*").

The SPEM meta-class *RoleUse* can correspond to two different elements in the BPMN meta-model, which are often used together. Visually, the BPMN notation provides only the element *Lane*, a named sub-partition within a *Pool*, to represent specific roles (e.g., see the role *Vendor* in Figure 5). The meta-class *Lane* is used to organize and categorize activities within a process, and often represents roles played by humans. However, BPMN does not specify the exact meaning of lanes, leaving the modeler to choose a specific meaning. On the other hand, when considering the modeling of executable processes that can be managed by a process engine, BPMN specifies the meta-class *HumanPerformer* (a type of *ResourceRole*) to assign people in various roles to activities. *HumanPerformer* supports the definition of a specific individual or group that will perform or be responsible for an activity. However, this meta-class has no visual representation. Thus, in executable BPMN models it is common to find both elements *Lane* and *HumanPerformer* to represent roles whereas in business models only the element *Lane* is used.

In the same way, SPEM *WorkProductUse* corresponds to BPMN meta-classes *DataObject*, *DataInput* and *DataOutput*. *DataObject* represents data used during process execution and whose lifecycle is tied to the lifecycle of the specific process. In contrast, *DataInput* and *DataOutput* are related to activities. *DataInput* defines data that an activity needs in order to execute and *DataOutput* defines data produced by an activity. The relationship between activities and their data is defined in the BPMN meta-model through the meta-class *InputOutputSpecification* (see Figure 4).

2.3.2 BPMN Extension Mechanism

The BPMN 2.0 meta-model, which is specified using the OMG's Meta Object Facility (MOF), has a built-in extension mechanism, represented in Figure 6(a). It supports extension by addition, in that groups of attributes and elements are attached to standard BPMN elements without modifying their original structure. Such an extension mechanism mostly depends on four meta-classes: *ExtensionAttributeDefinition*, *ExtensionAttributeValue*, *ExtensionDefinition*, and *Extension*. The meta-class *ExtensionAttributeDefinition* configures the attributes that can be added to any BPMN element by defining their name and type. The meta-class *ExtensionDefinition* groups these new attributes under a new concept name and can be created independently of any element or BPMN definition. However, in order to use this meta-class to represent an extension, it must be associated with the meta-class *Extension* that connects the *ExtensionDefinition* with the definition of a specific BPMN model (meta-class *Definitions*). The element *Extension* is an attribute (represented as the *extensions* relationship) in the meta-class *Definitions*, which associates an *ExtensionDefinition* with any child element of *BaseElement*. Furthermore, any "extended" BPMN element is associated with the meta-class *ExtensionAttributeValue*, which contains the new attribute value of type *Element*. The new attributes specified in an *ExtensionDefinition* element and bounded to a model definition by an *Extension* element can be used by any BPMN element (being an instance of a subclass of *BaseElement*) within that model. This is because the BPMN extension mechanism does not provide a way to specify which element of the language is being extended.

However, the BPMN specification (OMG, 2011) provides two representations of its elements. Besides the MOF based meta-model describing the language concepts, BPMN also provides a set of XML Schema documents that specify the interchange format for BPMN models. In this way, an interchangeable BPMN model is a set of domain-specific elements represented in XML Schema into a BPMN file (i.e., a file with *bpmn* extension). Since the MOF representation of the BPMN extension mechanism (Figure 6-a) has a limited capability, for example not supporting the definition of type structures for new attributes, the XML Schema extensibility representation (Figure 6-b) has been more widely used. The XML Schema representation of BPMN supports the definition of complex extensions that could be processed by BPMN tools. However, the BPMN specification (OMG, 2011) does not provide any graphical notation for the representation of extensions, meaning that designers need to work on detailed textual implementations.

Moreover, in the specific case of the extension mechanism, the MOF and XML Schema representations of BPMN are not equivalent (see Figure 6). The XML Schema representation does not contain the elements *ExtensionDefinition, ExtensionAttributeDefinition*, and *ExtensionAttributeValue* defined in the MOF representation. When using XML Schema, BPMN extensions are defined using native components of this language (such as type definitions and element declarations) in separate XML Schema documents (with their own namespaces), which are imported by BPMN model documents through the element *TImport*. Moreover, the element *TExtensionElement* can include extension attribute values of any type, even the types defined in other namespaces, since the XML Schema representation of BPMN extensibility specifies a relationship between *TExtensionElement* and *AnyElement*.

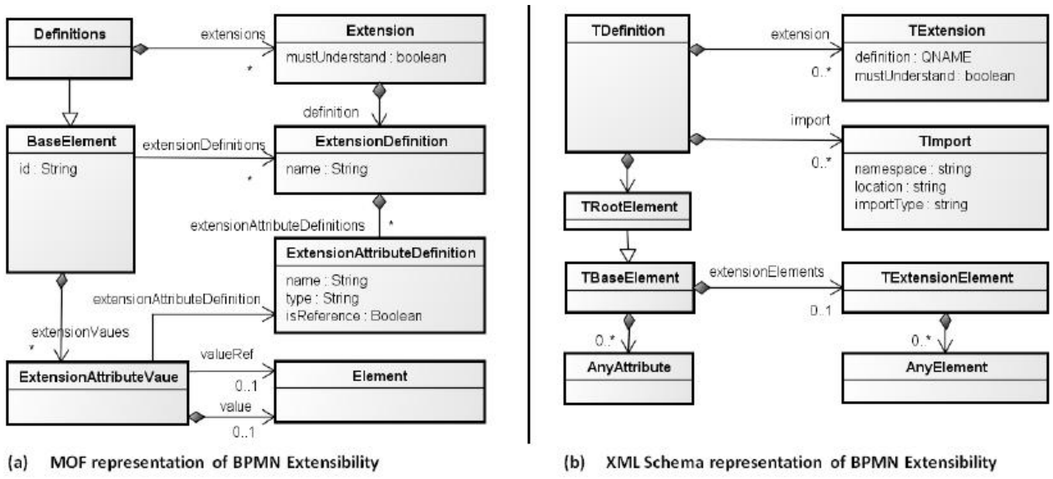


Figure 6: Representations of the BPMN extension mechanism.

2.4 Process Adaptation

This section introduces concepts related to process adaptation in Software Process Engineering (SPE) and Business Process Management (BPM) domains. In the SPE domain, process adaptation is usually called *Software Process Tailoring*.

2.4.1 Software Process Tailoring

Building processes from scratch can be risky and involve high overhead (XU and RAMESH, 2008). In this way, many reference process models have been created based on industry's good practices for software development, such as IEEE/EIA 12207 and the Rational Unified Process (RUP). Such models provide generic processes that capture common activities, information, artifacts and control flows encountered in different processes into the domain of software development.

However, it is unlikely that one of these "off-the shelf" approaches will meet the requirements of a specific project or organization. Therefore, adjustments are necessary to make reference processes suitable for specific environments (PEDREIRA *et al.*, 2007). Such adjustments are often referred to as process tailoring, an activity which has originally been defined as ''*adjusting the definitions and/or particularizing the terms of general description to derive a description applicable to an alternate (less general) environment*'' (GINSBERG and QUINN, 1995).

However, as GINSBERG and QUINN (1995) have already claimed in their seminal document, "*tailoring is not a one-time event, but a repeated, ongoing analysis*" that integrates a process improvement program. Therefore, tailoring can also be defined in terms of modifications that emerge from the monitoring of executions of a process, providing feedback on its definition (FERRATT and MAI, 2010) (SANTOS, OLIVEIRA and ABREU, 2015).

Several software process standards, such as CMMI and ISO/IEC 15504, or reference models, such as RUP (KRUCHTEN, 2004) and OpenUP (EPF, 2010), provide guidelines for process tailoring. These standards and reference models provide a wide range of guidelines, including information regarding activities that should be performed, the order in which to perform the activities, and the type of personnel that should perform them. Moreover, there are also several researches in the literature about tailoring software processes. Although such a topic is not new to researchers, it has recently received increased attention from the software engineering community (MARTÍNEZ-

RUIZ *et al.*, 2012) (KALUS and KUHRMANN, 2013) (ZAKARIA *et al.*, 2015). Here we mention four literature reviews conducted on software process tailoring. PEDREIRA *et al.* (2007) presented a review on software process tailoring, analyzing its current practice in general terms such as approaches, methods, tools, and guidelines. The review by MARTÍNEZ-RUIZ *et al.* (2012) aimed at discovering requirements for a process tailoring notation and mechanisms currently used to support it. KALUS and KUHRMANN (2013) investigated the concrete tailoring criteria that have been reported in the literature, their dependencies and influence on software processes. Finally, ZAKARIA *et al.* (2015) reviewed research works on software process tailoring in order to investigate the state of the art in terms of research activities. Three classifications were produced that group research works into critical success factors, experiences and practices with tailoring reference models and supporting tool.

In summary, there are currently several approaches for tailoring software processes. Such approaches focus on different perspectives related to tailoring such as: formal specification (MARTÍNEZ-RUIZ *et al.*, 2012), criteria (KALUS and KUHRMANN, 2013) (XU and RAMESH, 2008) and guidelines (KRUCHTEN, 2004) (EPF, 2010), standard compliance (YOON *et al.*, 2001) and well-formedness/consistency rules (PEREIRA *et al.*, 2008) (PEREIRA *et al.*, 2007), variability modeling / Software Process Lines (TEIXEIRA, 2016) (OLIVEIRA *et al.*, 2013) (MARTÍNEZ-RUIZ *et al.*, 2008), generative and automatic strategies (HURTADO ALEGRÍA *et al.*, 2011) (ALEIXO *et al.*, 2011), and composition of methods (HENDERSON-SELLERS *et al.*, 2014) (RALYTÉ *et al.*, 2003).

However, there is no current consensus on how to perform process tailoring; what criteria should be applied; or a standard notation.

2.4.2 Business Process Adaptation

In the area of business processes the *tailoring* term is not common. Such a concept is often referenced as *variability management* (ROSA et al., 2017) (AYORA et al., 2015) or *change management* (WANG and ZHAO, 2011) (RAJABI and LEE, 2009) (WEBER *et al.*, 2008). In general, Business Process Modeling Languages (BPMLs) do not provide explicit modeling support for specifying process adaptations. Moreover, when using existing BPM systems, process variants are defined and maintained in separate process models which are only loosely coupled based on naming conventions

(HALLERBACH *et al.*, 2009). Therefore, efforts for maintaining and changing process variants in this context become high since changes have to be separately applied for each variant model. Still, proposed approaches in this area for dealing with process adaptation usually do not address mechanisms for derivation traceability between designed models as it occurs with software process tailoring.

Our approach adds process tailoring support for a specific BPML (i.e., BPMN) and provides mechanisms for derivation traceability between process models created at design-time as it occurs with software process tailoring.

2.5 Well-formedness of BPMN Process Models

The popularity of BPMN is related primarily to the rich expressiveness of its graphical representation. Activities in a process and their technical constraints expressed graphically facilitate the communication about processes among the different involved stakeholders. According to ISO (2013), the notation is understandable by all business users, from business analysts that create the initial models of processes to the technical developers responsible for implementing the technology that will perform those processes, and finally, to the business people who will manage and monitor those processes. Thus, BPMN creates a standardized bridge for the gap between the business process design and process implementation (ISO, 2013).

The BPMN meta-model (OMG, 2011) provides an abstract syntax for the constructs of the language. This is described at the meta-level using a class diagram. The BPMN meta-model can serve as a precise description of the notation and is therefore useful in implementing modeling tools, since it can be used as a basis to define the language syntax. However, it cannot serve as a description of the meaning and usage of BPMN constructs.

BPMN well-formedness rules are described in the standard specification in natural language, scattered over more than 500 pages of the document. As a consequence, given the expressiveness of the language, it is difficult for process modelers to produce well-formed models (CORREIA, 2014). This also makes difficult for tools handling BPMN models to ensure their correctness.

The objective of this thesis is to provide flexible support for adapting BPMN process models while ensuring their correctness regarding the standard specification. In this sense, an important part of this research is to define and implement rules associated to each supported adaptation operation that lead to well-formed tailored BPMN models. As a first step in this direction, it was necessary to capture and analyze all the rules of the BPMN specification in order to identify which rules could be impacted when performing model adaptation actions. Besides the study of the BPMN specification's document, we have also based on the research of CORREIA (2014), which identified such rules from the BPMN specification and produced a summarized list describing them. Thus, we have extracted from this list the subset of well-formedness rules related only to the BPMN's control-flow perspective (our focus) and have selected those rules that refer to BPMN elements usually represented in *conceptual* process models, which are the target of this research. We distinguish between *conceptual process models*, which are intended for communication and analysis, and *executable process models*, which are intended for deployment in an execution engine. Executable models require numerous formalizations and configurations of specific properties that are out of the scope of our proposal. In this way, BPMN well-formedness rules that refer for *Compensation* activities and specific elements such as *Transaction Sub-Process* and *Event Sub-Process* have not been considered in this research, since these elements are typically used in executable process models (BRANCO, 2014). The complete list of BPMN well-formedness rules that our process adaptation solution takes into account to ensure correctness regarding the BPMN specification is presented in Appendix 1. There, one can also find more details about rules of the BPMN standard that we have not considered in our solution.

Such well-formedness rules are actually *static semantic rules*[6] (AABY, 1996) of the BPMN language, which we have based on to derive pre- and post-conditions associated with tailoring operations in our solution in order to assist process designers in moving towards correct specification of tailored process models.

[6] The static semantics defines restrictions on the structure of valid texts that are hard or impossible to express in standard syntactic formalisms, i.e., exclusively through the elements and relationships of meta-model.

CHAPTER III

3. Related Work

3.1. Introduction

This section presents the main research works related to our proposal in the area of process adaptation. *Section 3.2* presents an overview of main techniques for process adaptation management from the SPE and BPM domains. Next, *Section 3.3* presents the different types of adaptation operations that have been proposed in the literature of SPE (Software Process Engineering) and BPM (Business Process Management) for executing changes on process models.

Section 3.4 describes the support for process adaptation provided by two important process meta-models in SPE, SPEM 2.0 (an OMG standard) and V-Modell XT (a German standard), which define specific concepts representing adaptation operations. *Section 3.5* describes BPMN-based adaptation approaches, which are our main related works. In this section, we present the steps, results and analysis of a structured literature review that we have conducted. In this same section, we also compare the selected BPMN-based adaptation approaches through a set of specific criteria. Finally, in *Section 3.6* we present our final remarks.

3.2. Overview of Techniques for Process Adaptation Management

3.2.1. Techniques in SPE

3.2.1.1 Software Process Line

The Software Process Line (SPrL) approach is a concept derived from Software Product Line (CLEMENTS and NORTHROP, 2002). A SPrL is composed of a family of processes that have certain common as well as some variable characteristics

(SUTTON and OSTERWEIL, 1998). Most of the variability mechanisms for SPrL are based on variation points (places in which variability occurs) and variants (concrete elements that are placed at the variation points). WASHIZAKI (2006) and MARTÍNEZ-RUIZ *et al.* (2008) proposed new mechanisms to support SPEM 2.0 tailoring that allow the establishment of a part of the process specification as common to all processes and to limit variations by specifying which parts can vary (variation points) and in which range of values (variants). Similarly, OLIVEIRA *et al.* (2013) also proposed to extend SPEM for representing variability in the context of SPrL. They take into consideration the SMarty approach for variability management and, as an important differentiator the authors also provide guidelines that suggest how to identify variances in a SPrL.

BARRETO (2011) proposed to define a SPrL as a process architecture, similar to work flows, which can contain process components or activities. He defines a "skeleton" of workflow that a process must follow, containing its main elements and relationships. However, variations on this pre-determined process flow are very limited. Such variations are supported by defining if flow elements (edges) of the model are optional or mandatory.

MAGDALENO (2013) proposed an approach to assist project managers in selecting and combining process components to derive a project-specific process. To this end, a context-based SPrL was defined. The approach requires that information about project's context (e.g., organizational structure, size, complexity and time of the project, experience of the team, etc.) is previously identified and recorded to support decisions of a project manager.

TEIXEIRA (2016) proposed to develop a component-based SPrL with variability management in multiple perspectives, including the behavioral perspective of software process. However, such proposal also deals with limited variations on the process' flow.

In summary, Software Process Line approaches are only suitable in well-defined domains where there are few and known alterations. Adaptations to generate specific processes are limited to addition and removal operations of process elements defined in advance. In other words, a SPrL approach facilitates planned reuse, while classic tailoring must integrate techniques that can react to unanticipated variability in the process model (ARMBRUST *et al.*, 2009). In the case of software process tailoring, variations limited to a given set of values and specific points can prevent specific needs of an organization or project from being properly addressed, especially because variations in software processes are difficult to predict. In addition, Software Process Line-based approaches generally do not deal with sequence flows (ordering) and flow controls such as fork and join, but restrict their focus to which concrete elements will be reused. In any case, even if sequencing is considered by some of the SPrL approaches, it is addressed in a very limited way.

In general, SPrL approaches have some main limitations:

(1) They support limited variation mechanisms (basically: optional, mandatory, and alternative) (MARTÍNEZ-RUIZ *et al.*, 2012);
(2) They are appropriate to requirements specification of process, but not to activities sequencing (ZAVE, 1993);
(3) In most cases, support for transition between abstraction levels of models is lacking (BEUCHE *et al.*, 2004).

Conversely, our approach for BPMN-based process tailoring supports unanticipated variances and is able to address specific needs of an organization or project. Moreover, our approach also supports advanced adaptations on sequencing and control flow as well as managed process derivation.

3.2.1.2 Situational Method Engineering (SME)

The concept of process tailoring is also related to Situational Method Engineering (SME) (HENDERSON-SELLERS *et al.*, 2014). In the method engineering community, this term has been used to advocate that methods for the development of information systems must be adapted to the specific characteristics of a particular situation. In other words, SME focuses on the definition of organization-specific or project-specific methods.

Traditionally, SME promotes the definition of a method by selecting and assembling reusable method pieces such as core process elements, that have been already created and stored in a repository or *methodbase* (HENDERSON-SELLERS and RALYTÉ, 2010). The kinds of method elements as well as the high-level relationships that are possible between these elements are given by an underlying meta-model. In the Information System literature a method is considered as a set of pieces, also called fragments or components, that can have a very different granularity and can describe the product or the process of a method. The method components can comprise a single activity or construct but can also contain a complete method. Thus, an SME project can start from a set of method components which must be assembled as well as from an existing method (reference model or pattern) which has to be adapted (RALYTÉ *et al.*, 2003). According to BECKER *et al.* (2007), the first strategy (and the more usual) is supported by aggregation mechanisms whereas the second one is mainly applied with specialization mechanisms.

Comparing SME contributions to our proposal, the BPMN*t* approach is especially related to the ones that employ specialization operations on existing methods as RALYTÉ *et al.* (2003) and HENNINGER *et al.* (2002). However, BPMN*t* does not require a repository of reusable components. Therefore, novel additions to an existing process model are not limited to predefined components. Moreover, we have not identified any SME approach that preserves traces of the adaptations performed on the base method into the derived method. Such a feature, provided by our approach, allows us to identify easily the base process model that originated the tailored process model at hand and the changes made on it. In addition, SME approaches, such as PrL approaches, focus on insertion and removal of elements that make up the process (e.g., methods or activities), but they do not deal with managed adaptations on the process behavior (control flow).

3.2.1.3 Context-based Management

Recently, some researches related to process adaptation have proposed to use context information for determining how processes should be adapted (POPP and KAINDL, 2015) (NUNES, 2014) (HURTADO ALEGRÍA *et al.*, 2011) (HALLERBACH *et al.*, 2009). Most of these proposals use context information to automatically derive variant processes.

However, several researches still try to formalize the context definition. BAZIRE and BRÉZILLON (2005) cataloged more than 150 definitions and realized that such definitions vary according to the considered domain. A definition more widely used claims that context is "*any information that can be used to characterize the situation of entities (i.e., a person, place, or object) that are considered relevant to the interaction between a user and an application*" (DEY *et al.*, 2001).

Most of context-based approaches are from the BPM area (e.g., POPP and KAINDL, 2015; NUNES, 2014; DÖHRING and ZIMMERMANN, 2011). In fact, tailoring criteria seem to be more easily understood to business processes. For example, NUNES (2014) proposed an approach for dynamic process adaptation (i.e., at runtime) that was validated on business processes from specific domains.

However, in the SPE domain it still lacks understanding regarding context and criteria for software process tailoring (KUHRMANN, 2014) (KALUS and KUHRMANN, 2013). According to KUHRMANN (2014), selecting a tailoring criterion depends on many context variables (e.g., size and complexity of the software to be developed, time constraints, personnel availability, and team distribution pattern), but the way that different variables relate to each other is still poorly understood (KALUS and KUHRMANN, 2013).

In summary, context-based approaches tend to produce inaccurate results and provide low flexibility for process adaptation. Moreover, they require big effort before the tailoring to capture current context information and relationships. Due to these limitations, the BPMN*t* approach does not use context information. We have opted by proving a flexible solution, based on tailoring operations, in which a process manager can adapt processes according to his/her current needs.

3.2.1.4 Summary of Approaches in SPE

MARTÍNEZ-RUIZ *et al.* (2012) have conducted a systematic literature review (SLR) with the aim of analyzing existing mechanisms that support software process tailoring. From the review, they have found that only 17 of 32 proposals dealing with software process tailoring consider adaptations on the control flow of the process (i.e., its workflow), as shows the graph in Figure 7.

As we have already mentioned in previous sections, proposals addressing adaptations with respect to the behavioral perspective of software processes use limited possibilities of variation, such as defining sequence flows between tasks as optional or mandatory. Other approaches use components as mechanism to vary the flow of the process. However, it is important to highlight that these proposals do not define mechanisms to trace variations in the process behavior.

Conversely, our proposal intents to provide advanced mechanisms to adapt the control flow of processes, while tracking all the performed changes.

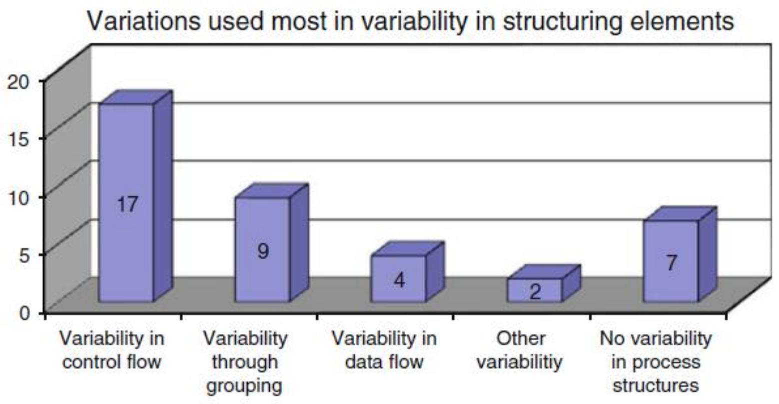


Figure 7. Adaptations on structuring elements in SPE (MARTÍNEZ-RUIZ *et al.*, 2012)

3.2.2. Techniques in BPM

3.2.2.1 Variability Management

Variability aims at capturing a family of process model variants in a way that individual variants can be derived via transformations, for example, adding or removing fragments (ROSA *et al.*, 2017). Therefore, variability modeling *encapsulates*, in some way, *all* customization decisions between process variants. These decisions may result in the removal or addition of behavior to a base process model. In this sense, ROSA *et al.* (2017) distinguish between two approaches to variability modeling: by restriction and by extension.

Variability by restriction or *configuration* starts with a base process model that contains the behavior of *all* process variants. This model is often called *configurable process model*, in which adaptation is achieved by restricting the behavior of the model through configuration. For example, activities may be skipped or blocked during the configuration. In this case, one can think about the configurable process model as the union of all process variants. This technique has been realized based on different methods, including hiding and blocking (e.g., SCHUNSELAAR *et al.*, 2012; YOUSFI *et al.*, 2016), configurable nodes (e.g., GOTTSCHALK *et al.*, 2007), annotations in models (e.g., FRECE and JURIC, 2012; SCHNIEDERS and PUHLMANN, 2006) and meta-model extensions (e.g., MOON *et al.*, 2008).

On the other hand, in the *variability by extension* the base process model does not contain all possible behavior. Instead, it represents the most common behavior or that one that is shared by most process variants (ROSA *et al.*, 2017). During the adaptation, this model is then extended with additional behavior to meet a particular situation. For example, one may need to insert new activities in order to create a dedicated variant. In this case, one can think on the base process model as the intersection of all process variants under consideration. For representing the additional behavior (variable process fragments), this technique can use, for example, process components (PASCALAU *et al.*, 2011), or a set of pre-specified change operations (e.g., DÖHRING and ZIMMERMANN, 2011; HALLERBACH *et al.*, 2009). In turn, the rules for adapting the base model may rely on methods such as business rules.

Our proposal *may* be used for modeling customization decisions that produce a particular process variant, but it is not a variability approach since it does not encapsulate all customization decisions that can take place when deriving variants in the context of process families.

3.2.2.2 Change Management

Approaches to (flexible) *change management* (RAJABI and LEE, 2009) (WEBER *et al.*, 2008) are not concerned with maintaining multiple process models that together form a family of processes. Instead, these approaches focus on unique adaptations of a particular process model.

From some of the most prominent works on *change management*, we highlight ADEPT and YAWL. The ADEPT project (REICHERT *et al.*, 2005) started in 1995 and is the origin of the AristaFlow BPM system, which supports users in modifying the structure of processes at *runtime*. It allows process participants to apply changes to all running instances (with an appropriate migration strategy). On the other hand, the YAWL environment (VAN DER AALST and TER HOFSTEDE, 2005) has been extended with Worklets Services, which enable planned changes for process instances as well as ad-hoc changes (unexpected) at runtime. However, both projects use their own process modeling languages and are intended to manage adaptations at *runtime*.

Although these systems are able to manage an extensive set of process adaptation operations, their purpose differs from our approach, which intends to support and track adaptations between process models at *design-time*, such as solutions for software process tailoring. While our approach addresses issues of traceability between process *types* (schemas), change management solutions such as ADEPT and YAWL address traceability between process *types* (schemas) and their instances.

3.3. Adaptation Operations from the Literature

In this section, we present the different types of adaptation operations that have been proposed in the SPE (Software Process Engineering) and BPM (Business Process Management) literature for performing changes on process models.

3.3.1 Adaptation Operations in SPE

From a systematic literature review, MARTÍNEZ-RUIZ *et al.* (2012) identified two types of variations from proposals in the literature of SPE: Individual modification of process elements, named *direct operations,* and simultaneous variation of several process elements, named *indirect operations*.

The specific types of operations that compose these two categories are described in Table 2, while Figure 8 presents the number of proposals (of 32) that apply each specific operation. Observing Figure 8, one can realize that most of proposals (20 of 32) does not address indirect operations, but applies only direct operations on single elements. Among these proposals, nine of them deal with direct variations on relationships between elements, which include sequence flows and data flows. However, an effective approach to deal with adaptations regarding the process behavior (i.e., its execution flow) should not perform modifications directly on flow elements, but to vary them indirectly due to changes on core process elements (e.g., activities, roles and artifacts) (WEBER *et al.*, 2008) (CASATI, 1998).

In this sense, other proposals use *patterns* to vary the control flow of software processes. In general, *process patterns* provide some additional knowledge to help process engineers in defining processes. However, as stated by BARRETO (2011), it is often difficult to distinguish the concept of *pattern* from other concepts as *components*, *frameworks, templates* or *process families*, especially when patterns involve groups of activities or tasks. In fact, analyzing works related to process patterns from the literature of SPE, we realized that they are generally used in a way very similar to *components*, containing domain- or context-specific solutions.

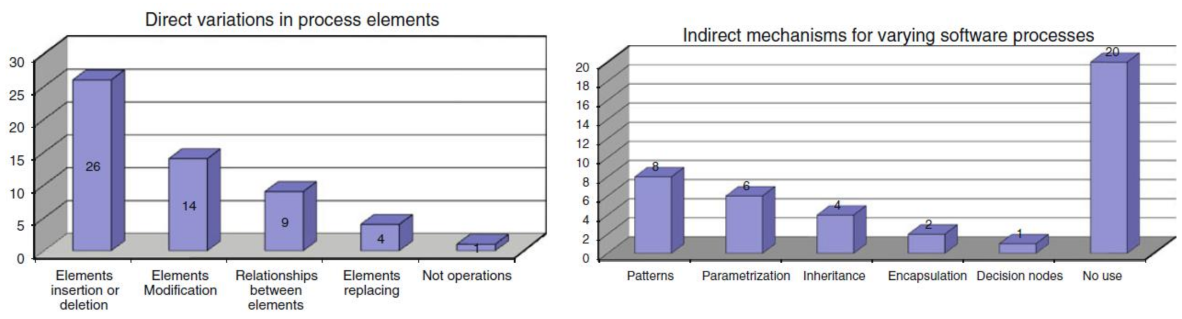


Figure 8. Usage of *direct* (left) and *indirect* (right) tailoring operations in SPE (MARTÍNEZ-RUIZ *et al.*, 2012)

Table 2. Tailoring Operations in SPE (MARTÍNEZ-RUIZ *et al.*, 2012)

Direct Operations	
Insertion	Adds process elements.
Deletion	Removes process elements.
Modification	Changes properties of elements instead of replacing them.
Replacement	A combination of deletion and insertion operations.
Relationships between elements	Adaptation of relationships and constraints between elements.
Indirect Operations	
Patterns	Well-defined variations to be applied when certain requirements are satisfied.
Parameterization	Assignment of a value to certain previously defined parameters of the process when it is going to be tailored.
Inheritance	Adaptation of a parent process by defining child processes that extend the properties of the parent, according to each particular context.
Encapsulation	Groups of activities that are dealt with jointly for the tailoring of the process.
Decision nodes	Decision nodes are composed of conditions, which are used to change the flows between activities (control flows) or to change usage of products (product flow).

Comparing the presented results by MARTÍNEZ-RUIZ *et al.* (2012) with our proposal, we intend to address many of the types of process variations reported by the authors, through basic and high-level tailoring operations. All operations of the category *direct operations* (see left part of Figure 8) will be covered by our set of *basic operations*. Regarding *indirect operations* (see the right part of Figure 8), our high-level operations are similar to concepts as *patterns* and *parameterization*. They are a set of well-defined variations (i.e., a set of basic or direct operations), but do not contain an "additional knowledge" by default as the *patterns* found in the literature of SPE. Process elements handled by high-level operations are assigned to them, through operation parameter, when the process is going to be tailored (such as the concept of *parameterization* specifies). This way, our set of tailoring operations remains independent of domain or context.

3.3.2 Adaptation Operations in BPM

3.3.2.1 Adaptation Patterns

The most widespread and complete set of adaptation operations in BPM has been proposed by WEBER *et al.* (2008) as *control-flow adaptation patterns* (see descriptions in Table 3). The authors use the concept of *high-level operation* to represent a set of well-defined operations that aims at reducing complexity, like design patterns in software engineering, and ensuring the model correctness (WEBER *et al.*, 2008).

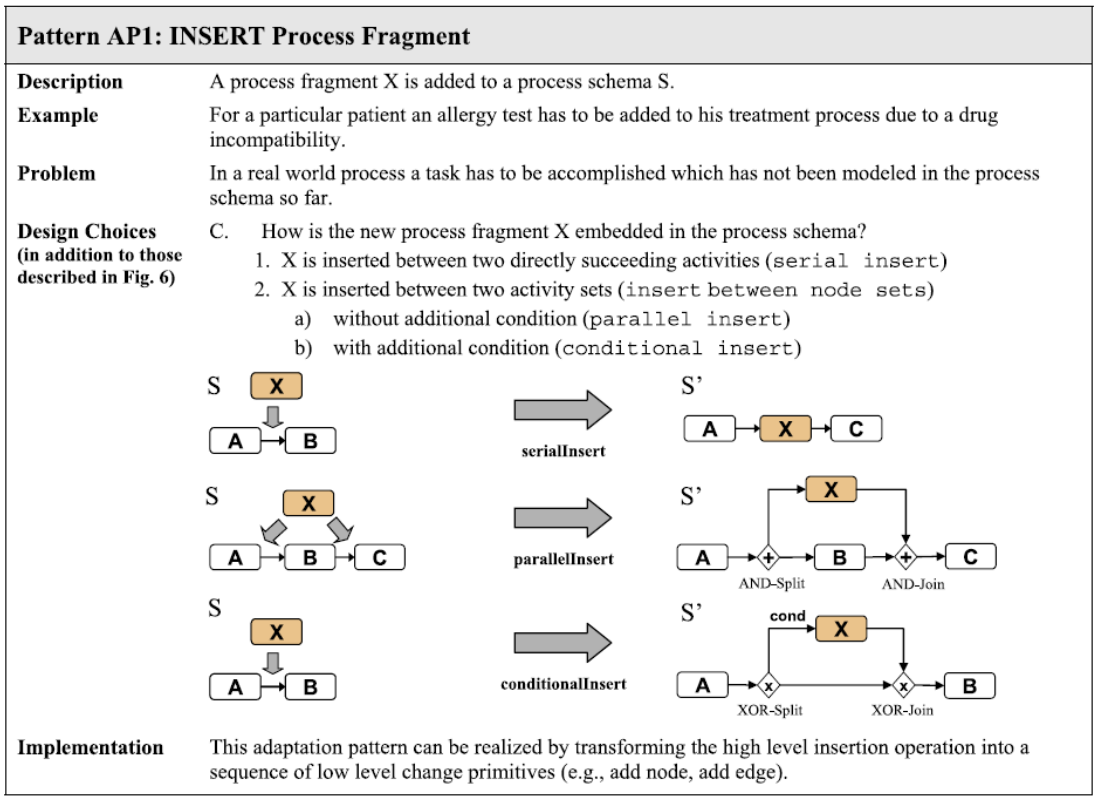
An *adaptation pattern* comprises exactly one *high-level operation*. Its application to a given process model preserves soundness of this process if certain pre- and post-conditions are met. Figure 9 exemplifies the definition of the adaptation pattern *Insert Process Fragment*. In the context of these patterns, a *process fragment* can represent an atomic activity, a subprocess or a subgraph.

Although process adaptations can be performed based on low-level change primitives, these primitives are not considered as real adaptation patterns due to their lack of abstraction. According to the authors, high-level patterns have been identified by analyzing a large collection of business process models from two domains.

Table 3. Control-Flow Adaptation Patterns in BPM (WEBER *et al.*, 2008)

Adaptation Patterns	
Insert Process Fragment	Adds a process fragment to an existing process. The fragment can be added between two directly succeeding activities (serial insert) or between two sets of activities (meting certain conditions). In the latter case, the insertion of a process fragment can occur in parallel to another one (parallel insert) or as a new conditional branch, if an execution condition is provided to the operation (conditional insert). This operation is exemplified in Figure 9.
Delete Process Fragment	Removes a process fragment from an existing process. Afterwards, pos-conditions ensure the correct reconfiguration of sequence flows of the workflow.
Move Process Fragment	Allows shifting a process fragment from its current position to a new one. Like for the *Insert Process Fragment* pattern, an additional design choice specifies the way the fragment can be re-embedded in the process, i.e., in serial, parallel or conditional way.
Replace Process Fragment	Replaces a process fragment by a new one.
Swap Process Fragment	Two existing process fragments are swapped (in their workflow positions) in the process model.
Copy Process Fragment	Allows to copy a process fragment. In contrast to the pattern *Move Process Fragment,* the respective fragment is not removed from its initial position.
Extract Subprocess	Allows to extract an existing process fragment from a process model and to encapsulate it in a separate subprocess. This pattern can be used to add a hierarchical level in order to simplify a process model or to hide information from process participants.

Inline Subprocess	Allows to inline a sub-process schema into the parent process, and consequently to flatten the hierarchy of the overall process. This can be useful in case a process model is divided into too many hierarchical levels or for improving its structure.
Parallelize process fragments	Enables the parallelization of process fragments that were confined to be executed in sequence in the original process model.
Add control dependency	A control edge (e.g., for synchronizing the execution order of two parallel activities) is added to the process model. As opposed to the low-level change primitive *add edge*, the added control dependency must not violate model soundness (e.g., no deadlock causing cycles).
Remove control dependency	A control dependency and its attributes can be removed from a process model. Similar considerations as for the previous patter can be made.
Update condition	Allows to update transition conditions in a process model.



Pattern AP1: INSERT Process Fragment

Description	A process fragment X is added to a process schema S.
Example	For a particular patient an allergy test has to be added to his treatment process due to a drug incompatibility.
Problem	In a real world process a task has to be accomplished which has not been modeled in the process schema so far.
Design Choices (in addition to those described in Fig. 6)	C. How is the new process fragment X embedded in the process schema? 1. X is inserted between two directly succeeding activities (`serial insert`) 2. X is inserted between two activity sets (`insert between node sets`) a) without additional condition (`parallel insert`) b) with additional condition (`conditional insert`)
Implementation	This adaptation pattern can be realized by transforming the high level insertion operation into a sequence of low level change primitives (e.g., add node, add edge).

Figure 9. Definition of an adaptation pattern (WEBER et al., 2008)

In this thesis, we support high-level operations from the *adaptation patterns* presented above for adapting the control-flow of BPMN process models. Our aim was to provide a meta-model representation for process tailoring and associated mechanisms (tool support) based on operations of low and high level, which allow process managers to specify managed process derivations, i.e., with tracked adaptations. This way, it is important to provide a set of high-level adaptation operations compliant to particularities of BPMN, i.e., operations that support users in deriving new processes while ensuring their correctness.

3.3.2.2 Refinement Patterns

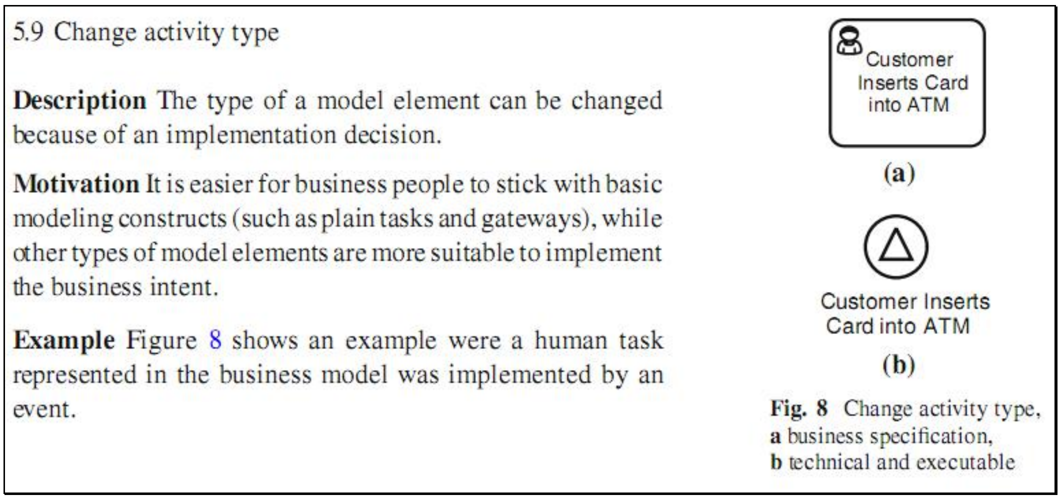
In the context of BPM, it is common a business process model to be represented in different levels of abstraction, including a general business specification (which focuses on the concepts and rules relevant to business analysts), a technical-level specification (which details tasks and flows as well as adds technical exceptions), and an executable specification. Typically, these representations of the same process in different levels of abstraction are created and maintained in different process models to effectively separate concerns and to convey the right information (with proper level of abstraction) to different groups of stakeholders (BRANCO *et al.*, 2014). The derivation of more technical process models from business-level process models is referred in the literature of the area as *Business-IT* refinement (BRANCO *et al.*, 2014).

In this context, the research conducted by BRANCO *et al.* (2014) and BRANCO (2014) has identified different *refinement patterns* applied when producing technical process models from business-level process models. The research was based on the analysis of 74 models in 5 BPM projects in the banking domain and more than 1,000 changes made on these models. However, unlike adaptation patterns, these patterns do not represent necessarily a high-level change operation and neither have any concern related to model correctness. Table 4 describes the entire set of refinement patterns, whereas Figure 10 exemplifies the pattern *Change activity type*.

Table 4. Business-IT Refinement Patterns in BPM (BRANCO *et al.*, 2014)

Refinement Patterns	
Add properties	Parameters for grounding the executable model on top of the underlying IT infrastructure are added during the implementation. Such properties do not change the workflow and may be tool or platform-specific.
Add script task	Script tasks are used to initialize variables and implement business rules and non-functional requirements that access or transform business objects data, e.g., logging steps of the workflow.
Add protocol task	An asynchronous service can be implemented by a connection-less request or reply protocol.
Add boundary event	Boundary events are used to divert the normal flow under special conditions, for example, because of a particular action performed by the operator on a human task.
Add technical exception flow	Technical exception flows are included to divert the flow in case of technical exceptions, such as an unavailable service or a permission denied. This operation is exemplified in Figure 10.
Change activity name	The name of a business activity can be changed to facilitate the identification of an IT service that has a similar but different name.
Change activity type	The type of a model element can be changed because of an implementation decision.

Split task into block	A single business task can be implemented by a combination of service tasks.
Split workflow	The specification workflow can be split into smaller workflows that should be orchestrated by a main flow.
Suppress specification activity	Business elements can be suppressed during the implementation.


Customer Inserts Card into ATM
(a)
Customer Inserts Card into ATM
(b)

5.9 Change activity type

Description The type of a model element can be changed because of an implementation decision.

Motivation It is easier for business people to stick with basic modeling constructs (such as plain tasks and gateways), while other types of model elements are more suitable to implement the business intent.

Example Figure 8 shows an example were a human task represented in the business model was implemented by an event.

Fig. 8 Change activity type, **a** business specification, **b** technical and executable

Figure 10. Definition of the refinement pattern *Change Activity Type* (BRANCO *et al.*, 2014)

In this thesis, we support refinement operations for conceptual BPMN models. That is, we are only interested in operations applied to derive a technical-level process model (that is still conceptual) from a business-level process model (that is also conceptual). In other words, we are not interested on refinement operations that are applied exclusively for deriving executable models and many of refinement patterns described in Table 4 refer to this type of refinement. Therefore, in our approach we derive refinement operations from the presented patterns for dealing only with refinements required between business-level models and technical-level models that are both conceptual.

3.4. Support for Process Adaptation in Standard Meta-models

Although there are different meta-models for specifying process, we focus on SPEM 2.0 (OMG, 2008) and V-Modell XT (TERNITÉ and KUHRMANN, 2009) because they explicitly define support for process tailoring through specific concepts representing adaptation operations. Process assets that are built on these meta-models can extend or modify other process assets. However, while SPEM supports generic tailoring operations (e.g., *extends* and *replaces*), V-Modell XT supports a set of *typed* operations (e.g., *rename work product* and *change role name*) and defines a specific concept for process variant (TERNITÉ, 2010).

SPEM 2.0 is a widely used OMG standard for specifying software process and V-Modell XT is the standard process framework for IT development projects in Germany's government agencies. The following subsections detail how process tailoring is supported by these meta-models.

3.4.1 Process Adaptation in SPEM 2.0

SPEM 2.0 (OMG, 2008) is an OMG standard for software process modeling that explicitly support concepts for tailoring representation. Figure 11 illustrates the main meta-classes of the SPEM's *Process Structure* package, which supports organizing process elements hierarchically and defines the mechanism for tailoring process elements.

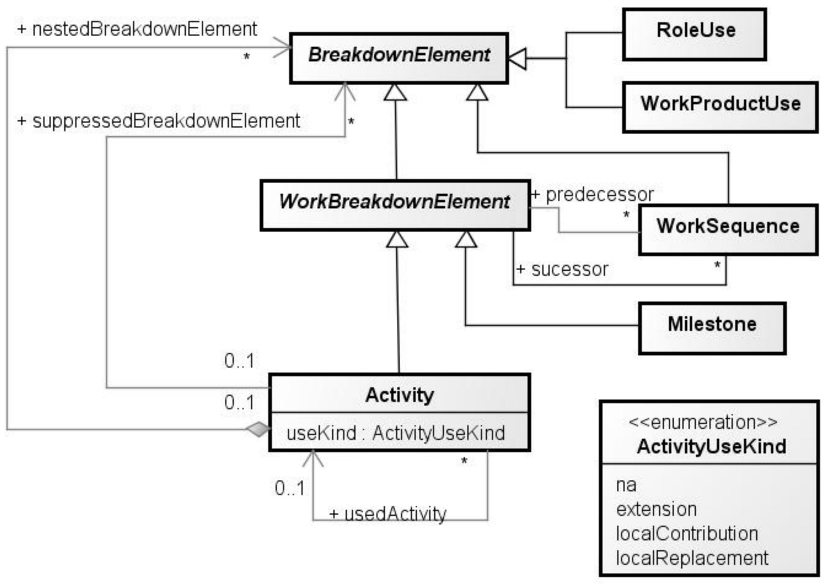


Figure 11. *Process Structure* package main meta-classes.

Tailoring operations in SPEM are implemented through relationships between two classes of type *Activity* (Figure 11). Activities are at the center of SPEM-based processes as they can relate to each other to define sequences as well as specify the roles in charge of their execution and the work products (artifacts) that are manipulated. Some other *Activity* properties in Figure 11 relevant to tailoring are:

- *nestedBreakdownElement*: Represents the relationship between nesting activities and allows the representation of a hierarchical structure where an Activity can represent an entire process, along with its nested elements.

- *usedActivity*: Represents an association between two classes of type *Activity*, where the current activity (source) extends the linked activity (target). The semantics of such association are conveyed by the attribute *useKind*.
- *useKind*: Defines the type of tailoring operation carried out between elements related through *usedActivity*. The enumeration *ActivityUseKind* defines the tailoring semantics:

  na: Defines the default value for activities that are not reused, i.e., that have no association of type *usedActivity*.

  extension: Indicates that an activity (and its associated elements) extends another activity**.** In other words, this relationship makes a copy of the activity being pointed at and associates it with the source activity (such as an inheritance mechanism). The relationships *localContribution, localReplacement* and *suppressedBreakdownElement* must be used in conjunction with *extension*.

  localContribution: Indicates that an activity adds process elements to another inherited by the *extension* relationship.

  localReplacement: Indicates that an activity replaces another inherited by the *extension* relationship.
- *supressedBreakdownElement*: This relationship allows process elements to be removed from an inherited activity. After an activity A extends an activity B, it is possible to remove elements (and not only activities) of A using this relationship.

Figure 12 illustrates the tailoring mechanism using instances of the meta-class *Activity* with their relationships and attribute values. In this case, composition relations represent the *nestedBreakdownElement* relationship. The example has two SPEM-based processes represented as hierarchies. *Process 2* (in the middle) adapts *Process 1* (on the far left) by performing a few adjustments. The first activity has the highest hierarchical level, being itself *Process 2*. It is related to the *Process 1* top activity through a *usedActivity* relationship with its attribute *useKind* set to *extension*. This allows the reuse of the whole structure that is defined through the relationship *nestedBreakdownElement*. In this example, however, the structure is modified through *suppression*, *local contribution* and *local replacement* operations.

The first change is the deletion of *Activity 1.1*. For this purpose, *Activity 2.1* points to *Activity 1.1* through the relationship *suppressedBreakdownElement*. As a result, the final representation for *Process 2* removes *Activity 1.1* from the process hierarchy.

Activity 2.2 is directly related to *Activity 1.2* through the relationship *usedActivity*. In this case, the attribute *useKind* is set to *localContribution*, which causes the addition of a child activity to *Activity 1.2* that remains in the final process. Finally, *Activity 2.3* relates to *Activity 1.3* with the attribute *useKind* set to *localReplacement*. In this case, *Activity 2.3* and its children completely overlap *Activity 1.3*, thus redefining the reused process. *Activity 1.4* is reused in Process 2 (through the *extension* relationship) such as defined in Process 1.

The hierarchy on the far right of Figure 12 presents the final Process 2, obtained from tailoring Process 1. It is possible to observe that the SPEM tailoring mechanism allows the representation and interpretation of process adaptations through a set of relationships that can be clearly specified when modeling software processes.

Our first specific goal was to include in BPMN a basic tailoring support similar to the one provided by SPEM in order to allow adaptations on the behavior of software processes. Thus, our approach extends the BPMN meta-model by providing additional concepts for representing the tailoring operations *extension*, *contribution*, *replacement* and *suppression*. However, since SPEM operations do not take into account the behavioral perspective of process, i.e. its execution flow, we have adapted these operations to application in workflow models. That is, our SPEM-based operations (presented in Chapter 5) were incremented with well-formedness rules to produce valid process models after tailoring. Such rules must consider the current state of the process when tailoring operations are created and processed.

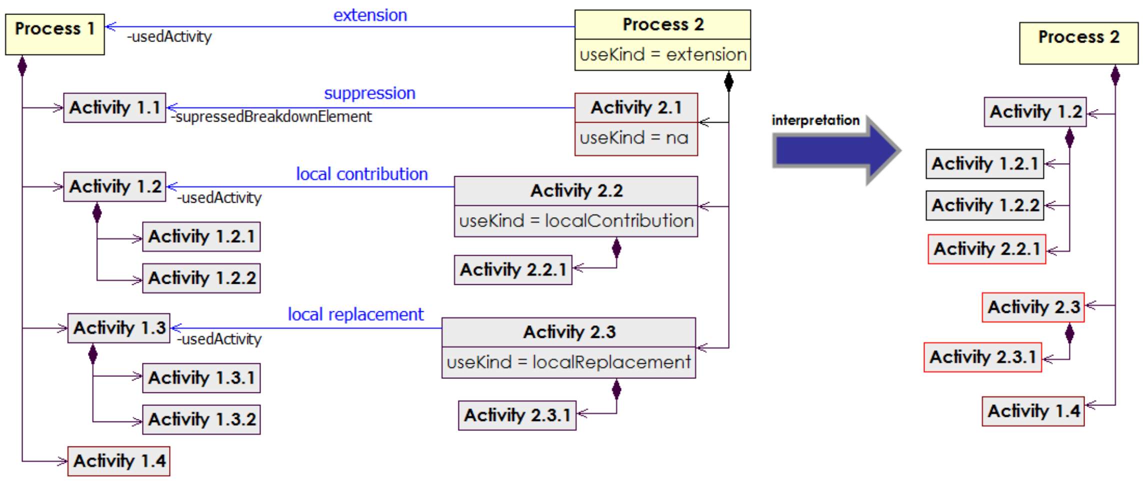


Figure 12. SPEM tailoring representation (left) and interpretation (right) (adapted from OMG, 2008)

3.4.2 Process Adaptation in V-Modell XT

The V-Modell XT meta-model is designed to support hierarchically organized process variants (TERNITÉ, 2010). A new process variant is created referring to a reference model (base process) on which the variant is based. A variant can be regarded as an extension applied to a reference model that refers to it extending or modifying any reference model element. Afterwards, a merge tool creates an integrated process from the variant and the reference model. New process assets introduced by the variant will be integrated with the reference model. For example, exclusions will be deleted and variability operations will be executed.

The types of variability operations are defined in meta-models representing architectures of Software Process Line (SPrL) that control supported variations (TERNITÉ, 2009). That is, each Process Line defined from the V-Modell XT meta-model can define its own variability operations, which will be applied to variant models. The types of operations supported can be classified as *positive* (addition of elements or relations), *negative* (removal of elements or relations), *extension* (elements or relations are extended), and *replacing* (elements or relations are replaced) (TERNITÉ, 2009). KUHRMANN *et al.* (2014) presented a review of all specific types of operations that have already been defined from the V-Modell XT meta-model.

Figure 13 shows the definition of a variability operation (concept) named *RenameWorkProduct* in the meta-model level (on the left part of the figure) and its instantiation in a process variant (on the center part). Such an operation is used to change the name of the work product *ABC* defined by the base process (reference model) from which the variant is derived. Finally, the change represented by the operation *RenameWorkProduct* must be processed by a merge tool to generate the resulting variant, in which the name of the work product *ABC* has been changed to *XYZ*.

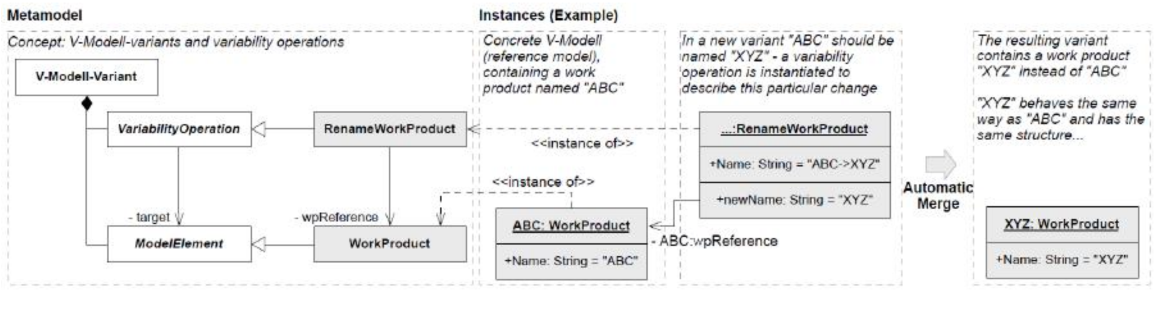

Metamodel
Instances (Example)
Concept: V-Modell-variants and variability operations
V-Modell-Variant
VariabilityOperation
RenameWorkProduct
- target
- wpReference
ModelElement
WorkProduct
Concrete V-Modell (reference model), containing a work product named "ABC"
<<instance of>>
<<instance of>>
ABC: WorkProduct
+Name: String = "ABC"
In a new variant "ABC" should be named "XYZ" - a variability operation is instantiated to describe this particular change
...:RenameWorkProduct
+Name: String = "ABC->XYZ"
+newName: String = "XYZ"
- ABC:wpReference
Automatic Merge
The resulting variant contains a work product "XYZ" instead of "ABC"
"XYZ" behaves the same way as "ABC" and has the same structure...
XYZ: WorkProduct
+Name: String = "XYZ"

Figure 13. Concept and example of variability operation in V-Modell XT (KUHRMANN *et al.*, 2014)

The support of tailoring provided by the framework and meta-model V-Modell XT was clearly based on the tailoring support of SPEM, since the solution structure of both frameworks is very similar. They specify adaptations on a base model through an extension model, which does not modify the original model, but only refers to it.

SPEM and V-Modell XT are two important technologies for process tailoring from the SPE domain, since they explicitly represent tailoring concepts in meta-model. Well-defined concepts specifying tailoring possibilities, i.e. adaptation operations, and their constraints are important in order to support a process engineer in specifying process adaptations (MARTÍNEZ-RUIZ *et al.*, 2012). The explicit representation of adaptation operations also favors the traceability of process derivations, as supported by SPEM and V-Modell XT. However, the main drawback of these technologies is that they do not support adaptations regarding the behavioral perspective of processes, which specifies when activities are performed (i.e., the workflow of the process). In fact, as discussed in Section 3.2.4, few approaches for software process tailoring address adaptations on the process behavior. SPEM and V-Modell XT also support only basic adaptation operations (which modify single elements of a process), which can require very effort and time of the user to specify even simple adaptation scenarios.

Our solution structure has also been based on SPEM and, for this reason, it also presents similarities to the framework V-Modell XT. However, unlike SPEM and V-Modell XT, our solution takes into account the behavioral perspective of process (i.e., its execution flow), which means our tailoring operations need to result in valid workflow models (regarding the BPMN process meta-model). To this end, the state of the model when specifying and processing operations must be considered by pre- and post-conditions associated to adaptation operations. Regarding the supported operations, we believe it is important to provide with the solution a set of operations to cover the most of tailoring scenarios from the BPM domain in general.

3.5. BPMN-based Adaptation Approaches

The Business Process Model and Notation (BPMN) is an ISO and OMG standard for modeling business processes and a *de-facto* standard in professional practice (CHINOSI and TROMBETTA, 2012). In this section, we investigate which approaches/techniques have been proposed from the literature to adapt BPMN process models and thus derive new process models. A possible solution for this issue would be to extend BPMN with specific concepts, i.e., related to process adaptation, but our investigation has not been limited to researches proposing BPMN extensions. Our scope was a broad analysis of BPMN-based adaptation approaches. With this purpose, we conducted a *structured* literature review[7] on such approaches.

3.5.1 Structured Literature Review

We started the review by adopting a predefined protocol to avoid the possibility of bias (selection of individual studies not driven by our own expectations). Therefore, we followed the methods specified in the protocol, including the identification of the research question, the selection of studies, data extraction and synthesis of findings. The protocol underpinning our structured review is organized according to the following activities:

1. Define a research question.
2. Locate and select relevant research studies – We tried to find papers and reports in journals and papers with peer review.
3. Critically evaluate the studies – We assessed each research work against a set of criteria related to the quality of BPMN-based adaptation proposals.
4. Combine the results – The findings were compiled and aggregated in Table 6 for comparison purposes.

3.5.1.1 Research Question

The goal of this review was to identify studies that provide an approach for design-time process adaptation based on BPMN process models. Thus, the research question that guided this structured literature review was:

- *How BPMN process models have been adapted at design-time for deriving new process models?*

[7] We called our review of *structured* and not *systematic* because it has been conducted uniquely by the researcher of this thesis.

3.5.1.2 Studies Selection

This step was intended to specify the search strategy aiming to detect relevant literature on BPMN models' adaptation. To comply with the protocol, we searched for studies using a predefined query string. We were interested in research works related to process adaptation exclusively in the context of BPMN and that deal with *design-time* adaptation. Therefore, we chose a set of terms related to adaptation and configured our query string to disregard results containing the word "dynamic", since it is usually associated with *runtime* solutions:

- BPMN AND (adaptation OR adaptability OR adaptive OR tailoring OR variability OR variant OR configurable OR customization OR customizable OR flexibility OR flexible OR refinement) AND NOT dynamic

After the definition of the query string, the next step was to submit the query to the chosen search base, which corresponds to Scopus8. We have chosen only this base because it groups publications from the more relevant sources in Software Engineering and Business Process Management. The string has been applied evaluating the title, abstract, and keywords of researches from this base. The result of this search retrieved 315 publications. Then, the selection of relevant researches was done by reading the abstract, introduction, and conclusion of each article.

We have selected only publications finding the following inclusion criteria:

(1) Article containing information about any kind of approach concerning the adaptation of BPMN process models;
(2) Article referring for design-time adaptation.

We also applied some exclusion criteria:

(1) Publications with less than 6 pages;
(2) In case several studies refer to the same process adaptation approach, all studies, except the latest and most complete version, were excluded.

As a result of applying such criteria, the final set of researches was composed by 9 articles (listed in Table 5).

[8] http://www.scopus.com

3.5.1.3 Summary of Selected Works

We summarize below each of the selected approaches (Table 5):

1. CUI (2017) proposed a template-based approach for developing new processes. A Template Model is used to model a process template, which represents common features of a set of concrete processes. On the other hand, a Customization Model expresses the way of building a concrete process based on a Template Model. To specify Customization Models, the author proposed a BPMN extension, allowing *add* and *delete* process elements and *modify* properties. These operations, however, are used only for creating customized forms associated to process tasks, i.e., the approach does not deal with adapting control-flow of process templates. Finally, from these two models, an Instance Model can then be generated, which is a standard BPMN model. According to the author, the approach is practically used in an organization, facilitating process model reuse and consequently the development and maintenance of new processes. He argues because all the proposed models are based on the standard BPMN language, such models can easily be understood and manipulated by people and BPMN-compliant tools. *Main limitations:* (1) BPMN extension is not formalized; (2) it does not support control-flow adaptation (only of task's form elements).
2. YOUSFI *et al.* (2016) designed a configurable business process in which variability is directly embedded into a BPMN process diagram. They defined variability mechanisms for BPMN by using flow controls of the own language, i.e., Gateways, for imposing variation points in the process. Each flow branch from a variation point (gateway) represents a variable partition (activated by a data flow or an event flow) that delimits where multiple possibilities may occur. Data- or Event-based variation points can be of three types: single choice (represented by a XOR Gateway), multiple choice (represented by an OR Gateway), or still optional (also represented by an OR Gateway). In this proposal, variant elements can be activities, intermediate events, or sequence flows. Performed adaptations by this proposal result in well-structured process models (DUMAS *et al.*, 2010) and meeting control-flow requirements related to the use of Gateways posed by the BPMN standard (OMG, 2011). *Main limitations:* (1) As a configuration approach, adaptation is limited to addition and removal of pre-defined variant fragments in pre-determined

workflow positions; (2) derived processes by the approach contain only basic flow controls (i.e., mutually exclusive or parallel ones).

3. POPP and KAINDL (2015) proposed to model high-level reference processes (represented in BPMN) with less detail than "*fully-fledged"* processes and to capture missing details in business rules (represented as model-transformations) that operationalize how an organization performs miscellaneous tasks. The application of these rules to a high-level reference process leads to its refinement, generating another process model (more detailed). This proposal executes automatic process model adaptations without any extension of the BPMN standard. Business rules are separately maintained and only coupled with a given process in the course of refinement. In this way, the approach allows that any existing process can potentially be adapted according to business rules. *Main limitations:* (1) It does not define a set of adaptation operations and model well-formedness rules (only provides an example); (2) it does not extend BPMN for specification of adaptations (uses ATL to this end).
4. ASSY *et al.* (2014) proposed a technique for automatically deriving configurable BPMN process fragments from existing process models. The configurable fragments subsume the behavior of the origin fragment models allowing designers to derive any of them from the configurable one. New process variants can then be created by configuring model elements that represent configuration points. In this approach, Gateways and Events are configurable elements. According to authors, a gateway can be configured by restricting its behavior, i.e., reducing its incoming or outgoing branches while preserving its behavior. For example, a configurable OR could be configured to an OR, an AND or a XOR with restricted outgoing flows in the case of a split. A configurable event, in turn, can be included or excluded from a concrete process fragment or refined for a specific type of BPMN event. *Main limitations:* (1) As a configuration approach, adaptation is limited to addition and removal of pre-defined variant fragments in pre-determined workflow positions; (2) the refinement of events for specific types does not have any associated constraint validating the operation.
5. ZHANG *et al.* (2014) proposed a *semantic* extension to BPMN in order to support configurable process modeling with a focus on control-flow perspective. They named the resulting language Configurable BPMN, i.e., C-BPMN. The proposal

defines configurable BPMN tasks that can be set as ON, OFF, and OPT as well as configurable gateways that can be mapped to a concrete choice gateway (representing the logic construct of split or join) or even to a sequence. Likewise the previous works proposing configurable BPMN models, this one also does not define new meta-model concepts. It uses existing BPMN constructs for representing variations. *Main limitations:* (1) As a configuration approach, adaptation is limited to hiding and blocking operations on pre-defined variant fragments in pre-determined workflow positions; (2) no constraint is provided for ensuring semantic correctness of the configured process regarding the BPMN specification; (3) it does not deal with events or different gateways of Exclusive one.

6. DÖHRING and ZIMMERMANN (2011) proposed the approach named vBPMN (variant BPMN) for design-time and runtime customizations of executable process models using BPMN. In our review, we will focus on the aspects of this proposal related to design-time adaptation. It applies structural adaptations to a base model defined in BPMN and annotated with adjustment points, which can be *adaptive activities* or fragments of the model (called *adaptive segments*). The authors also provide a set of specific patterns, defined in the form of block structures, that can be assigned to adaptive activities or inserted into an adaptive segment to customize the model. *Main limitations:* (1) The only possible adaptation operation is *Insert*; (2) adaptation rules for applying patterns to adaptive parts of the model are specified only for runtime settings (i.e., they are triggered after the execution of a particular event), then the approach does not provide any flexibility for customizing the process model at design-time; (3) it is a non-conservative BPMN extension, which requires the base process model to have adaptive segments explicitly marked, so that the usual tooling for BPMN cannot be directly used with this approach.
7. SANTOS *et al.* (2010) proposed the approach GV2BPMN (Goal-Oriented Variability Analysis to BPMN), which promotes the use of goal models to represent variability in BPMN. The aim is to use these models to drive the configuration of business processes, mapping BPMN tasks to goals and keeping links between them. Goals models represent variability in terms of variation points and variants, like approaches for Software Product Line (SPL). Goals in different branches of the decomposition tree can also be related, for example, to indicate exclusion or dependency among them. *Main limitations:* (1) Fails in the process workflow can be introduced by using this type of configuration approach; (2) authors do not mention

how the process' structural correctness is maintained; (3) adaptation is limited to pre-defined configuration options.

8. Although stereotypes are an extensibility mechanism of UML, they were applied by SCHNIEDERS and PUHLMANN (2006) to add variability in BPMN models in this pioneer approach. Four complex variability mechanisms were introduced: *Encapsulation of Sub-Processes* (ESP), *Extension Points, Parameterization*, and *Inheritance*. Encapsulation of sub-processes along with extension points are forms of defining extension points where variants are sub-processes. In this case, ESP imposes that a single variant sub-process be chosen while the Extension Points mechanism specifies that all offered options (sub-processes) are possible, including no choice. Next, the Inheritance mechanism modifies an existing (default) sub-process by adding activities related to specific business rules. Finally, Parameterization, unlike all the previous mechanisms, offers the possibility to represent variability in both events and data, e.g., by customizing the condition of occurrence of an event. Except by this last mechanism, events and gateways cannot be customized. <u>*Main limitations*</u>*:* (1) Adding stereotypes to BPMN models, the approach burdens process diagrams with superfluous notations, hampering the comprehensibility of the model (YOUSFI *et al.*, 2016); (2) *Parameterization* offers very restrictive support for modeling events, while other mechanisms do not offer any possibility in this sense; (3) Gateways cannot be customized, meaning control-flow variations are not supported by the approach.
9. PILLAT *et al.* (2015) describes our approach, named BPMN*t* (BPMN + tailoring), intended for BPMN-based *software* process tailoring, which is depicted in details in Chapter 5. In this article we proposed to extend the BPMN meta-model for including tailoring support similar to the one provided by the SPEM meta-model, which is an OMG standard for software process modeling. This extension is compliant with the standard extension mechanism of BPMN (therefore, conservative) and allows adding SPEM-based tailoring operations to BPMN process elements as extension elements. The proposal supports flexible process tailoring (change), like SPEM, and maintains change traceability links into the tailored process model itself. In this article, we took a first step towards the specification of rules for structural correctness and well-formedness of tailored BPMN models. However, these features were only partially supported. Our more recent contribution

for supporting process tailoring in BPMN (presented in Chapter 6), now focusing on business processes in general, has not yet been submitted for divulgation.

Table 5. BPMN-based Adaptation Approaches

#	Title	Authors	Year
1	An approach implementing template-based process development on BPMN	X. Cui	2017
2	Variability patterns for business processes in BPMN	A. Yousfi, R. Saidi and A. Dey	2016
3	Automated refinement of business processes through model transformations specifying business rules	R. Popp and H. Kaindl	2015
4	Deriving configurable fragments for process design	N. Assy, N. Chan, W. Gaaloul and B. Defude	2014
5	Extending BPMN for Configurable Process Modeling	H. Zhang, W. Han and C. Ouyang	2014
6	vBPMN: Event-Aware Workflow Variants by Weaving BPMN2 and Business Rules	M. Döhring and B. Zimmermann	2011
7	A Goal-Oriented Approach for Variability in BPMN	E. Santos, J. Castro, J. Sanchez and O. Pastor	2010
8	Variability mechanisms in e-business process families	A. Schnieders and F. Puhlmann	2006
	Our Research: BPMN*t*		
9	BPMN*t*: A BPMN extension for specifying software process tailoring	R. Pillat, T. Oliveira, P. Alencar and D. Cowan	2015

3.5.2 Comparison between Approaches

In order to facilitate the analysis and comparison of the selected researches in our review and listed in Table 6, we classified them according to a number of criteria that have been derived considering the analysis perspective "how" of our research question. The adopted criteria were inspired in other literature reviews, such as ROSA *et al.* (2017), which surveyed approaches for business process variability modeling, and BRAUN and ESSWEIN (2014), which surveyed domain-specific BPMN extensions.

Thus, we adopted the following comparison criteria:

- ***Flexibility type***: Do the transformations supported by the approach restrict, extend or change the process behavior?
    - *Restriction* (variability by restriction or configuration). An approach matches this criterion if a process model is customized by restricting its behavior through *configuration*.

    - *Extension* (variability by extension). An approach matches this criterion if a process model is customized of controlled way (e.g., at specific regions) by extending its behavior (i.e., adding elements).
    - *Change* (adaptation). An approach matches this criterion if a process model can be freely modified by the process designer.
- ***Adaptation operations:*** Which adaptation operations are provided by the approach for customizing/modifying the base process model?
- ***Structural correctness***: Is the structural (syntactical) correctness of the derived models guaranteed? This criterion corresponds to the ability of the approach in avoiding disconnected nodes in the derived process.
- ***Well-formedness:*** Are the main control-flow well-formedness rules of the BPMN standard enforced for derived models? This criterion verifies if the approach provides means for ensuring the control-flow well-formedness in produced models considering as comparison base the list of rules presented in Appendix 1.
- ***BPMN-compliant extension:*** Does the approach present an extension definition compliant with the BPMN standard? This criterion verifies if a BPMN extension is defined by using the standard extension mechanism provided by the BPMN specification (OMG, 2011) (described in Section 2.3.2).
- ***Change Traceability:*** Does the approach maintain change traceability links between the tailored model and its base model?
- ***Tool support:*** Does the research describe some tool support?

Table 6. Comparison between BPMN-based Adaptation Approaches

#	**Approach**	**Flexibility Type**	**Adaptation Operations**	**Variant Process Elements**	**Structural Correctness**	**Well-formedness**	**BPMN-compliant extension**	**Change Traceability**	**Tool support**
1	Cui (2017)	Change	*Add*, *Delete* and *Modify*	Activity properties	--	--	--	--	+
2	Yousfi, Saidi and Dey (2016)	Restriction	*Add/Remove* pre-defined process fragments in variation points	Activities, Intermediate Events, Sequence Flows	+	+/- (ref. gateways)	--	--	--
3	Popp and Kaindl (2015)	Change	unspecified	unspecified	--	--	--	--	--
4	Assy et al. (2014)	Restriction	*Add/Remove* pre-defined process fragments in variation points; *Refine* pre-defined events	Activities, Intermediate Events, Sequence Flows	+	--	--	--	+
5	Zhang, Han and Ouyang (2014)	Restriction	*Add/Remove* pre-defined process fragments in variation points	Activities, Sequence Flows	+	--	--	--	+
6	Döhring and Zimmermann (2011)	Extension	*Insert* of customized process fragments in variation points	Activities, Adaptive Segments	+	+	--	--	+
7	Santos, *et al.* (2010)	Restriction	*Add/Remove* pre-defined process fragments according to variation points	Activities	--	--	--	--	--
8	Schnieders and Puhlmann (2006)	Restriction	*Add/Remove* pre-defined process fragments in variation points	Activities (restrictive support for Events)	--	--	--	--	+
9	**SPEM-based BPMN*t*** Pillat *et al.* (2015)	Change	*Extension, Local Contribution, Local Replacement, Suppression*	Activities, Events, Gateways, Sequence Flows	+/- (except for *Contribution*)	+/- (only for *Suppression*)	+	+	+
10	**Pattern-based BPMN*t***	Change	*Delete, Replace, Move, Parallelize, Insert, Encapsulate, Split, Merge, Rename, Specialize, Add Exception Handler, Add Exception Flow*	Activities, Events, Gateways, Sequence Flows	+	+	+	+	+

3.5.3 Analysis

In order to facilitate the comparison of approaches listed in Table 5 with our current research (detailed in Chapter 6), we have included it (#10) in Table 6. Observing this comparative table, some findings are evident:

1) Most of the works proposing BPMN-based adaptation approaches address *variability* modeling, essentially through the technique of *configuration* (restriction) (see column *Flexibility Type* of the table). This meaning that *all* customization decisions need to be known during the creation of the *configurable process model* that will guide user's adaptation decisions. As discussed in Chapter 1 of this research, although this technique facilitates reuse of models, it can only succeed in well-defined contexts where there is little change. Approaches based on configuration are not proper for evolving contexts and, in general, do not support adaptation across levels of abstraction (i.e., refinements), for example, for deriving technical-level BPMN models from business-level models. Conversely, our proposal BPMN*t* (#9 and #10) is based on flexible change operations that can meet adaptation needs from different contexts.
2) Types of adaptations (fourth column) and variant process elements (fifth column) supported by most of approaches are very limited, what can prevent user' adaptation needs being adequately addressed. Adaptation of events, for example, is supported by only three approaches from other researchers (#2, #4, and #8). Conversely, our more recent proposal (#10) supports an extensive set of adaptation operations that can produce process variations affecting (directly or indirectly) all elements from the control-flow perspective.
3) Most of approaches based on variability by restriction (configuration) support structural correctness (except #7 and #8), which is a feature relatively trivial to be achieved when using the technique *hiding and blocking*. However, this feature is rarely supported by techniques of flexible change. From Table 6, only our more recent adaption approach (#10 - Pattern-Based BPMN*t*) fully supports structural correctness, i.e., it avoids flow breaks in the process workflow.
4) Concerning the well-formedness of BPMN models derived from adaptation, only our approach in #10 ensures all control-flow well-formedness rules in Appendix 1, such as prescribed by the BPMN specification. Our approach in #9 takes well-

formedness into account only for removal operations and the proposal in #2 considers this issue only regarding the use of Gateways (flow controls).

5) Our BPMN*t* solution (#9 and #10) is the only research identified from the structured literature review proposing a BPMN-compliant extension. Other researches in Table 6 also present BPMN extensions. However, only we have concerned on proposing new meta-model concepts related to adaptation based on the built-in extension mechanism of BPMN. The importance of using such a mechanism is explained by BRAUN and ESSWEIN (2014): *"authors of BPMN extension should strictly use the BPMN extension mechanism in order to provide a valid extension and enable model exchangeability"*. This is indispensable for reasons of standard conformity, comprehensibility, and tool support. According to the same authors, *"model engineers fail in reusing the most BPMN extensions since they do not provide a valid BPMN extension model. Thus, it is necessary to transform the provided dedicated meta model into a BPMN conform model in order to integrate it within a BPMN tool"*.
6) At last, we also highlight that no approach from other authors describe a specific strategy for maintaining traceability of performed changes. Conversely, we store change traceability links into the adapted process model itself, through adaptation operations.

In summary, our solution seems to be the one that encompasses all evaluated criteria in this section.

3.6. Concluding Remarks

From our analysis of the state-of-the-art on approaches for process adaptation, we have observed some main limitations in the target domains of this research.

In the context of Software Process Engineering (SPE), while variations on the elements that make up a software process (e.g., activities, roles, and data artifacts) are addressed by different proposals (e.g., SPEM and V-Model XT), little attention has been given for variations on the process control-flow. Such variations show differences in *when* process activities are performed and are related to the order (or sequencing) of activities within a process workflow. Although control-flow differences related to the process execution flow may seem primarily just details, they are very important to understand how processes have been performed, how they could be improved and even simulated (ALI *et al.*, 2014). Therefore, effective techniques and mechanisms to specify control-flow variations in software processes are also important.

In the context of Business Process Management (BPM), on the other hand, approaches for process adaptation usually deal with control-flow variations. However, at *design-time* these variations are generally represented through variability modeling, especially by using the technique of configuration. This means that all adaptation possibilities need to be known *a priori*, during the creation of the configurable process model. According to LA ROSA (2017), this technique leads to highly complex models, which hamper the analysis and maintenance of individual process model variants. Approaches based on this technique are not proper for evolving contexts or in which process variations are difficult to predict.

In BPM, *flexible* process adaptation is only supported at *runtime* by some researches addressing change management. However, these researches do not intend to develop new process models from existing ones. Their focus is to manage relationships between a process model and its running instances.

At last, considering the narrow scope of BPMN-based adaptation approaches, we can still highlight that no research from the literature entirely guarantees control-flow well-formedness for the adapted process model regarding rules prescribed by the BPMN specification. Moreover, our solution is the only identified research proposing a BPMN-compliant extension and describing a strategy for maintaining traceability of performed changes.

Thus, in the next chapter we introduce the BPMN*t* solution by presenting its solution structure and depicting the concept of tailoring operation, which is the base of the solution.

CHAPTER IV

4. BPMN*t*

This chapter introduces BPMN*t* (BPMN + tailoring), our thesis contribution that aims at providing a meta-model extension and associated infrastructure to address process adaptation in BPMN. Our solution allows specifying flexible process tailoring in different application contexts, ensuring the correctness of tailored process models and explicitly capturing change traces.

Process tailoring involves adapting an existing process definition to derive a new alternate one. In this thesis, we use the terms *Base Process* or *Reference Process* to refer to the existing process (i.e., the target process of tailoring) and the terms *Tailored Process* or *Variant Process* to refer to the derived process (i.e., the resulting process of tailoring).

The objective of this chapter is only to provide an introduction for the BPMN*t* solution, presenting some general aspects. BPMN*t* comprises two BPMN extensions, which are presented in the next chapters. Thus, Section 4.1 presents the solution structure of BPMN*t* whereas Section 4.2 describes and exemplifies the concept of tailoring operation, which is the base of the approach. Section 4.3 presents some final remarks.

4.1. Solution Structure

The proposed solution structure to address the target problem of this thesis was based on principles of MDE (Model-Driven Engineering) (SCHMIDT, 2006). We defined and applied model transformations (SELIC, 2003), which are actually adaptation operations, to automate each step of the process tailoring procedure and connect the involved process models, i.e., the base (reference) process model and the tailored process model. These adaptation operations are *agnostic* to the reference process, meaning they can be applied to adapt any well-formed BPMN process model. The reference model does not need to be previously “prepared” for adaptation neither to contain special modeling elements, what could limit its use in practice. Our adaptation operations are also *outplace* (MENS and VAN GORP, 2006), i.e., they transform (adapt) the reference process in a separate model, preserving the unmodified original model.

We defined these adaptation operations as meta-model concepts into our BPMN's extension, which we call BPMN*t*. Such operations also represent traceability relationships that trace links between elements of the tailored process and elements of the reference process at the model level, identifying which elements of the second model the first one modifies.

Inspired by techniques of software process tailoring embedded in meta-models of widespread process modeling languages in the SPE's domain, we also applied in our BPMN-based solution the concept of partitioned process (KUHRMANN *et al.*, 2016) for creating tailored processes. Thus, every process derived from tailoring relies on two physically separated, but logically connected models: the *Base Process Model* and the *Tailored Process Model*. This relation is schematically represented in Figure 14, which shows the process models required and produced by our approach.

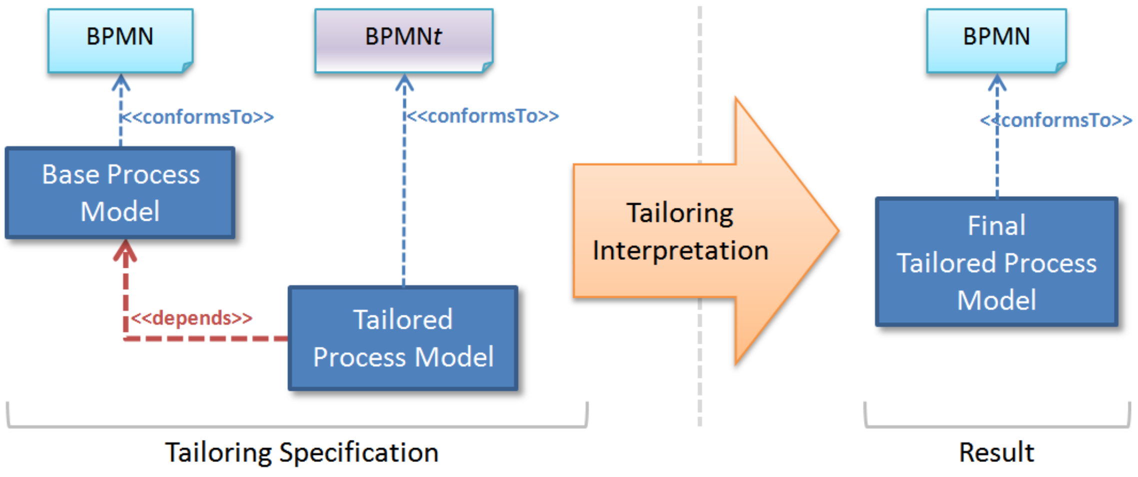


Figure 14: Process Models used by BPMN*t*

Our solution requires a *Base Process Model*, which is a BPMN model containing the specification of the process to be tailored. BPMN process models consist mainly of activity workflows represented as diagrams.

In the tailoring specification phase (left part of Figure 14), a *Tailored Process Model* must be built to represent the process derived from the first one. This model must only contain the changes (or differences) in relation to the *Base Process Model*, which are specified using BPMNt tailoring concepts (i.e., *adaptation operations*) and relationships (i.e., *adaptation traceability links* connecting model elements, represented by the arrow *<<depends>>* in Figure 14). Therefore, the *Tailored Process Model* contains some BPMN process elements (additions to the base process) plus BPMN*t* elements that make it dependent on the *Base Process Model*. Thus, since such a model is not entirely compliant to BPMN (because it contains only part of a process), we refer to it as a BPMN*t* model instead of BPMN model. The *Base Process Model,* which is target of tailoring, will remain unmodified.

Finally, in order to provide users with a unified tailored process model, both models involved in the creation of a variant process need to be merged into one integrated process model, which we refer as *Final Tailored Process Model* (represented on the right part of the figure). The procedure that integrates these models into a single one we call *tailoring interpretation.* In this moment, a set of correctness rules is applied to interpret tailoring concepts and relationships of the tailored process and generate the *Final Tailored Process Model*. This last one contains a complete BPMN model, i.e., the complete set of BPMN elements that compose the derived process. Thus, this model can be loaded into any BPMN tool.

This solution structure provides as advantage the separate declaration of the required changes by a variant process (defined in the *Tailored Process Model*), such that they can easily be accessed and analyzed when it is necessary. Moreover, the *Base Process Model* remains unmodified and decoupled from its variants.

Figure 15 shows a more comprehensive view of our solution structure. *Tailoring operations* are used as a container for the declaration of a process change and an *interpreter* component is provided to execute such operations, performing actions on models according to the semantics of the operations.

The solution structure represented in Figure 15 involves two model levels: the meta-model level (upper part) and the instance level (lower part). The represented meta-model level corresponds to our BPMN*t* extension, which defines the types of tailoring operations that can be used on the instance level. For each tailoring operation type, specific semantics must be provided and implemented in the *interpreter*.

At the instance level, instances of tailoring operations reference target elements in a *Base Process Model* to indicate that these elements will be adapted. The references to base model elements are created by means of operation parameters, which also act as *adaptation traceability links*.

An interpreter is necessary to execute the tailoring operations on the target elements. The interpreter must know the semantics of the tailoring operation types and be able to take into account specific parameters provided by attributes of each operation type.

The interpreter uses a *Tailored Process Model* and merges it with a *Base Process Model*, thus executing all tailoring operations in the order in which they appear in the first model. The result (i.e., a *Final Tailored Process Model*) is generated in a separate file.

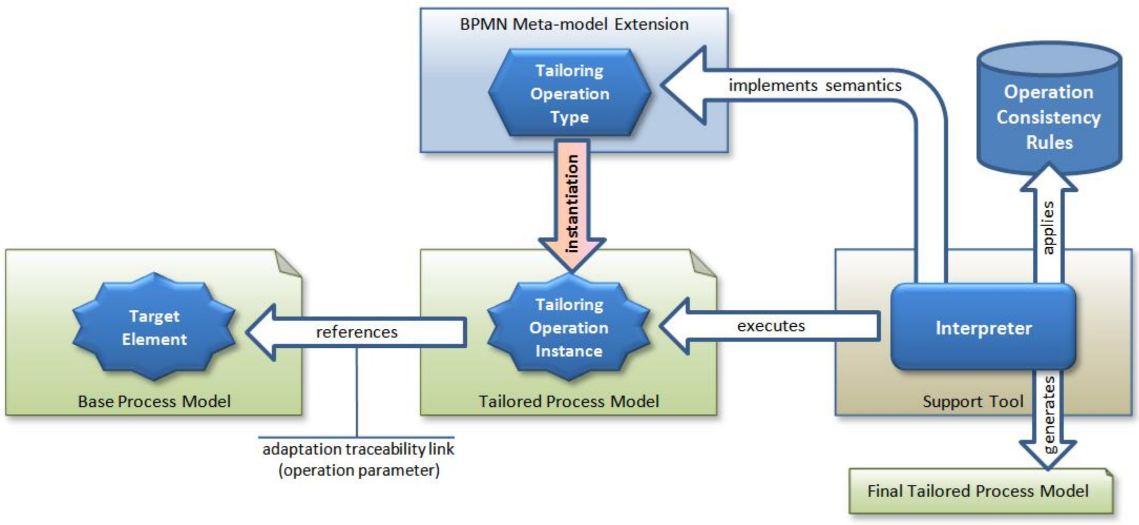


Figure 15: Solution structure of the BPMN*t* approach

4.2. Tailoring Operation

Tailoring operations allow a variant process to reuse and modify content from a base process. These modifications are valid only into the scope of the variant process; they do not alter the base process itself. Indeed, our tailoring operations are model elements (integrated to BPMN via conservative extension) that define a process modification, e.g., renaming elements or inserting new elements in specific workflow positions. Figure 16 exemplifies the tailoring operation *Serial Insert* as concept in the meta-model level (at the top) and as instance in the model level (at the bottom). According to the BPMN extension mechanism, tailoring concepts defined in the *BPMNt Tailoring Extension* package can be added to any BPMN element as extension elements. However, we have limited their application to relevant process elements through additional rules. The concept *SerialInsert* defines two relationships to the BPMN meta-class *FlowNode* (only one of them is mandatory), which represents all process graph nodes (task, subprocess, event, or gateway). *SerialInsert* aims at adding a new process node to the workflow position after the node identified by *after* or before the node identified by *before*.

In the model level at the bottom of Figure 16, a *tailored process* extends a *base process* by using another tailoring operation named *Extension*. Basically, it indicates that a given process will reuse the content of the base process, thus inheriting all its element structure and enabling other adaptation operations. An instance of operation *SerialInsert* (dotted arrow in the figure) is added as extension element to the task *Treat Incorrect Measures*, the only one defined in the *tailored process*. This task is referred as source of the operation, since the extension element is defined by it. The parameter *after* of the operation is set to task *Verify Measures*, meaning the *tailored process* modifies the process inherited from the base process (via *extension*) by adding the task *Treat Incorrect Measures* after the task referenced by the operation parameter *after*. We refer to this parameter as *tailoring relationship* or *adaptation traceability link*. It is configured by a process engineer while specifying process adaptations with BPMN*t*. Such relationship creates a link between an element of the variant and base processes in order to keep traces of changes. Tailoring relationships (or traceability links) are essential for supporting future evolution of processes. Finally, the process model shown in Figure 16 identified as *final tailored process* presents the result of applying tailoring operations defined by the *tailored process* on the *base process* model.

4.3. Concluding Remarks

This chapter depicted the solution structure of our reseach and the concept of tailoring operation, which is the base of the solution. In the next two chapters, we describe the BPMN extensions that compose the BPMN*t* solution.

In Chapter 5, we present SPEM-based BPMN*t*, which consists of a BPMN-compliant extension designed for representing *software* process tailoring. It extends the BPMN meta-model for including tailoring support similar to the one provided by the SPEM meta-model, which is an OMG standard for software process modeling. This extension aims at allowing the specification of control-flow variations in software process models represented with BPMN. Therefore, it addresses the main limitation identified from the state of art in SPE. It also moves towards providing some rules associated to tailoring operations in order to help in the well-formedness of adapted models. The content of this chapter corresponds to our publication in the *Information and Software Technology* journal (PILLAT *et al.*, 2015).

In Chapter 6, we present Pattern-based BPMN*t*, which is also a BPMN-compliant extension, but intended for *business* process adaptation in general (including software processes). It extends the BPMN meta-model for including tailoring support based on high-level operations, which encapsulate basic operations in order to abstract the user from details of process model transformation and ensure correctness of the adapted model. These operations have been derived from *adaptation patterns* (WEBER *et al.*, 2008) (described in *Section 3.3.2.1*) and *refinement patterns* (BRANCO *et al.*, 2014) (described in *Section 3.3.2.2*) identified for the BPM domain.

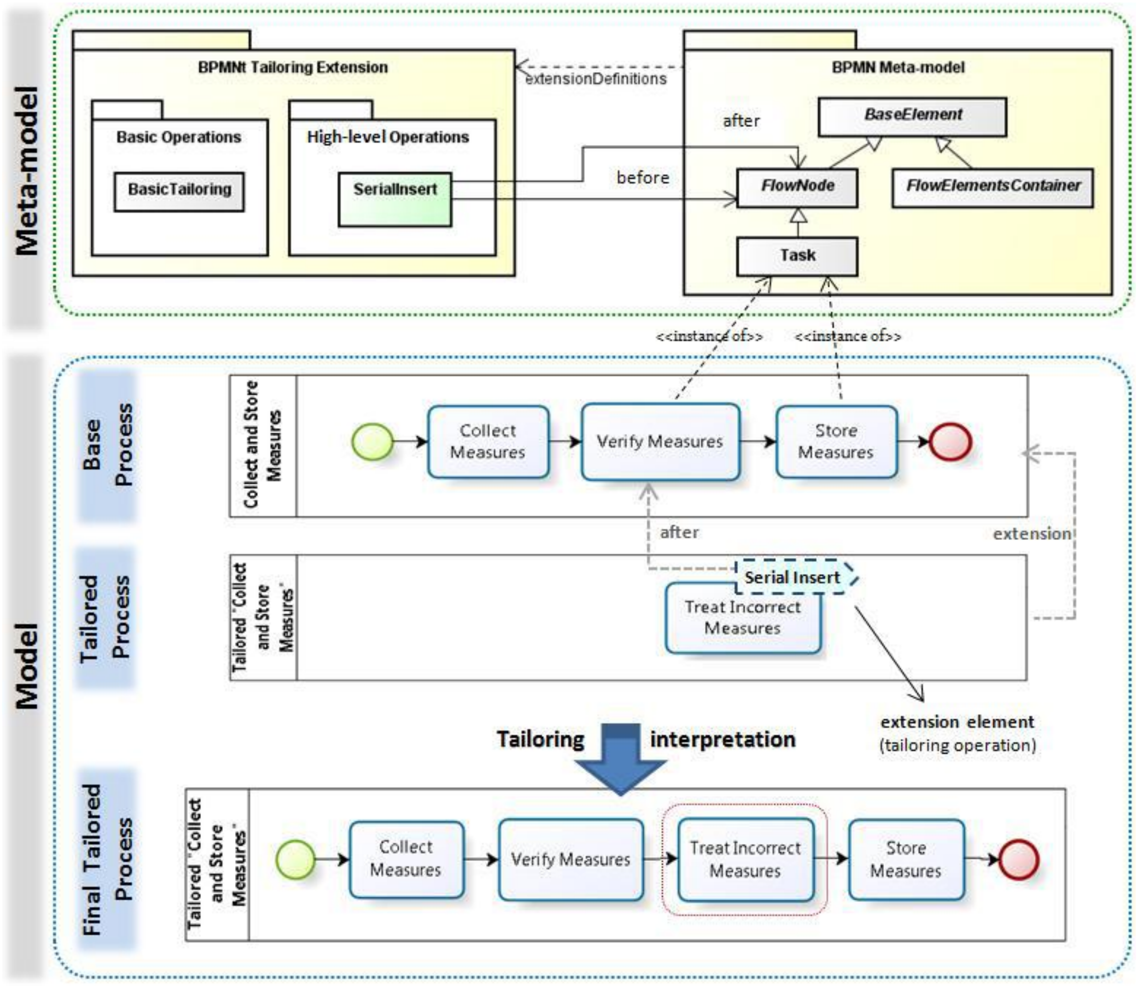


Figure 16: Concept and example of tailoring operation

CHAPTER V

5. SPEM-BASED TAILORING SUPPORT

5.1 Introduction

BPMN 2.0 supports the representation of process activities, roles and information, which resembles SPEM capabilities, but also provides additional concepts related to the technical support and execution of processes, which allow the precise modeling of their behavior. However, BPMN lacks a representation and associated mechanism for process tailoring. Thus, this chapter presents our SPEM-based BPMN*t* approach, which includes tailoring capabilities in BPMN by supporting a process adaptation mechanism similar to the one provided by SPEM 2.0. This way, our solution enables the reuse of BPMN-based software process representations. To this end, we have extended BPMN to include the representation of SPEM-based tailoring operations such as *suppression*, *contribution* and *replacement*. This extension for tailoring support is presented in Section 5.2.

We have also identified some rules to ensure model well-formedness when using such adaptation operations. They are presented in Section 5.4. However, well-formedness checking does not cover contribution operations. Such a limitation will be addressed by the solution presented in the next chapter. In order to validate this initial tailoring proposal we have implemented a prototype to support the BPMNt approach, presented in Section 5.5, by extending resources of the MDT/BPMN2 Project (MDT, 2012). Then the approach has been applied to represent real process adaptation cases from an academic management system development project, as presented in Section 5.6. A running example from the evaluation study (depicted in Section 5.3) is used along this chapter to explain our approach.

5.2 SPEM-based Tailoring Operations in BPMN*t*

In order to represent the concepts related to process tailoring in the context of BPMN-based software processes, we have developed BPMNt (BPMN + tailoring), a conservative BPMN extension. By conservative we mean an extension that does not modify the original semantics of the BPMN specification. The proposed extension introduces elements into BPMN that capture the syntax and semantics to support the ability to remove, replace and add process elements. We have defined such an extension based on the process tailoring concepts found in SPEM 2.0 (discussed in Section 3.4.1 and represented in Figure 11) and on the BPMN standard extension mechanism (discussed in Section 2.3.2 and represented in Figure 6), which allows specifying new attributes (of extension) for BPMN elements.

The SPEM-based BPMNt extension is represented by a new concept, named *Tailoring,* and its associated attributes. The three new attributes will be used by sub-classes of BPMN *FlowElementsContainer* (i.e., *Process* and *Subprocess* elements) and sub-classes of BPMN *FlowElement* (e.g., *Task*, *Gateway*, and *Event*) because they allow the specification of processes and sub-processes in a hierarchical structure, similar to that used in SPEM. The new attributes are:

- ***usedBaseElement***: A process element represented by *FlowElementsContainer* or *FlowElement* should have an attribute to represent its association with another similar element indicating that a tailoring operation will occur. Thus, we have added the attribute *usedBaseElement*, which must be related to the class (process element) that will be reused (tailored).
- ***useKind***: This attribute defines the type of tailoring operation carried out between elements related through *usedBaseElement*. The semantics of tailoring operations are represented by the enumeration *ElementUseKind*, which has the same values as its SPEM counterpart:
  - **NA** denotes the default when the *usedBaseElement* relationship is not defined.
  - **Extension** defines the reuse of a process structure based on an instance of class *FlowElementsContainer* (i.e., a process or subprocess). In other words, an extension relationship makes a copy of the process being pointed at and associates it with the source process (such as an inheritance mechanism). The

relationships *localContribution*, *localReplacement* and *suppressedBaseElement* must be used in conjunction with *extension*.

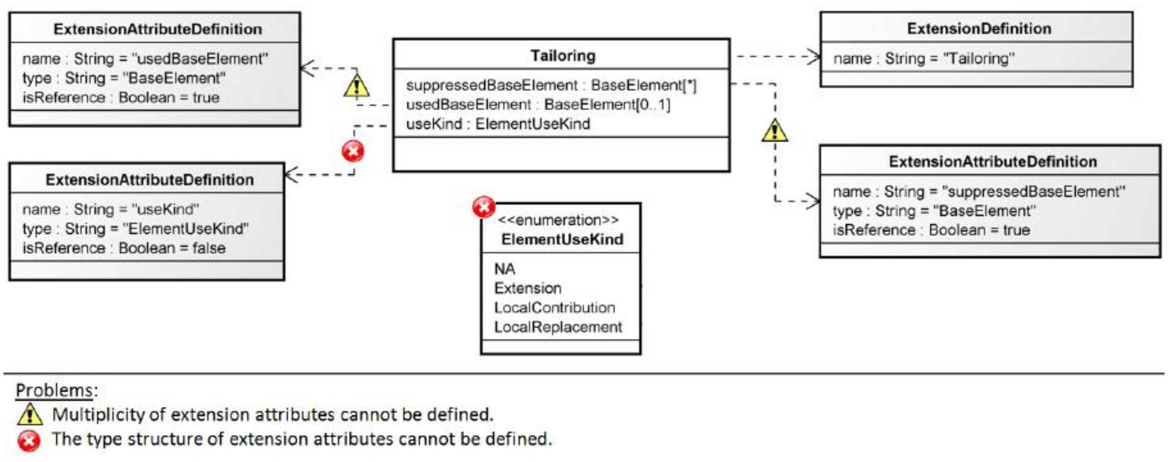
  - **LocalContribution** indicates that a (sub)process adds elements to another inherited by the extension relationship. For example, if process A extends process B, a subprocess of A, say A.1 can add elements (contribution) to a subprocess of B such as B.1 relating to it through a localContribution. Thus, in this case the subprocess B.1 in A will have its own elements plus all the content of A.1.
  - **LocalReplacement** indicates that a process element replaces another inherited by the extension relationship. For example, if process A extends process B, a subprocess of A, say A.1 can replace a whole subprocess of B such as B.1 linking to it through a localReplacement. Thus, in this case the result in A will be A.1 with its own content.

- *supressedBaseElement*: This relationship attribute supports excluding any *BaseElement* of the inherited process structure. It is used in the context of a process reused by the *usedBaseElement* attribute. After a process A extends a process B, it is possible to remove elements of A using this relationship.

5.2.1 Meta-model Representation

Figure 17 illustrates the new BPMNt concept *Tailoring*, its attributes and how they could relate to BPMN extension meta-classes (presented in Figure 6-a). The relation between the new elements and the original meta-classes are represented in this figure by *dependency* relationships, although those relations should be *instance* relationships representing how the model elements conform to the meta-model elements. Although using the BPMN meta-model extension mechanism makes it easier to define and explain a new extension, such a representation does not properly support the BPMNt extension. The BPMN extensibility meta-classes do not support the definition of extension attribute multiplicity (number of objects referenced by the attribute). For example, it is not possible to represent that the attribute *suppressedBaseElement* can reference zero or more *BaseElement* elements whereas the attribute *useKind* must contain only one *ElementUseKind* enumeration value. Another problem is that the BPMN extensibility meta-model does not provide any element to define the structure of new types of attributes specified by *ExtensionAttributeDefinition* elements. Consequently, the

ElementUseKind enumeration type of the BPMNt extension (marked with an error icon in Figure 17) is not able to be defined through the BPMN meta-model extensibility mechanism. The definition of attribute multiplicities and new types is only supported in XML Schema. Thus, in order to provide implementation support to the BPMNt extension, we first have defined it using the XML Schema representation.

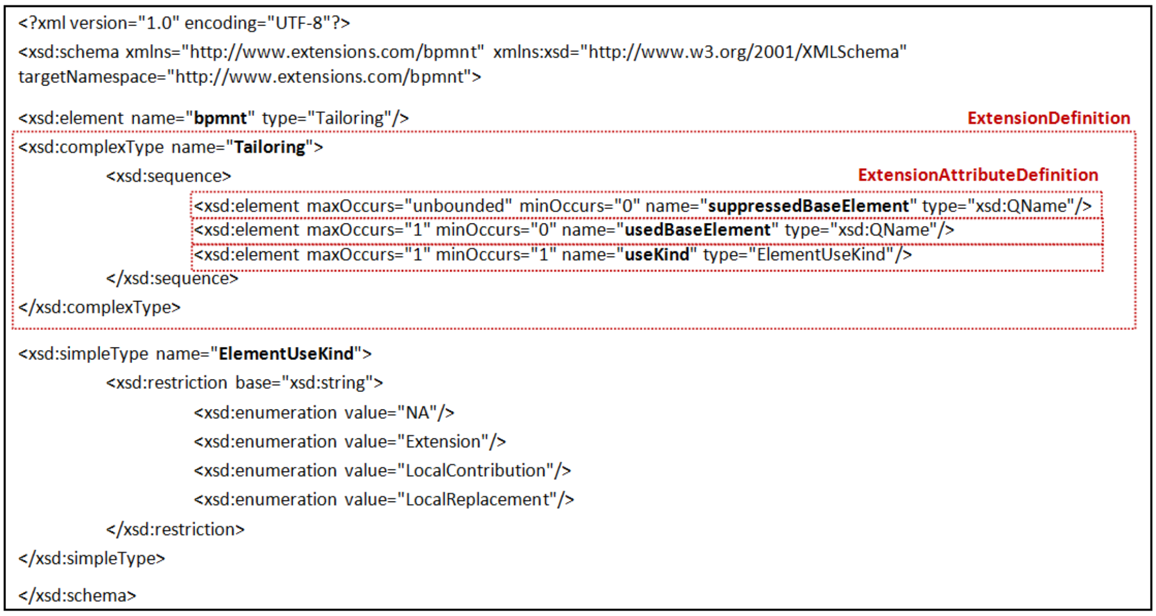

ExtensionAttributeDefinition
name : String = "usedBaseElement"
type : String = "BaseElement"
isReference : Boolean = true
ExtensionAttributeDefinition
name : String = "useKind"
type : String = "ElementUseKind"
isReference : Boolean = false
Tailoring
suppressedBaseElement : BaseElement[*]
usedBaseElement : BaseElement[0..1]
useKind : ElementUseKind
<<enumeration>>
ElementUseKind
NA
Extension
LocalContribution
LocalReplacement
ExtensionDefinition
name : String = "Tailoring"
ExtensionAttributeDefinition
name : String = "suppressedBaseElement"
type : String = "BaseElement"
isReference : Boolean = true

Problems:
⚠ Multiplicity of extension attributes cannot be defined.
✖ The type structure of extension attributes cannot be defined.

Figure 17. Defining the BPMNt extension using the BPMN extensibility meta-classes

5.2.2 XML Schema Based Representation

Figure 18 shows the structure of the BPMNt extension defined using native elements of the XML Schema language. The dotted boxes link elements of the XML Schema document to corresponding concepts of the BPMN meta-model representation. In our case, the element *xsd:complexType* groups the new extension attributes, such as the BPMN meta-class *ExtensionDefinition*. However, unlike the BPMN meta-model representation, our XML Schema extension representation creates a new type of element (named *bpmnt*) to contain the extension attributes of a BPMN model. We have chosen this approach for clarity and modularity reasons. In the tag structure *xsd:complexType*, each extension attribute itself is created by an XML Schema element *xsd:element*, which corresponds to the BPMN meta-class *ExtensionAttributeDefinition*. However, unlike the BPMN meta-model representation, the XML Schema representation supports definition of the multiplicity of extension attributes through the configurations *maxOccurs* and *minOccurs* of the tag *xsd:element*. The type *QName* is used in XML Schema documents to represent references to elements within the same file or in external files, expressed via IDs. Finally, the element *xsd:simpleType* in conjunction with the nested elements *xsd:enumeration* are used to represent the BPMNt enumeration type *ElementUseKind* and its literal values. As mentioned previously, it is not possible to specify the structure of the types of extension attributes using the BPMN meta-model extension representation. To this end, the XML Schema elements *xsd:complexType* or *xsd:simpleType* must be used.



```
<?xml version="1.0" encoding="UTF-8"?>
<xsd:schema xmlns="http://www.extensions.com/bpmnt"  xmlns:xsd="http://www.w3.org/2001/XMLSchema"
targetNamespace="http://www.extensions.com/bpmnt">

<xsd:element name="bpmnt" type="Tailoring"/>
<xsd:complexType name="Tailoring">
        <xsd:sequence>
                <xsd:element maxOccurs="unbounded" minOccurs="0" name="suppressedBaseElement" type="xsd:QName"/>
                <xsd:element maxOccurs="1" minOccurs="0" name="usedBaseElement" type="xsd:QName"/>
                <xsd:element maxOccurs="1" minOccurs="1" name="useKind" type="ElementUseKind"/>
        </xsd:sequence>
</xsd:complexType>
<xsd:simpleType name="ElementUseKind">
        <xsd:restriction base="xsd:string">
                <xsd:enumeration value="NA"/>
                <xsd:enumeration value="Extension"/>
                <xsd:enumeration value="LocalContribution"/>
                <xsd:enumeration value="LocalReplacement"/>
        </xsd:restriction>
</xsd:simpleType>
</xsd:schema>
```

Figure 18. *BPMNt.xsd* – XML Schema Extension Definition.

In the next section, we present a running example that illustrates the application of the BPMNt extension (defined in Figure 18) to a BPMN process model (representing a tailored process).

5.3 Tailoring Software Processes: Running Example

We will use as a running example one simple tailoring scenario from the SIGA-EPCT system development project (SIGA, for short), described in detail in Section 5.6. Such a project has a software development process defined and organized in six phases. However, we will consider only the Specification and Design phase. The modeling of this process phase in BPMN notation is shown in Figure 19 and represents the base process model of the example. The same figure also shows the correspondence between BPMN notation elements and meta-classes. With regard to the modeling of process activities, we have used the BPMN concept *Task* since all modeled activities of the process phase are atomic. Observing the names of these activities in Figure 19, one can note that the project process is centered on use cases, that is, the process is instantiated for each use case of the project. In addition, the project records data about all instances of this process for each use case (tasks, roles and sequence flows) and the process shown in Figure 20 represents an execution for a given use case. As an example tailoring scenario, we consider the representation of the adaptations needed on the Specification and Design phase's base process shown in Figure 19 to derive the specific process shown in Figure 20.

From compliance analysis, it was identified that the executed process represented in Figure 20 had the following changes related to the project base process:

1) The task *Specify Report* was not executed for the use case under consideration and for this reason was suppressed from the tailored process model.
2) Immediately after the task *Design Screen* was performed, a new task *Validate Screen* was also performed, and both tasks were executed in sequence two times; as a result, they have been grouped in the BPMN model as a recursive subprocess named *Design and Validate User Interfaces* (since it involves a loop).
3) A new task *Elaborate Mapping of Use Case Links* was performed immediately after the task *Update General Class Diagram* and before the task *Review Use Case Specification*, requiring changes in their related sequence flows and the addition of the new task to the tailored process model.

Figure 21 shows how the derived process (on the right side of the figure) is related or linked to its base process (on the left side) through BPMNt tailoring relationships to represent the aforementioned adaptations. In such a figure, we have represented both the processes as element trees to facilitate the visualization of tailoring relationships. However, a BPMN process model can be represented either as an element tree or as a diagram.

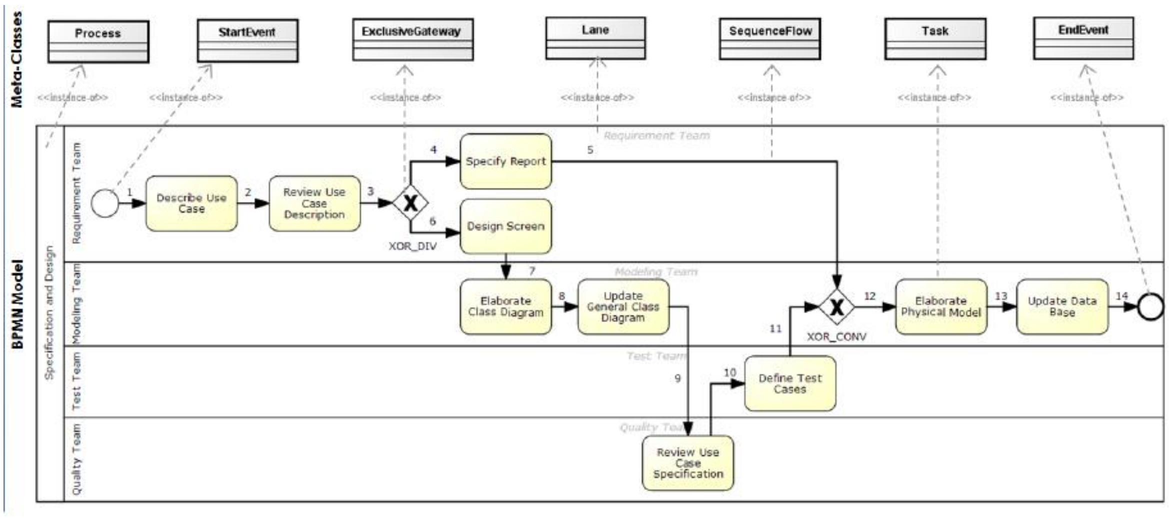


Figure 19. Base process of the SIGA project for the Specification and Design phase.

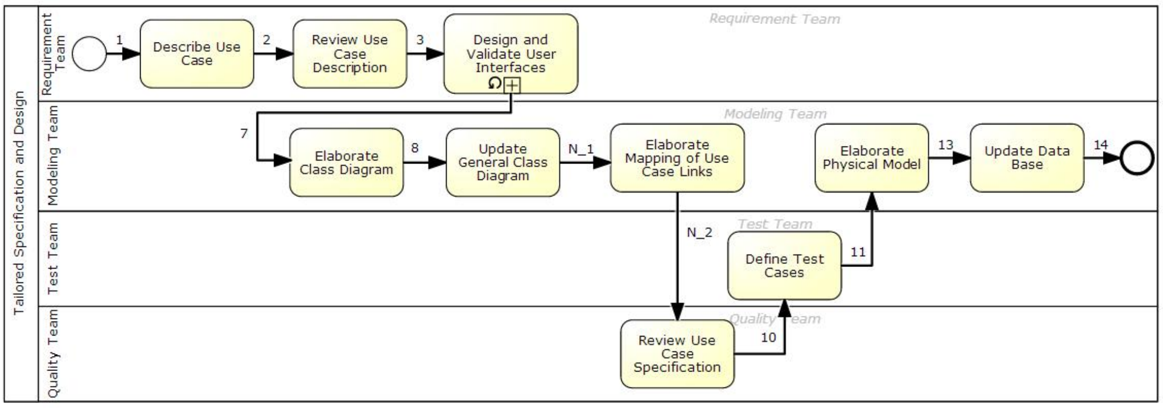

Tailored Specification and Design
Requirement Team
Modeling Team
Test Team
Quality Team
Describe Use Case
Review Use Case Description
Design and Validate User Interfaces
Elaborate Class Diagram
Update General Class Diagram
Elaborate Mapping of Use Case Links
Elaborate Physical Model
Update Data Base
Define Test Cases
Review Use Case Specification
1
2
3
7
8
N_1
N_2
10
11
13
14

Figure 20. Process tailored for a specific use case (process after tailoring interpretation).

In Figure 21, we have named the use case specific process *Tailored Specification and Design*. Such a process is related to the original project process through an *extension* relationship, which means that the tailored process reuses all process elements specified under the original *Specification and Design* process. Moreover, as the derived process adapts the original one, it was required to represent tailoring relationships. There is more than one way to represent the process adaptations just mentioned, but we have adopted a configuration as shown in Figure 21:

- *Suppression*: A relationship between the task *Suppress Task Specify Report* of the tailored process and the task *Specify Report* of the base process using the *Suppressed Base Element* attribute to represent exclusion.
- *Replacement*: A relationship of type *LocalReplacement* between the subprocess *Design and Validate User Interfaces* (which contains the new activity *Validate Screen*) and the task *Design Screen* to represent replacement.
- *Suppression*: A relationship from the task *Suppress Sequence Flow 9* of the tailored process pointing to the sequence flow *9* of the base process using the *Suppressed Base Element* attribute to represent exclusion.
- *Contribution*: A relationship of type LocalContribution between the subprocess Add New Task of the tailored process (which contains the new task Elaborate Mapping of Use Case Links as well as two new sequence flows that connect it to other activities) and the base process itself to represent element addition.

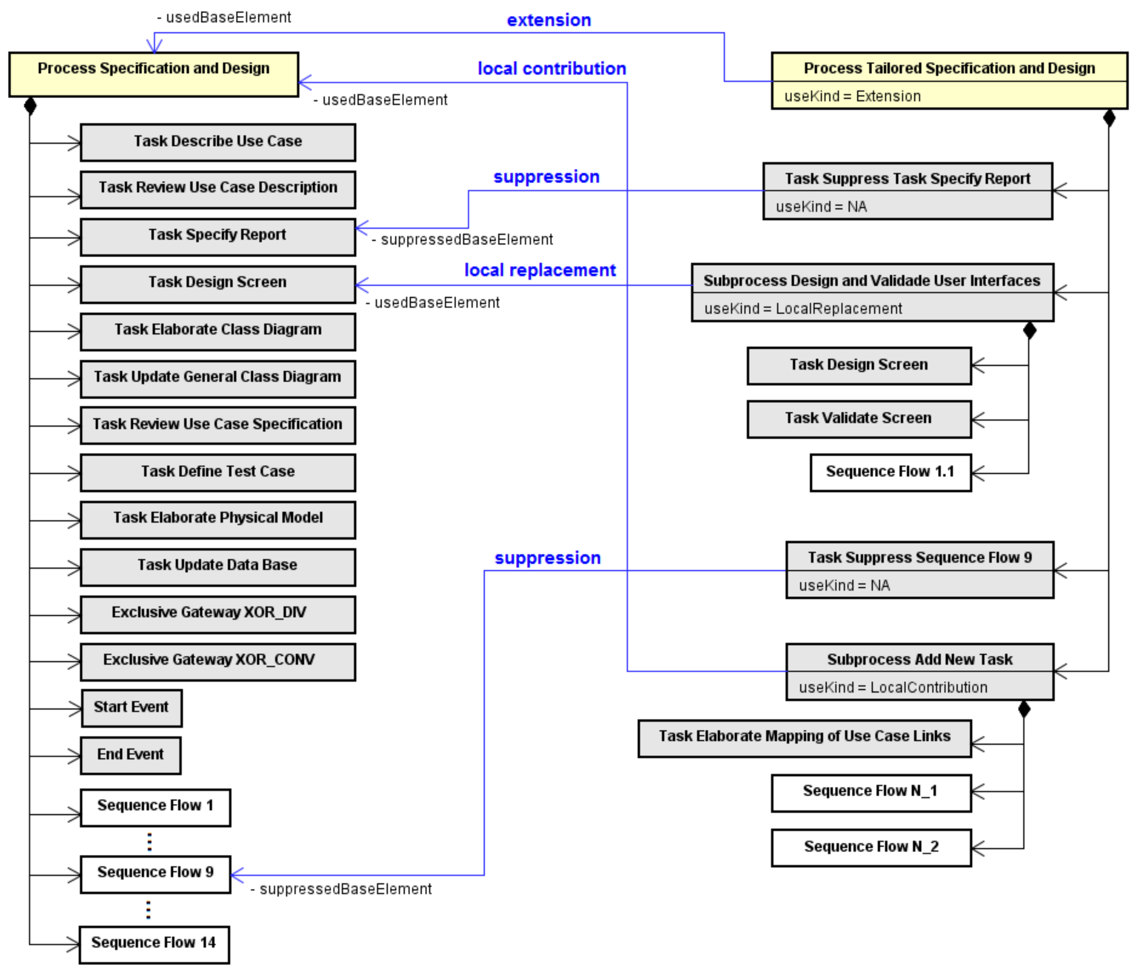


Figure 21. Representation of the BPMNt tailoring specification

Figure 22 shows how the tailoring relationships of the running example (illustrated in Figure 21) are represented in the tailored process file (*Tailored Specification and Design* process) using the BPMNt extension defined in the XML Schema in Figure 18. Elements in bold are part of the BPMN's XML Schema extensibility mechanism whereas elements in red are part of our BPMNt extension. Note that we have imported two files: the first one, *BPMNt.xsd*, contains the definition of our tailoring extension detailed in Figure 18, and the second file, *Specification_and_Design.bpmn*, is a BPMN model file, which contains the process that will be reused (*Specification and Design*). As in the case of the SPEM extension mechanism, the new defined process does not override or modify the base process, but only links to it and its elements through tailoring relationships. For example, the suppression relationship defined by the task *Suppress Task Specify Report* links to task *Specify Report* of the base process through the QName value *BaseProcess:Specify_Report* provided by the extension attribute *extension:suppressedBaseElement*. The first part of the QName value identifies the namespace of the base process (*Specification and Design*), whereas the second part represents the ID of the task *Specify Report*.

```
<?xml version="1.0" encoding="UTF-8"?>
<definitions id="def1" xmlns="http://www.omg.org/spec/BPMN/20100524/MODEL"
xmlns:BaseProcess="BaseProcess" xmlns:extension="http://www.extensions.com/bpmnt"
targetNamespace="TailoredSpecificationAndDesign">
  <import importType="http://www.w3.org/2001/XMLSchema" location="BPMNt.xsd"
namespace="http://www.extensions.com/bpmnt"/>
  <import importType="http://www.omg.org/spec/BPMN/20100524/MODEL"
location="Specification_and_Design.bpmn"
     namespace="BaseProcess"/>
  <extension mustUnderstand="true" definition="extension:bpmnt"/>
  <process id="Tailored_Specification_and_Design" name="Tailored Specification and Design">
      <extensionElements>
          <extension:bpmnt>
              <extension:usedBaseElement>BaseProcess:Specification_and_Design</extension:usedBaseElement>
              <extension:useKind>Extension</extension:useKind>
          </extension:bpmnt>
     </extensionElements>
      <task id="Suppress_Task_Specify_Report" name="Suppress Task Specify Report">
          <extensionElements>
              <extension:bpmnt>
              <extension:suppressedBaseElement>BaseProcess:Specify_Report</extension:suppressedBaseElement>
                  <extension:useKind>NA</extension:useKind>
              </extension:bpmnt>
          </extensionElements>
     </task>
     <subProcess id="Design_and_Validate_User_Interfaces" name="Design and Validate User Interfaces">
          <extensionElements>
              <extension:bpmnt>
                  <extension:usedBaseElement>BaseProcess:Design_Screen</extension:usedBaseElement>
                  <extension:useKind>LocalReplacement</extension:useKind>
              </extension:bpmnt>
          </extensionElements>
          <standardLoopCharacteristics id="loop1"/>
          + <task id="Design_Screen" name="Design Screen">
          + <task id="Validate_Screen" name="Validate Screen">
          <sequenceFlow id="SF1.1" name="1.1" sourceRef="Design_Screen" targetRef="Validate_Screen"/>

     </subProcess>
     <task id="Suppress_Sequence_Flow_9" name="Suppress Sequence Flow 9">
          <extensionElements>
              <extension:bpmnt>
                  <extension:suppressedBaseElement>BaseProcess:SF9</extension:suppressedBaseElement>
                  <extension:useKind>NA</extension:useKind>
              </extension:bpmnt>
          </extensionElements>
     </task>
     <subProcess id="Add_New_Task" name="Add New Task">
          <extensionElements>
              <extension:bpmnt>
                  <extension:usedBaseElement>BaseProcess:Specification_and_Design
</extension:usedBaseElement>
                  <extension:useKind>LocalContribution</extension:useKind>
              </extension:bpmnt>
          </extensionElements>
          + <task id="Elaborate_Mapping_of_Use_Case_Links" name="Elaborate Mapping of Use Case Links">
     <sequenceFlow id="N_1" name="N_1" sourceRef="Update_General_Class_Diagram"
targetRef="Elaborate_Mapping_of_Use_Case_Links"/>
     <sequenceFlow id="N_2" name="N_2" sourceRef="Elaborate_Mapping_of_Use_Case_Links"
targetRef="Review_Use_Case_Specification"/>
     </subProcess>
  </process>
</definitions>
```

Figure 22. *Tailored_Specification_and_Design.bpmn* – Application of the BPMNt extension to a BPMN model

Finally, the complete tailored process, which corresponds in our example to the process shown in Figure 20, is generated by a support tool from the interpretation of the tailoring relationships. Such interpretation is performed following the top-down order in which the tailoring operations (or relationships) appear in the tailored process structure. In the tailored process model shown in Figure 22, the order of tailoring interpretation follows the sequence: (1) extension; (2) suppression (of task *Specify Report*); (3) local replacement; (4) suppression (of sequence flow *9*); and (4) local contribution.

The next section describes the effect of BPMNt tailoring operations on the BPMN elements and presents some well-formedness rules.

5.4 Variant BPMN Elements and Well-Formedness Rules

The approach described in this section to tailor BPMN processes involves four operations to enable process tailoring: *extension*, *suppression*, *local contribution* and *local replacement*. These operations are applied to BPMN flow elements, i.e., elements that interfere directly in the process execution flow. This element group encompasses Task, Subprocess, Gateway, Event and Sequence Flow. Changes in data objects, roles and associations have not been considered yet.

For some tailoring operations, well-formedness rules are provided by the BPMN meta-model specification through UML relationships and their multiplicities. For example, the composition relationship between the meta-classes *Activity* and *ResourceRole* states that the suppression of the first element implies the automatic deletion of the second one. However, many rules are not expressible using the UML class model, especially those that aim to ensure a valid process workflow. Thus, we have defined some rules that help ensure the well-formedness of BPMN models when performing tailoring operations. Such rules are summarized in Table 7 and Table 8 according to the tailoring operation to which they apply. We have used natural language to present our well-formedness rules to be consistent with the BPMN specification. In order to facilitate identification, each rule is numbered and preceded by the tailoring target element type involved in its activation.

5.4.1 Extension Operation

An extension operation must have elements source and target of type *FlowElementsContainer*. In this research, we have considered the containers *Process* and *Subprocess* and an extension relationship on both elements has the same effect: Child elements of the target container (base process) are copied to the source container (tailored process). During the phase of tailoring specification, an extension operation makes all elements of the base process accessible to other tailoring operations as if they were part of the tailored process. For instance, in Figure 21 the extension relationship from the process *Tailored Specification and Design* (the tailored process) pointing to the process *Specification and Design* (the base process) results in the first process containing all elements of the second one. Moreover, the definition of an extension operation enables the use of the tailoring operations suppression, local contribution and local replacement from the elements of the tailored process to the elements of the base process.

5.4.2 Suppression Operation

In a suppression relationship, the type of the source element does not matter because this element does not interfere with the result of the suppression operation. Thus, a suppression relationship can be added to any element of the tailored process or to the process itself. On the other hand, the target element of a suppression relationship must be a task, subprocess, gateway, event or sequence flow. For instance, in Figure 21, the suppression relationship from the task *Suppress Task Specify Report* to task *Specify Report* results in the removal of the latter task from the final tailored process.

Since our approach supports adaptations involving flow elements, the suppression of one or more of these elements from a BPMN process model may lead the process into an incorrect state, such as breaking the sequence of the process execution flow. In order to avoid such scenarios, we have defined a rule set that aims to preserve and recover the process well-formedness after suppression operations are performed. Such rules are presented in Table 7 and are grouped by the element type to which they apply. Supposing, for example, that the task *Review Use Case Description* in Figure 19 was suppressed from the final tailored process, it would cause a workflow break immediately after the task *Describe Use Case*. However, this would be resolved by rule 1, which would automatically connect the precedent element of *Review Use Case Description*

(*Describe Use Case*) to its subsequent (*XOR_DIV*) through the sequence flow *2*. This last one would be reconfigured by the rule to point to the gateway *XOR_DIV*. Such a rule is applied to suppression of any flow element considered in this approach, except *Sequence Flow*. Otherwise, the process execution flow will not be automatically restored, and the process engineer becomes responsible for ensuring that the final process is correct. Next, we describe the rules that apply to each process element that can be the target of a BPMNt suppression operation.

Task suppression: If a task requires input data or produces output data, it contains elements *Data Input* and/or *Data Output* as well as associations that move data to or from the task. Such data elements and associations will be automatically removed if the task is removed because of composition relationships of the BPMN meta-model that connect *Task* to these elements. In addition, we have defined rules 2 and 3 in Table 7 which ensure data objects and artifacts related to the deleted task are also eliminated, since they are not related to other elements. Similarly, rule 4 ensures that events attached to the boundary of a task as well as their exception flows are also removed entirely. The types of events that can be attached to the boundary of a task are *Message, Timer, Error, Escalation, Cancel, Compensation, Conditional, Signal, Multiple,* or *Parallel Multiple* (OMG, 2011). Finally, associations of the deleted task with elements of class *Lane* have to be eliminated (rule 5) as well as message flows that originate from or point to such a task (rule 6).

Subprocess suppression: Since a subprocess is a container for other elements, all the subprocesses' children also have to be removed from the final tailored process, but this is already ensured by the BPMN meta-model's composition relationships. Furthermore, rules 1, 2, 3, 4, 5 and 6 can also be applied to subprocesses just as they have been applied to tasks.

Event suppression: Events may also have specific data requirements to catch a trigger or throw a result that is specified by elements *Data Input* and/or *Data Output.* However, as in the case of tasks, these elements and their associations are removed as a consequence of composition relationships in the BPMN meta-model. Additionally, rule 1 of Table 7 automatically connects the precedent element of a deleted intermediate flow event to its subsequent element. On the other hand, when the removed element is a boundary event (annexed to the border of an activity), rule 7 ensures that exception flows that have such an event as source are also removed. Moreover, data objects, artifacts, message flows or any associations that have the deleted event as source or target are also excluded from the model (see rules 2, 3, 5 and 6).

Gateway suppression: In our approach, it is not allowed to suppress a gateway that shares or unifies more than one process stream (rules 8 and 9). To suppress a gateway, it is first necessary to remove entirely at least one of its streams, so that, in the end, there is only one linear flow coming into and one going out from the gateway. In this situation, the gateway will be automatically removed by rule 10. Then, the element immediately preceding the removed gateway will be connected to the element immediately following this gateway (rule 1). For instance, in the tailoring specification represented in Figure 21, the task *Specify Report* is suppressed from the final tailored process. Observing Figure 19, which shows the base process as a diagram, one can note that such suppression leaves both dangling exclusive gateways (with only one input and output stream). However, this problem is automatically resolved by rule 10. As a result, both the gateways are also suppressed and, then rule 1 connects through sequence flows the tasks *Review Use Case Description* and *Design Screen* (after being replaced by the subprocess *Design and Validate User Interfaces*) as well as the tasks *Define Test Cases* and *Elaborate Physical Model*. The result of these rule transformations can be observed in Figure 20, which shows the final tailored process as a diagram. Moreover, artifacts connected to a deleted gateway and its relationships with lanes are also excluded from the model (see rules 3 and 5).

Sequence flow suppression: No well-formedness rule applies to suppression operations of sequence flows, but they apply only to elements that represent flow nodes. When a sequence flow is removed, the process engineer is responsible for ensuring the correctness of the tailored process model. For example, in Figure 21 there is a suppression relationship that removes the sequence flow *9* from the final tailored process. This causes a break in the process workflow that must be repaired by the process engineer. In the example, the correction of this problem is performed with the contribution operation, which adds new sequence flows.

Table 7. BPMNt well-formedness rules related to element suppression.

ID	Element Type	Suppression Rule
1	Task, Subprocess, Gateway, Event	The last element of the flow immediately preceding the deleted element will be connected to the element immediately following it, except if the predecessor is a diverging gateway and the successor is a converging gateway.
2	Task,	Data objects related to the deleted element must also be eliminated, as

	Subprocess, Event	long as they are not related to other elements.
3	Task, Subprocess, Gateway, Event	Artifacts related to the deleted element must also be eliminated, as long as they are not related to other elements.
4	Task, Subprocess	Boundary events and their exception flows linked to suppression should also be removed entirely.
5	Task, Subprocess, Gateway, Event	Relationships between the deleted element and lanes have to be eliminated.
6	Task, Subprocess, Event	Message flows which originate from or point to the suppressed element have to be eliminated.
7	Boundary Event	Flows of exception (error, time or any other type) with source in the deleted boundary event should also be removed.
8	Gateway	It is not permitted to suppress a gateway that shares a process flow in more than one output stream without completely removing at least one of its flows entirely, so that in the end there is only one final linear flow.
9	Gateway	It is not permitted to suppress a gateway that unifies more than one input stream without removing at least one stream, so that in the end there is only one linear flow coming immediately after suppression.
10	Gateway	By deleting an entire flow to or from a gateway, if there is only one flow entering and leaving this gateway, then the gateway will also be removed.

5.4.3 Local Replacement Operation

The interpretation of a replacement relationship results in the target element being entirely replaced by the source element in the final tailored process. Since our approach supports adaptations of flow elements (task, subprocess, gateway, event or sequence flow), the source and target elements of a replacement relationship can be any of these element types. Further it is not necessary for the source and target elements to have the same type, except when one of them is a sequence flow. For example, a task can be replaced by an event, subprocess, gateway, or another task. It just cannot be replaced by a sequence flow, according to rule 11 of Table 8. Moreover, when replacing a flow node element by another, rule 12 ensures that sequence flows connected to the replaced element are maintained and reconfigured to connect to the new element. For example, in Figure 21, the replacement relationship from the subprocess *Design and Validate User Interfaces* to the task *Design Screen* results in the replacement of the last element by the first one (see Figure 20). The sequence flows *3* and *7* have been reused and reconfigured by rule 12 in order to link to the subprocess *Design and Validate User Interfaces* (new element) instead of the task *Design Screen* (replaced element). Such a rule is not applied when elements connected by a replacement relationship are of type *Sequence Flow*. Finally, rule 13 ensures that the new element (the substitute element) is allocated to the same lane as the replaced element, when this last element has a lane defined.

Table 8. BPMNt well-formedness rules related to element replacement.

ID	Element Type	Replacement Rule
11	Sequence Flow	A sequence flow element can replace only another sequence flow element.
12	Task, Subprocess, Gateway, Event	Sequence flows connected to the replaced element must be reconfigured to connect in the same way as they were connected to the substitute element.
13	Task, Subprocess, Gateway, Event	The substitute element must be allocated to the same Lane to which the replaced element was allocated.

5.4.4 Local Contribution Operation

In a local contribution operation, the elements source and target of the contribution relationship must be of type *FlowElementsContainer*. Such a type is represented by elements *Process* and *Subprocess,* which assemble other process elements. In this way, the interpretation of a contribution relationship results in the final tailored process containing the child elements of the target (sub)process plus the child elements of the source (sub)process of the relationship. In other words, all child elements of the container to which a contribution relationship is added in the tailored process represent contributions or additions to the extended target container that was originally defined in the base process.

It is important to highlight that through this mechanism our approach makes it possible to add new process elements to an existing process by generating an adapted version from that process. However, with this approach the process engineer is responsible for specifying adaptations that do not generate invalid BPMN processes. For instance, in Figure 21, the contribution relationship from the subprocess *Add New Task* to the main process *Specification and Design* results in addition of all child elements of the source subprocess into the target process. Note that the source subprocess of the contribution relationship contains, besides the task *Elaborate Mapping of Use Case Links,* two new sequence flows named *N_1* and *N_2*. These elements insert the new task into the execution flow of the final tailored process in the position between the tasks *Update General Class Diagram* and *Review Use Case Specification* (see Figure 20). The sequence flow *N_1*, for example, connects the task *Update General Class Diagram*, defined originally in the base process model (*Specification and Design*), to the new task *Elaborate Mapping of Use Case Links*, defined in the tailored process model (*Tailored Specification and Design*). Such connection across processes is possible because an extension operation has made all elements of the base process accessible in the tailored process as if they were part of this last process.

In this way, after the interpretation of tailoring relationships in Figure 21, the final tailored process *Tailored Specification and Design* shown in Figure 20 will have its complete element structure, resulting in a valid BPMN process with continuous flow. Finally, as process correctness after a contribution operation must be ensured by the process engineer by using this approach, we do not provide here validation rules related to such an operation. However, after generating the final tailored process, this process can be submitted to a model checker that will detect modeling errors.

5.5 BPMNt Support Prototype

We have developed a support prototype for BPMNt tailoring specification that extends the MDT/BPMN2 Project (MDT, 2012). This project uses the Eclipse Modeling Framework (EMF) to produce Java classes for the BPMN model specification as well as to provide a basic editor for manipulating such models, which are represented as hierarchical structures (trees). In order to support the BPMNt extension proposed in this research, we have transformed its specification in XML Schema (file *BPMNt.xsd* shown in Figure 18) to an EMF model and its manipulation has been integrated with the standard BPMN editor. Thus, our tool prototype supports the addition of BPMNt extension elements to standard BPMN elements without requiring manual extension importation.

Figure 23 shows the typical process of using of the BPMNt prototype. The tailoring process starts by importing into the tool a BPMN file that contains the model specification of a base process (*BaseProcess.bpmn*), which will be reused by another process. Such a model file can be designed using any compliant BPMN 2.0 tool. After, the tailored process, which will reuse part of the structure of the previous one, can be created by the process engineer as a new file of BPMNt type (*TailoredProcess.bpmnt*). Although its structure is of a BPMN file (such as the structure shown in Figure 22), we have opted to use a different extension because such a file contains an incomplete BPMN process specification. In other words, it contains only new elements to be added to the base process and definitions of tailoring operations. Thus, the BPMNt file (tailored process) makes sense only in conjunction with the BPMN file that defines its base process. For this reason, every BPMNt file is linked to a BPMN file through a tag *import* that references the latter file.

After defining a tailoring operation in the tailored process (BPMNt file), the process engineer can observe its result by requesting the generation of the final tailored process. This process is generated by our support prototype in a new BPMN file (*FinalTailoredProcess.bpmn*) that contains the interpretation of the tailoring operations. Such a procedure can be performed at each tailoring operation definition or just at a single time as represented in Figure 23, at the end of the BPMNt file's specification.

Our prototype supports the visual representation of BPMN models as element trees, but the generated model file can be exported to a graphic modeling BPMN tool in order to visualize the final process as a diagram. However, the BPMN tool must support automatic diagram generation since the model produced by our prototype does not contain diagram information. Some examples of BPMN tools that contain such a feature are BPMN Web Modeler[9] and Activiti BPM[10]. Moreover, by specifying some additional configurations on the generated model, it can be interpreted using BPMN-based workflow tools.

Most of our well-formedness rules for tailoring, which are described in natural language in Table 7 and Table 8, are applied by our tool prototype on BPMN models during the tailoring interpretation. Each of these rules is implemented as a Java method and another overarching method is used to fire the appropriate rule(s) for each adaptation in the BPMN model. However, rules 8, 9 and 11 are applied before the tailoring operation interpretation, while the process engineer specifies tailoring operations. When any of these rules is violated in such a phase, the user of the tool is immediately notified.

Figure 24 shows the main graphic interface of the BPMNt support prototype and some of its resources to assist the activities related to the process in Figure 23, using as an example the process tailoring scenario shown in Figure 21. The numbers in Figure 23

[9] http://www.bpmnwebmodeler.com/
[10] http://www.activiti.org/

and Figure 24 trace correspondences between activities of the use process and its support mechanisms in the tool. The activity *Import Base Process* is not detailed in Figure 24. This figure only shows the file *Specification_and_Design.bpmn* (base process of the running example) already available in the current tailoring project (region 1 in the figure). In the central region of the main interface, the element structure of the file *Tailored_Specification_and_Design.bpmnt* (tailored process of the running example) is shown.

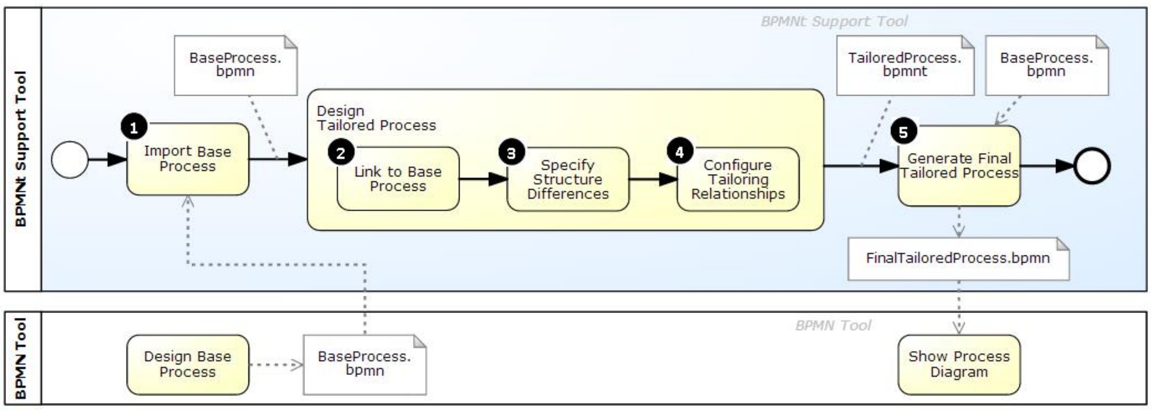
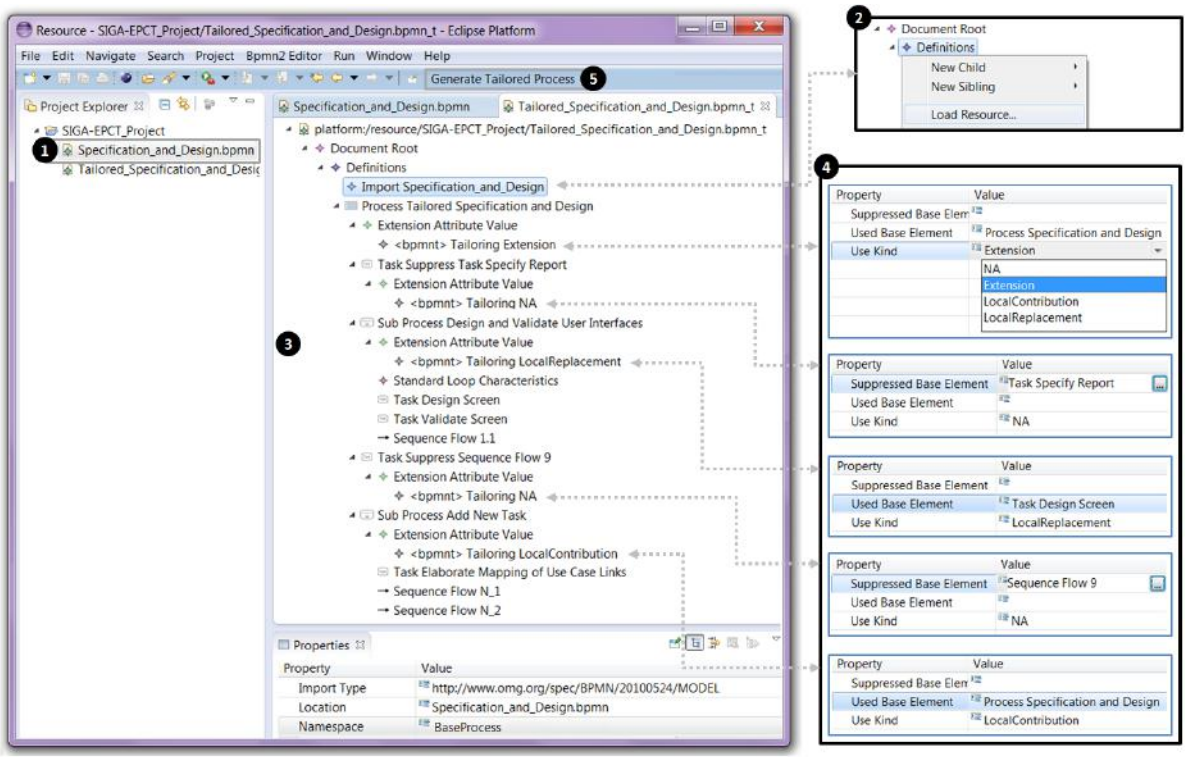


Figure 23. Tailoring process of the BPMNt tool prototype



Resource - SIGA-EPCT_Project/Tailored_Specification_and_Design.bpmn_t - Eclipse Platform

File Edit Navigate Search Project Bpmn2 Editor Run Window Help

Generate Tailored Process 5

Project Explorer

```
SIGA-EPCT_Project
  1 Specification_and_Design.bpmn
    Tailored_Specification_and_Desig
```

Specification_and_Design.bpmn | Tailored_Specification_and_Design.bpmn_t

```
platform:/resource/SIGA-EPCT_Project/Tailored_Specification_and_Design.bpmn_t
  Document Root
    Definitions
      Import Specification_and_Design
      Process Tailored Specification and Design
        Extension Attribute Value
          <bpmnt> Tailoring Extension
        Task Suppress Task Specify Report
          Extension Attribute Value
            <bpmnt> Tailoring NA
        Sub Process Design and Validate User Interfaces
          Extension Attribute Value
            <bpmnt> Tailoring LocalReplacement
          Standard Loop Characteristics
          Task Design Screen
          Task Validate Screen
          Sequence Flow 1.1
        Task Suppress Sequence Flow 9
          Extension Attribute Value
            <bpmnt> Tailoring NA
        Sub Process Add New Task
          Extension Attribute Value
            <bpmnt> Tailoring LocalContribution
          Task Elaborate Mapping of Use Case Links
          Sequence Flow N_1
          Sequence Flow N_2
```

3

Properties

Property	Value
Import Type	http://www.omg.org/spec/BPMN/20100524/MODEL
Location	Specification_and_Design.bpmn
Namespace	BaseProcess

2

```
Document Root
  Definitions
    New Child
    New Sibling
    Load Resource...
```

4

Property	Value
Suppressed Base Elem	
Used Base Element	Process Specification and Design
Use Kind	Extension
	NA
	Extension
	LocalContribution
	LocalReplacement

Property	Value
Suppressed Base Element	Task Specify Report
Used Base Element	
Use Kind	NA

Property	Value
Suppressed Base Element	
Used Base Element	Task Design Screen
Use Kind	LocalReplacement

Property	Value
Suppressed Base Element	Sequence Flow 9
Used Base Element	
Use Kind	NA

Property	Value
Suppressed Base Elem	
Used Base Element	Process Specification and Design
Use Kind	LocalContribution

Figure 24. The BPMNt tool prototype and its mechanisms that assist activities related to the tailoring process in Figure 23.

According to the process in Figure 23, the first activity for designing the tailored process is *Link to Base Process*. Such activity is performed by selecting the menu option

Load Resource (box 2 in Figure 24) and then the base process file. Such a procedure results in the addition of an *import* element in the tailored process structure (selected element in region 3 in Figure 24). The activity *Specify Structure Differences* is carried out through the menu options *New Child* and *New Sibling* (see box 2 in the figure) that support the addition of new BPMN elements of specific types to a process structure. The final difference structure of the tailored process is shown in region 3 in the figure. Note that such a structure contains elements of the BPMNt extension (the names preceded by <*bpmnt*>) and they are always contained by elements *Extension Attribute Value*. These elements are instances of the BPMN extensibility meta-class *ExtensionAttributeValue* (see Figure 6-a) and have been used to contain BPMNt elements in the tool's model graphic representation because of its use in the extension mechanism. Tailoring operations are defined during the activity *Configure Tailoring Relationships* of the tool's use process, in which the properties of the BPMNt elements are set according to each operation type (see box 4 of the figure for configuration details on each tailoring operation). In this context, links to elements of the base process are defined by simply selecting the name of the target element(s) in the properties' selection boxes. Finally, the activity *Generate Final Tailored Process* is automatically executed by the BPMNt tool support after the user clicks on the button *Generate Tailored Process* in the main interface (see component identified by number 5 in the figure). The generated BPMN file will be added to the current tailoring project in region 1 of the figure.

5.6 Evaluation

We have conducted an evaluation study of the BPMNt approach by using real-world tailoring data. This section presents the study, which aimed at assessing the applicability of the proposed BPMNt extension to specify typical tailoring scenarios of software processes. The following subsections present our study's goal, context description, the procedure that was followed, a tailoring case more complex than the running example (used to show the proposed approach), and finally the results and conclusions of the evaluation.

5.6.1 Goal

Our goal with this evaluation was to assess the applicability of the proposed BPMNt solution. In other words, we wanted to verify the ability of BPMNt (and related support mechanisms) to express process variations performed in the context of the SIGA Project (detailed in the following subsection). This goal is detailed bellow according to the structure proposed by BASILI *et al.* (1994):

Analyze	the specification of process adaptation scenarios by using our BPMNt tailoring approach implemented in an EMF-based support prototype.
For the purpose of	Characterize
Regarding	Feasibility of the proposed solution
From the viewpoint of	Researcher and Domain Expert
In the context of	the SIGA software development project, regarding the Software Process Engineering domain.

5.6.2 Research Question

The evaluation aimed to answer the following question:

1) **Q:** *Is the current BPMNt extension (and related support mechanisms) able to specify software process tailoring scenarios from the SIGA Project?*

5.6.3 Context Description

We have adopted software process models and tailoring scenarios from the SIGA-EPCT Project (SIGA, for short). SIGA-EPCT stands for *Academic Management Integrated System for Technological, Scientific and Professional Education* and is intended to develop an academic management system to be used by public universities in Brazil. Such a system is a joint project of several Brazilian research institutions using open technologies. The project has a defined software development process, which comprises the following phases: Planning, Requirements, Specification and Design, Implementation, Test, and Deployment. However, driven by the aim to evaluate our work, we have focused on the Specification and Design phase, since this is core to the project. The representation in BPMN notation of the process associated with the Specification and Design phase is shown in Figure 19. The SIGA project is centered on use cases and as a result a new process is instantiated for each use case of the project. The project records all process instances in a task management system for each use case (that is, the tasks, roles and sequence flows), to allow better orchestration of the people involved. Using the records from the task management system, another research project (SANTOS, OLIVEIRA and ABREU, 2015) has applied data mining and compliance analysis techniques to identify variances between the defined project process and the process executed by the project team in some core use cases. Such variances would be useful to the process engineer to evaluate the possibility of adaptations on the project base process in order to better reflect the actual execution behavior since, according to FERRATT and MAI (2010), tailoring can also be defined in terms of modifications that emerge from the results of monitoring a project and providing feedback on the development process. Thus, in this project we have an opportunity to evaluate our approach and support prototype on real process variability scenarios.

5.6.4 Procedure

A specialist from the SIGA Project team has modeled in BPMN the process of the Specification and Design project phase, used as base process in this evaluation study, and we have specified variances in its execution from ten real use cases of the project by using the BPMNt support prototype. That is, we have created ten BPMNt files in our prototype, each one containing the differences of a specific process execution and tailoring relationships relative to the base process model. Finally, to ensure the correctness of the tailored process models, they have been checked by the domain expert of the SIGA project.

5.6.5 Description of a Complex Tailoring Scenario

Throughout this chapter, we have used a running example, which is a simple tailoring scenario from the SIGA project and corresponds to case 1 in Table 10. However, in order to demonstrate the behavior of our current approach on a more complex scenario, this section provides details about another tailoring case related to the SIGA project. Thus, in this section we consider scenario 10 in Table 10, where the tailoring specification using the BPMNt support prototype is shown in Figure 25. Instead of adding a new task to the tailored process as in the running example, which adds the task *Elaborate Mapping of Use Case Links*, the process in this scenario (represented in Figure 25 box 8) had the following difference:

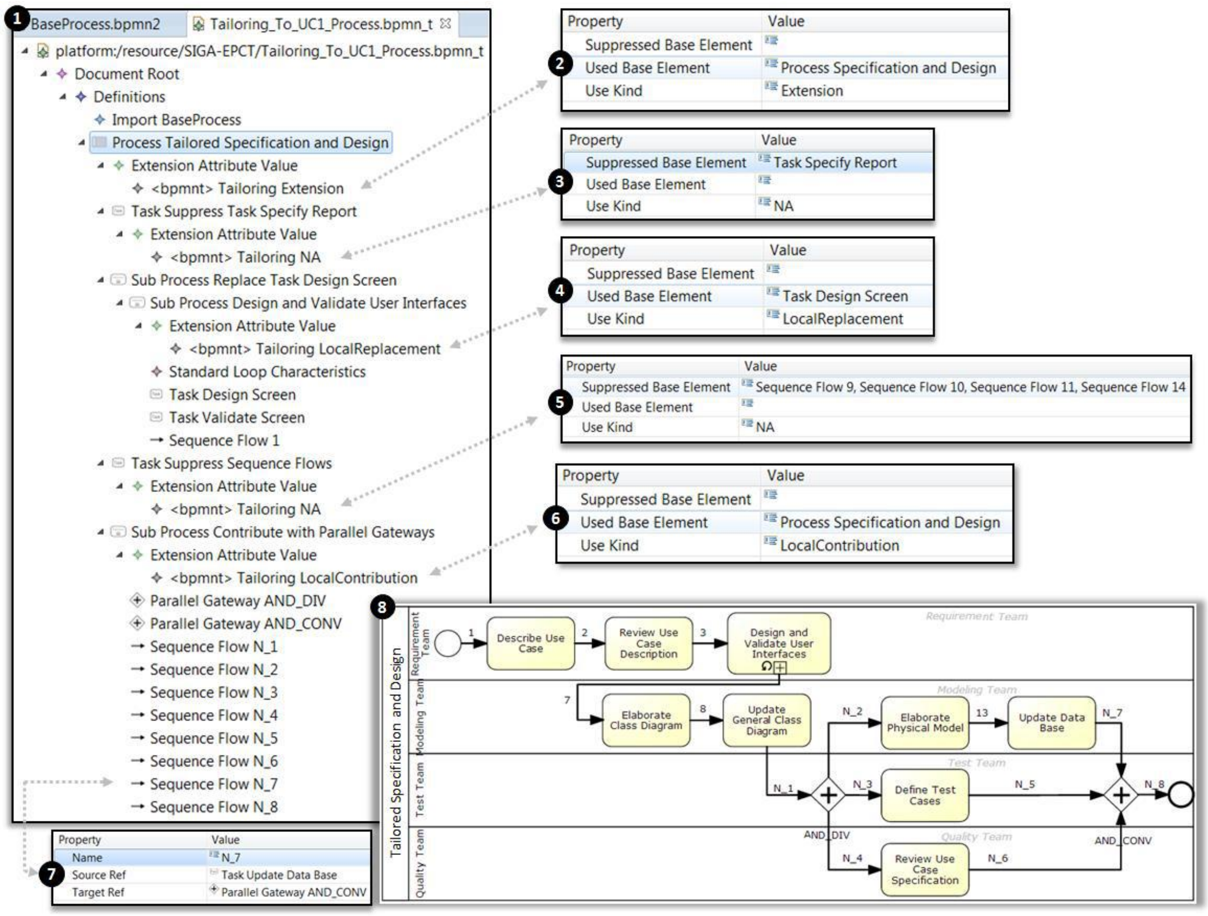
- The tasks *Elaborate Physical Model* and *Update Data Base* were performed in parallel with *Define Test Cases* and *Review Use Case Specification*, requiring changes in their related sequence flows and the addition of parallel gateways, one to share process flow and another to unify it.

To represent this specific change we have adopted the following configuration:

- Suppression: A relationship from the task *Suppress Sequence Flows* of the tailored process pointing to the sequence flows *9,10, 11,* and *14* of the base process using the *Suppressed Base Element* attribute to represent exclusion (see boxes 1 and 5 of Figure 25).
- Contribution: A relationship of type *LocalContribution* between the subprocess *Contribute with Parallel Gateways* of the tailored process, which contains two new gateways to share and unify the process flow as well as new sequence flows connecting them to activities, and the base process itself to represent element addition (see boxes 1, 6 and 7 of Figure 25).

The other tailoring operations of the running example's scenario were repeated in the scenario considered in this section: suppression of the task *Specify Report* (boxes 1 and 3 of Figure 25) and replacement of the task *Design Screen* by the subprocess *Design and Validate User Interfaces* (boxes 1 and 4 of Figure 25).

The tailoring operations are processed in the top-down order in which they are specified in the process structure of the BPMNt file. Thus, after the extension operation, the first tailoring operation interpreted as part of the generation of the final tailored process involves the suppression of task *Specify Report*. In this case, rule 5 in Table 7 identifies the lane to which the task is associated and also removes such a relationship. In addition, the suppression of this task (and its related sequence flows) leaves both dangling exclusive gateways with only one input and output stream and this problem is solved by rule 10. As a result, both gateways are suppressed as well as their relations with lanes, which are removed by rule 5. Now rule 1 connects the tasks *Review Use Case Description* and *Design Screen* through sequence flows as well as the tasks *Define Test Cases* and *Elaborate Physical Model*. With regard to the replacement operation, rules 12 and 13 are applied. This implies that sequence flows connected to the replaced task *Design Screen* are reconfigured to connect to the substitute subprocess *Design and Validate User Interfaces*, and such a subprocess is allocated to the same lane as the previous element. It is also important to mention that, in accordance with rules 8 and 9, no gateway can be directly removed from the process in Figure 19 when it has more than one input or output stream. On the other hand, no well-formedness rule is applied with respect to the last two tailoring operations just described. As mentioned previously in this chapter, our rules do not apply to sequence flow suppressions and contribution operations. Finally, box 8 in Figure 25 illustrates the final tailored process for scenario 10 of the SIGA project using the BPMN notation.


1

```
BaseProcess.bpmn2    Tailoring_To_UC1_Process.bpmn_t
platform:/resource/SIGA-EPCT/Tailoring_To_UC1_Process.bpmn_t
  Document Root
    Definitions
      Import BaseProcess
      Process Tailored Specification and Design
        Extension Attribute Value
          <bpmnt> Tailoring Extension
        Task Suppress Task Specify Report
          Extension Attribute Value
            <bpmnt> Tailoring NA
        Sub Process Replace Task Design Screen
          Sub Process Design and Validate User Interfaces
            Extension Attribute Value
              <bpmnt> Tailoring LocalReplacement
            Standard Loop Characteristics
            Task Design Screen
            Task Validate Screen
            Sequence Flow 1
        Task Suppress Sequence Flows
          Extension Attribute Value
            <bpmnt> Tailoring NA
        Sub Process Contribute with Parallel Gateways
          Extension Attribute Value
            <bpmnt> Tailoring LocalContribution
          Parallel Gateway AND_DIV
          Parallel Gateway AND_CONV
          Sequence Flow N_1
          Sequence Flow N_2
          Sequence Flow N_3
          Sequence Flow N_4
          Sequence Flow N_5
          Sequence Flow N_6
          Sequence Flow N_7
          Sequence Flow N_8
```

2

Property	Value
Suppressed Base Element	
Used Base Element	Process Specification and Design
Use Kind	Extension

3

Property	Value
Suppressed Base Element	Task Specify Report
Used Base Element	
Use Kind	NA

4

Property	Value
Suppressed Base Element	
Used Base Element	Task Design Screen
Use Kind	LocalReplacement

5

Property	Value
Suppressed Base Element	Sequence Flow 9, Sequence Flow 10, Sequence Flow 11, Sequence Flow 14
Used Base Element	
Use Kind	NA

6

Property	Value
Suppressed Base Element	
Used Base Element	Process Specification and Design
Use Kind	LocalContribution

7

Property	Value
Name	N_7
Source Ref	Task Update Data Base
Target Ref	Parallel Gateway AND_CONV

8

Figure 25. Tailoring specification for scenario 10 in Table 10 and the process diagram after tailoring.

5.6.6 Results and Conclusions

Table 9 shows the size of the tested base process BPMN model in terms of the number of tasks, subprocesses, events, gateways, and sequence flows. Table 10 shows the number of adaptations performed for each process execution case when using the BPMNt support prototype where the tailoring scenario is identified by the column ID. This table shows the number of direct and indirect adaptations regarding the process elements. Direct adaptations (contribution, replacement and suppression) are those specified explicitly by the process engineer (through tailoring relationships) when designing a tailored process model. On the other hand, indirect adaptations are those executed automatically by well-formedness rules (in Table 7 and Table 8) in an effort to ensure the validity of BPMN models after tailoring. Such adaptations primarily affect sequence flows.

Table 9. Base Process of the SIGA Project for the Specification and Design Phase

Base Process BPMN Model				
Tasks	*Subprocesses*	*Events*	*Gateways*	*Sequence Flows*
10	0	2	2	14

Table 10. Results of Evaluation of the Process Tailoring involving the SIGA Project

ID	*Contribution*			*Replacement*			*Suppression*			*Indirect Tailoring (by Rule)*				
	Tasks/ Subproc.	*Gateways*	*Sequence Flows*	*Task/ Subproc.*	*Gateways*	*Sequence Flows*	*Tasks/ Subproc.*	*Gateways*	*Sequence Flows*	*Rule 1*	*Rule 5*	*Rule 10*	*Rule 12*	*Rule 13*
1	1	-	2	1	-	-	1	-	1	2	3	2	1	1
2	-	3	10	-	-	-	1	-	5	2	3	2	-	-
3	1	3	11	-	-	-	1	-	5	2	3	2	-	-
4	-	-	-	-	-	-	5	-	-	6	7	2	-	-
5	3	2	7	-	-	-	5	-	1	6	7	2	-	-
6	-	2	11	1	2	-	1	-	6	-	1	-	3	3
7	-	2	8	1	-	-	1	-	4	2	3	2	1	1
8	1	2	10	1	-	-	1	-	4	2	3	2	1	1
9	1	-	2	-	-	-	5	-	1	6	7	2	-	-
10	-	2	8	1	-	-	1	-	4	2	3	2	1	1

As shown in Table 10, tailoring scenarios 1 to 10 allowed us to evaluate all proposed tailoring operations (contribution, replacement, and suppression) in conjunction with the main well-formedness rules (rules 1, 5, 10, 12, and 13). Moreover, all tailoring scenarios (and their specific adaptations) considered in this evaluation could be successfully specified using the BPMNt support prototype. The set of proposed tailoring operations as well as the set of BPMN process elements that currently support such operations were enough to specify the adaptations required by the tailoring scenarios tested from the SIGA Project.

However, through this experience we realized that the tailoring specification using proposed tailoring operations is laborious and time-consuming. In particular, applying contribution operations, such as the last operation specified in the model in Figure 25, was laborious and time-consuming because the modeler needs to specify how new elements are connected to the process workflow. The modeler must manually add sequence flow elements to the tailored process model (see the final part of the process structure in box 1 Figure 25) and set their properties to connect with node elements (see box 7 in Figure 25).

Thus, this evaluation allowed us to conclude that: (1) the proposed approach was expressive enough to represent process tailoring in the evaluated scenarios; and (2) the tailoring operations must be improved to facilitate the definition of adaptation scenarios, mainly contribution operations.

Finally, we conclude by answering our initial evaluation question:

- **Q:** *Is the current BPMNt extension (and related support mechanisms) able to specify software process tailoring scenarios from the SIGA Project?*
  Based on scenarios we evaluated from the SIGA Project, we can state that the BPMNt approach and its support prototype satisfactorily dealt with the adaptations needed to produce tailored software process models from the base process in this context. However, more research is necessary to extend the evaluation with additional real-world data and potentially identify more test scenarios in which our approach needs to improve, besides the "easy to use" aspect already identified in this study.

5.6.7 Threats to Validity

This evaluation is subject to four main threats:

- *Limited number of evaluation scenarios*: It is difficult to obtain data to drive research in the domain of software process modeling. Companies that adopt process modeling usually consider process artifacts extremely sensitive and confidential. We obtained access to people and artifacts from the SIGA Project, a collaborative system development project involving several Brazilian research institutions. However, because mining the artifact repository and applying compliance analysis techniques to identify variances between the defined process model and executed process models is a laborious and time-consuming effort, we could obtain just a few tailoring scenarios from the aforementioned project. Moreover, all evaluated scenarios were related to the same process phase (i.e., Specification and Design phase).
- *Data coming from a single project:* The models and tailoring scenarios used in this evaluation may not be representative of those occurring in other realistic settings. Different software development processes, domains and organizations may lead to different results. Thus, our approach should be tested further on models and tailoring scenarios from other organizations and domains.
- *Modeling of the SIGA's tailored development processes (processes actually executed by the project team):* The modeling of the tailored processes in BPMN plus BPMNt and processed by our prototype was outside the main project. This

threat was minimized by checking with a domain expert from the SIGA project that the BPMN models obtained after our modeling were correct.

Although the observed results are promising, it is important to note that in practice there may exist many other types of organization-, domain-, or even project-specific tailoring operations. Clearly, there may also exist many test cases where the context (dependencies) of the model elements affected by a tailoring operation may lead to failures uncovered by the rules proposed in this research so far.

5.7 Concluding Remarks

In order to analyze our results from different perspectives, we first consider the analysis dimensions adopted by BECKER *et al.* (2007). Such dimensions include: (1) complexity of the reuse situation, (2) repetition rate of the reuse situation, (3) cost of preparation, and (4) cost of utilization. We believe the BPMNt approach can be especially interesting in situations that involve a high complexity and low repetition rates. The complexity of the reuse situation describes how many contingency factors influence the suitability of a solution and the repetition rate measures whether the specific conditions of a reuse situation are unique or regularly reoccurring (BECKER *et al.*, 2007). In our case, the reuse context considered in this chapter is related to software processes. There are papers in the literature indicating that this process type is difficult to predict because it is influenced by several factors that are still poorly understood (CLARKE and O'CONNOR, 2012) (KALUS and KUHRMANN, 2013). The way tailoring occurs remains unclear and is, therefore, often left to experts, such as process engineers or project managers (KALUS and KUHRMANN, 2013). Therefore, software processes involve complex reuse situations and our approach can be useful in such a context because it does not require anticipating circumstances that may affect a process. BPMNt tailoring operations enable an existing process to be altered with a high degree of freedom. Moreover, we believe that many of the specific conditions that lead to the adaptation of software processes are not recurring and our approach can be useful in this context because it does not work on predefined adaptation situations.

The cost of preparation depends on how much effort is necessary before a reuse mechanism can be used (BECKER *et al.*, 2007). Our approach does not require any predefinition before applying tailoring because we do not exclude or limit certain types of alterations on models, which would need to be specified *a priori*. The only constraints applied are related to the correct specification of tailoring operations themselves. However, the BPMNt approach requires the process engineer understands the provided tailoring operations and their effects before using them. For this reason, in the next chapter we present a detailed specification of catalog for our tailoring operations, describing their effects, application rules, and use examples in which they apply to.

On the other hand, the cost of using the BPMNt approach is currently high. This cost varies based on how much the modeler is instructed when a reuse mechanism is employed, that is, it is related to the degree of guidance in adaptation (BECKER *et al.*, 2007). Process tailoring using the BPMNt support prototype is performed in an ad-hoc manner by the process engineer, who selects which actions to apply and when. To improve guidance in adaptations, in the next chapter we expand the set of rules associated with tailoring operations, thus avoiding the development of incorrect tailored process models.

We also acknowledge that tailoring specification using our SPEM-based tailoring operations is laborious and time-consuming, since for some operations the modeler needs to specify several basic (or primitive) operations (e.g., add task, add sequence flow 1, add sequence flow 2, etc.) to obtain the same result of a single high-level operation such as the addition of a new process element to a specific workflow position. It is also difficult for the modeler to predict *a priori*, before the tailoring interpretation, how a set of adaptation operations interact with each other and how they modify the final tailored process when a set of basic operations needs to be applied to achieve each user's adaptation intention. That is, a set of basic (or primitive) operations are difficult for humans to interpret. For these reasons, in the next chapter we propose to evolve tailoring operations currently supported (SPEM-based operations) for a higher level of abstraction, which can represent each user's adaptation intention through a single tailoring operation, also decreasing the possibility of errors in the final tailored process model. For example, flow breaks will not occur anymore. We believe that the use of high-level operations will facilitate the task of specifying process adaptations as well as the interpretation by humans of performed changes.

CHAPTER VI

6. Pattern-Based Tailoring Support

6.1 Introduction

Process adaptations can be performed based on a set of primitive operations such as our SPEM-based operations presented in the previous chapter, e.g., *contribute* with new task x, *suppress* sequence flow 5, *contribute* with sequence flow N_1, and *contribute* with sequence flow N_2. Following this approach, a particular adaptation (e.g., to insert a new task) usually requires the application of multiple primitive operations. However, as discussed in the previous chapter, specifying process adaptations at this low level of abstraction is a complex and error-prone task (WEBER *et al.*, 2008) (PILLAT *et al.*, 2015). Moreover, when applying a single primitive operation, the correctness of the resulting process model (e.g., absence of flow breaks) cannot be ensured. It is not possible to associate pre- and post-conditions related to semantic rules of the BPMN language (e.g., well-formedness rules in Appendix 1) with the application of primitive operations. Instead, correctness of a process model has to be explicitly checked after applying a set of primitives.

On the other hand, adaptations on process models can be based on high-level adaptation operations (e.g., to insert a process fragment between two tasks), which abstract from the process model transformations to be conducted. Instead of specifying several primitive operations, the process engineer can apply a smaller number of high-level adaptation operations to accomplish a same desired adaptation. Moreover, pre- and post-conditions associated with high-level operations can guarantee correctness when applying such operations. Still, high-level operations are more easily understood (LANGER et al., 2013) (WEBER *et al.*, 2008), since their semantics is directly related to certain adaptation intentions.

Thus, from a deeper analysis of adaptations required to represent tailoring scenarios in BPMN, considering use cases from both SPE and BPM domains, as well as limitations identified for the SPEM-based operations (presented in the previous chapter), in this chapter we expand our tailoring solution by providing high-level operations as a cohesive set of atomic operations that are applied together to achieve a user's adaptation intent. High-level operations intend to improve the understandability of operation semantics and avoid modeling errors on the tailored process model.

6.2 High-Level Tailoring Operations from Patterns

The BPMN*t* high-level operations consider adaptations of BPMN process models regarding the process' *control-flow* perspective and all they were designed to ensure the structural correctness (i.e., absence of flow breaks) and well-formedness of the adapted process model regarding control-flow's semantic rules of the BPMN specification (presented in Appendix 1). For preventing structural fails in the model, it is essential to abstract the user from specific model transformations, such as redefining workflow edges.

We derived our high-level operations based on definitions of *adaptation patterns* proposed by WEBER, REICHERT and RINDERLE-MA (2008) and *refinement patterns* proposed by BRANCO *et al.* (2014). In order to make clear our contributions regarding such works, in Table 11 we summarize the main characteristics of our tailoring operations in comparison to characteristics of *adaptation patterns* and *refinement patterns.* Next, we describe in more details the differences between works.

Table 11. Comparison between BPMN*t* and adaptation/refinement patterns

Adaptation Patterns	BPMN*t*
- Generic definition (independent of language) of high-level adaptation operation patterns for control-flow	- High-level adaptation operations specialized for BPMN (supported as standard-compliant extension concepts) - Additional high-level adaptation operation specific for BPMN
- Do not define application rules (related to model correctness)	- Defines operation pre- and post-conditions (related to structural correctness and well-formedness of BPMN models)
-	- Built-in change traceability mechanism
Refinement Patterns	**BPMN*t***
- Definition of refinement patterns for the Business-TI context (from business model for technical-level model) - Patterns do not correspond to high-level	- Derivation of high-level refinement operations (supported as standard-compliant extension concepts)

operations	
- Do not define application rules (related to model correctness)	- Defines operation pre- and post-conditions (related to structural correctness and well-formedness of BPMN models)
-	- Built-in change traceability mechanism

The research conducted by WEBER, REICHERT and RINDERLE-MA (2008) has identified diverse control-flow *adaptation patterns* for business processes from the analysis of real-world process models. Each pattern corresponds to exactly one high-level operation (e.g., to insert a task in parallel to another one) that has been observed at least three times in different models. However, such as the definition of a pattern requires, these high-level operations are presented as generic solutions, described in high-level, and independent from any process meta-model. Thus, as stated by the authors, the realization of these operations for specific modeling languages requires identifying and associating pre- and post-conditions to each operation in order to ensure the correctness of the resulting model. More specifically, these conditions need be related to the meta-model of a process language to enable automation of the operations.

Therefore, we adapted these generic operation definitions according to the BPMN meta-model specification and defined constraints for application of each operation (pre-conditions) as well as action rules (post-conditions), which evaluate the state of the model after the adaptation and execute certain actions to ensure its correctness. For example, consider the simple tailoring scenario from Figure 26, which shows a tailored process on the right part and its BPMN base process on the left part. In this scenario, the high-level adaptation operation *Parallel Insert* was applied to insert the new task *Check credit history* in parallel to the existing task *Check income sources.* This operation can be applied since the resulting model is valid. Constraints of pre-conditions for this operation determine, for example, that no insertion can be performed in parallel to the start or end events of the process. As post-conditions associated to this operation, specific control connectors of the BPMN language for diverging and converging parallel flows (i.e., Parallel Gateways) are automatically inserted by the operation before and after the parallelized tasks (see model on the right part of Figure 26). In this way, high-level operations allow focusing on the adaptation intention, abstracting structural configuration details of the model, and guarantee its structural correctness by keeping all connected workflow nodes as well as preventing the modeling of errors concerning semantic rules of BPMN. For example, the model on the left part of Figure 27 violates the BPMN well-formedness rule #16 in Appendix 1, which prescribes "*a Parallel Gateway (AND) joins only non-exclusive Sequence Flows*". In fact, the violation of this rule leads to a case of local deadlock, such as in the example on the left part of the figure. Similarlly, the model on the right part of Figure 27 violates the BPMN well-formedness rule #17 in Appendix 1, which prescribes "*a join Exclusive Gateway (XOR) must merge only exclusive Sequence Flows*". The violation of this rule leads to a case of lack of synchronization, such as in the example on the right part of the figure.

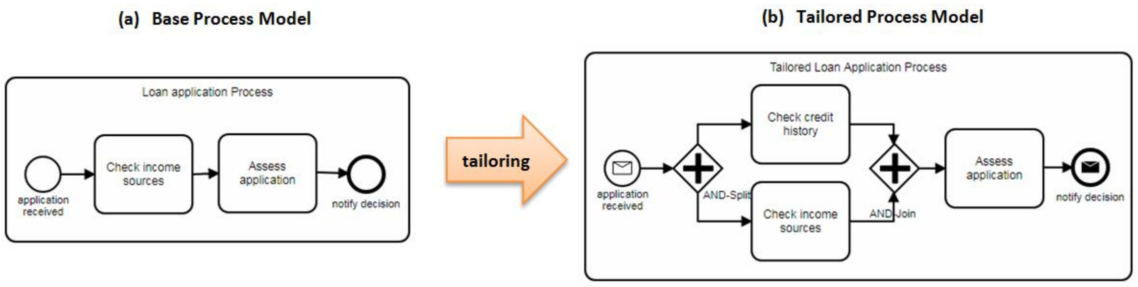


Figure 26: Example tailoring scenario

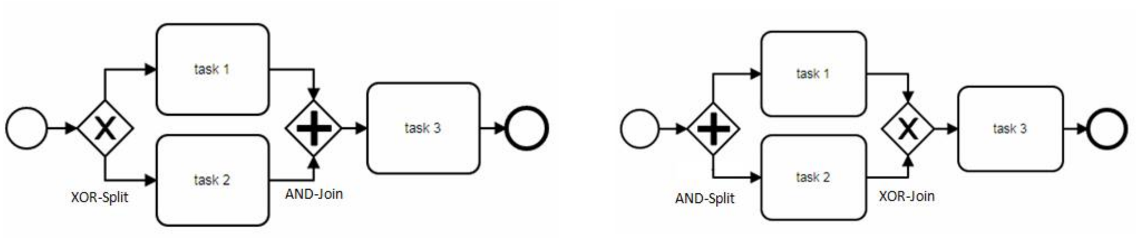


Figure 27. Modeling errors: Local deadlock (left) and lack of synchronization (right)

It is also important to highlight that due to the generality purpose of the adaptation patterns, they only consider the most common types of flow variations (enabled by basic control connectors), which are supported by any process modeling language. However, BPMN offers other possibilities of flow control (e.g., based on the occurrence of events), which are found in many process models designed with this language and can also be necessary when adapting its process models. Thus, in order to support such possibilities, we also extended the set of operations defined by *adaptation patterns* (WEBER, REICHERT and RINDERLE-MA, 2008).

The adaptation operations mentioned above have as main focus to adapt the control-flow of a process, generally implying in significant changes in its behavior. However, in many scenarios of process adaptation/tailoring, the most of the performed adaptations correspond to refinements, in which the flow of the process and its elements are changed in a way that does not essentially change the process' normal behavior (e.g., if a high-level activity is refined into a sequence of low-level activities or into a subprocess with the same input/output behavior). In this sense, the research conducted by BRANCO *et al.* (2014) has identified different *refinement patterns* applied when producing technical-level process models from business-level process models. However, such patterns do not represent necessarily a high-level change operation and neither have any concern related to model correctness. Thus, we analyzed their descriptions and examples for deriving refinement operations in our solution. Such as for adaptation operations, we were concerned with the correctness of the resulting process model, then we also defined for these operations a set of pre- and post-conditions in order to prevent problems in the resulting model. However, in this case, most of the conditions are not concerned on ensuring model structural correctness, but on its well-formedness concerning semantic constraints of the BPMN specification (Appendix 1). For example, in the tailoring scenario represented in Figure 26, two refinement operations *Specialize* were applied to the start and end event of the reused base process. As a result, the original plain events were changed to message events (see right part of the figure). For this operation, we defined pre-conditions (well-formedness rules) that verify if the specialized element type provided for the original element is valid according to the BPMN specification. A process' start event, for instance, can only be specialized to one of the following types: Message, Timer, Conditional, or Signal (in accordance with rule #1 from Appendix 1).

It is also important to mention that, as the SPEM-based extension proposed in the previous chapter, the BPMN*t* extension presented in this chapter also aims at being totally compliant with the standard extension mechanism of BPMN.

Therefore, considering our proposed solution structure, we present in Figure 28(b) a schematic representation of the Pattern-based BPMN*t* tailoring specification for the example in Figure 26 by using high-level operations. The new process elements are specified in the *Tailored Process Model* and each of these elements has an associated tailoring operation (represented as a dotted arrow on the top right corner of the element) that defines how the new element is related to its *Base Process Model*. The interpretation of this tailoring specification is presented on the right part of the figure.

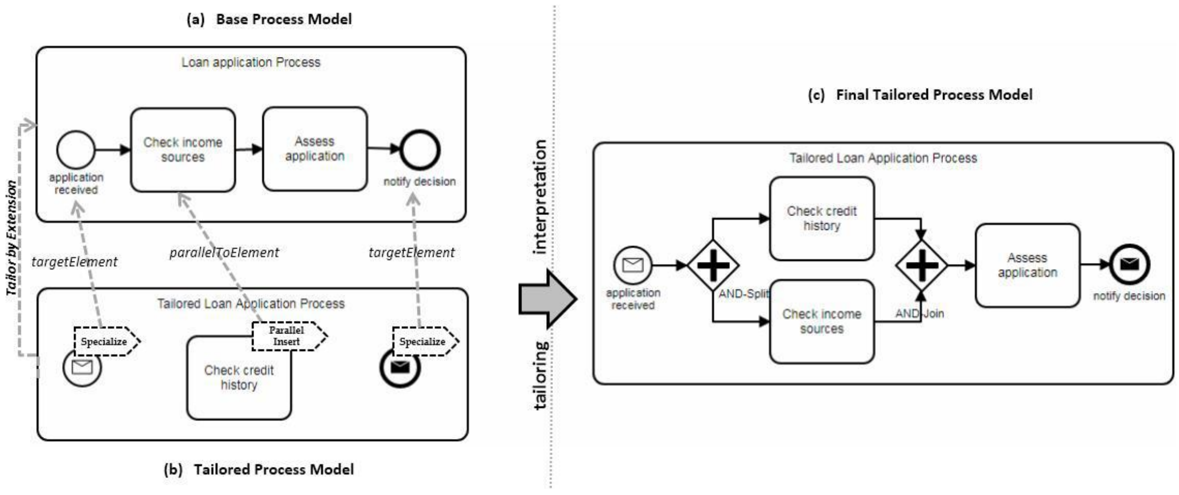


Figure 28: BPMN*t* tailoring specification based on high-level operations

Finally, in Table 12 we present our BPMN*t* high-level tailoring operations and patterns from which they were derived. As one can observe from the table, operations for removal and addition of activities (or process fragments) are present in both sets of patterns (adaptation and refinement). For example, our operation *Delete* was derived from an operation of same name from adaptation patterns and an operation named *Suppress* from refinement patterns. BPMN*t* operations *Delete*, *Replace*, *Move*, and *Parallelize* have the same semantics of their corresponding patterns and their detailed specifications are presented in Appendix 2.

We opted to provide three different operations related to insertion of elements: *Serial Insert*, *Conditional Insert*, and *Parallel Insert*. Our intent was to make explicit the way in which a given process fragment is inserted in the workflow reused from the base model. Moreover, this decision allowed us to include operation configuration parameters customized for each case. In special, the *Conditional Insert* operation (specified in Figure 55 from Appendix 2) includes a parameter called *condition* that indicates when the new fragment will be performed in the process. We also adapted this operation for supporting both exclusive (XOR) and inclusive (OR) inserts. To this end, an additional binary parameter was included in the operation definition, which aims at informing if the inserted conditional fragment can be performed in parallel to another alternative conditional fragment. In affirmative case, the operation will automatically include a BPMN Inclusive gateway (OR) before the inserted fragment to diverge the process flow and another gateway of same type after the inclusion to converge again the flow. Otherwise, the operation will include Exclusive gateways (XOR) before and after the inserted fragment.

We also considered a special type of insertion, supported by the *Event-based Insert* operation (specified in Figure 56 from Appendix 2), which has not been derived from any pattern. This operation enables insertion of a conditional process fragment that is executed only when a given event (of message, signal, time, or condition) occurs before other alternative events. This type of flow variation is specific of BPMN, supported by a particular gateway of the language (named Event-Based Gateway), and can be found in many models designed with this language. Thus, we opted to create a specific operation for supporting this type of flow variation of the language. Such as other BPMN*t* tailoring operations, this one also has associated rules (pre- and post-conditions) that avoid inconsistencies in the adapted process model. With this objective, the operation automatically adds in the adapted process the control gateway for this case as well as the event that determines the activation of the inserted process fragment by the user. The specification of this operation enforces the well-formedness rules #21, #24, #25 and #26 from Appendix 1, which are related to the use of *Event-Based Gateway*.

BPMN*t Encapsulate* and *Split* operations have the same semantics of their corresponding patterns. However, we provide a single *Split* operation that supports two much related refinement patterns. At each use of the operation, one can choose which specific split pattern will be applied by configuring a parameter of the operation.

The BPMN*t Merge* operation does not have a directly related pattern. Its necessity was observed from scenarios exemplifying refinement patterns (BRANCO *et al.*, 2014). The authors relate use cases of this operation to the pattern *Suppress Specification Activity*. However, we considered this an inadequate relation, since the *Suppress* pattern does not adequately represent the intent of the performed action.

Our *Rename* operation corresponds to a change of element name, but also enables update of conditions related to process flows in conceptual models. In this model type, flow conditions are represented informally through the name property of sequence flows (edges). We do not provide a specific operation for such an update because our solution does not intent to generate executable models. This type of model requires detailed configuration and formalization of various specific properties that are out of the scope of our solution. Technical-level models derived with the BPMN*t* extension and support infrastructure are still conceptual models, which need further refinement in order to be enacted in a BPM system.

The BPMN*t Specialize* operation corresponds essentially to a change in the type of an element (i.e., its specialization), such as the first related refinement pattern.

However, since in many use cases of this pattern the element name is also changed, we opted to provide an operation for change of type that optionally can also change the element name.

Finally, BPMN*t* operations *Add Exception Handler* and *Add Exception Flow* were derived for performing refinement actions related to exception handling while still resulting in a correct BPMN model. The refinement patterns related to these operations do not match to a complete adaptation intention (i.e., a high-level operation). For instance, the pattern *Add Boundary Event* (BRANCO *et al.,* 2014) does not correspond to a high-level operation, since the addition of a boundary event that catches an exception without indicating a handler for such exception does not make sense. This behavior, actually, violates the well-formedness rule #14 (Appendix 1) of the BPMN specification that prescribes "*a Boundary Event must have exactly one outgoing Sequence Flow (unless it has the Compensation type)*".

Table 12. Derivation of BPMN*t* tailoring operations from patterns

BPMN*t* High-Level Tailoring Operation	Adaptation (AP) or Refinement (RP) Pattern
Delete	AP: Delete Process Fragment RP: Suppress Specification Activity
Replace	AP: Replace Process Fragment
Move	AP: Move Process Fragment
Parallelize	AP: Parallelize Activities
Serial Insert	AP: Insert Process Fragment RP: Add Script Task RP: Add Protocol Task
Conditional Insert	AP: Insert Process Fragment
Parallel Insert	AP: Insert Process Fragment
Event-based Insert	---
Encapsulate	AP: Extract Process Fragment to Sub-Process
Split	RP: Split Task into Block RP: Split Workflow
Merge	RP: Suppress Specification Activity
Rename	RP: Change Activity Name AP: Update Condition
Specialize	RP: Change Activity Type RP: Change Activity Name
Add Exception Handler	RP: Add Boundary Event RP: Add Technical Exception Flow
Add Exception Flow	RP: Add Boundary Event

	RP: Add Technical Exception Flow

6.3 Catalog of BPMN*t* Tailoring Operations

In order to facilitate the correct understanding of our high-level operations, we have defined them in a catalog (Appendix 2). We present this catalog of high-level tailoring operations for BPMN providing the following information about each operation (e.g., see Figure 29): purpose, motivation, description, source element, list of parameters, pre- and post-conditions (correctness-related rules), a schematic use representation, related patterns, and a use example.

In order to apply the proposed operations on a BPMN process model, it must meet some constraints that we describe in the following. A base process model must be well-structured. A model is well-structured if for every node with multiple outgoing edges (a split) there is a corresponding node with multiple incoming edges (a join), and vice versa, such that the fragment of the model between the split and the join forms a single-entry-single-exit (SESE) component (POLYVYANYY *et al.*, 2017) (DUMAS *et al.*, 2010). Any split or join of the process flow must be represented through a BPMN *Gateway* of type diverging or converging, respectively. That is, tasks, subprocesses, or events must not have multiple input or output flows. When it is necessary to converge or diverge the process flow, a gateway must be used. Moreover, the model must have its input point and output point explicitly represented through *Start* and *End Events*, respectively. All other process nodes (tasks, subprocesses, gateways, or events) must be preceded and succeeded by at least one node.

In the description of the catalog of BPMN*t* operations, a *process element* generally corresponds to objects from subclasses of BPMN *FlowNode*. That is, a *Task, Subprocess, Event,* or *Gateway*. However, for the operation *rename*, a process element can correspond to objects of subclasses of BPMN *FlowElement*. In these cases, in addition to the *FlowNode* elements, a process element can also refer to a *Sequence Flow* or *Data Object*. A *process fragment*, in turn, corresponds to a sub-graph (containing *FlowNode* elements directly connected through sequence flows) with single entry and single exit node.

Our operation catalog for tailoring BPMN-based processes focusing on the control-flow perspective includes the operations:

1. ***Extend***: Reuses the elements structure of an existing process model to derive a new one and enables the use of tailoring operations (complete definition in Figure 49);
2. ***Delete***: Removes a process element or fragment from the reused base process (complete definition in Figure 50);
3. ***Replace:*** Replaces a process element or fragment from the reused base process by another element or fragment (complete definition in Figure 51);
4. ***Move:*** Shifts a process element or fragment from its current position in the base process to another position within the variant process (complete definition in Figure 52);
5. ***Parallelize:*** Enables the concomitant (concurrent) execution of elements from a process fragment which has been defined as sequential in the base process (complete definition in Figure 53);
6. ***Serial Insert:*** Adds a new process element or fragment between two directly succeeding elements of the reused base process (complete definition in Figure 54);
7. ***Conditional Insert:*** Adds to the reused base process a conditional process element or fragment which is executed only when a given condition (situation) is true (complete definition in Figure 55);
8. ***Parallel Insert:*** Adds to the reused base process a new process element or fragment that is executed *while* another element or fragment is also executed (complete definition in Figure 29);
9. ***Event-based Insert:*** Adds to the reused base process a conditional process fragment that is executed only when a given event (of message, signal, time, or condition) occurs before other alternative events. The occurrence of this event cancels the others, i.e., event-based alternatives are mutually exclusive (complete definition in Figure 56);
10. ***Encapsulate:*** Encapsulates a process fragment with related activities into a separate subprocess (complete definition in Figure 57);
11. ***Split:*** Splits a single task from the base process to a process fragment or subprocess that details its procedure in the variant process (complete definition in Figure 58);
12. ***Merge:*** Merges two or more directly succeeding tasks into a single particular task (complete definition in Figure 59);

13. ***Rename**:* Allows to change the name of a reused process element (complete definition in Figure 60);
14. ***Specialize:*** Changes the type (and optionally the name) of a basic process element (e.g., plain tasks) for a more specific one (complete definition in Figure 30);
15. ***Add Exception Handler:*** Adds an exception handler (task or subprocess) to deal with a given type of exception event that can occur in the context of one or more task(s) or subprocess(es) (complete definition in Figure 61);
16. ***Add Exception Flow:*** Adds an exception flow to deal with a given type of event that is triggered and handled by activities from the base process (complete definition in Figure 62).

Given the large number of high-level operations that comprise our catalog, in this section we present the complete definition of only two of them: *Parallel Insert* and *Specialize*. The former represents an example of BPMN*t* operation that has been derived from the set of *adaptation patterns* (WEBER, REICHERT and RINDERLE-MA, 2008) whereas the second operation exemplifies a case of derivation from the set of *refinement patterns* (BRANCO *et al.*, 2014). The complete definition of other high-level operations that comprise our catalog are presented in Appendix 2 (Figure 49 to Figure 62).

Figure 29 shows the definition of the BPMN*t Parallel Insert* operation and Figure 30 shows the definition of the *Specialize* operation. Both operations have already been exemplified in the previous section for inserting a new task in parallel to another existing one and for changing the type of events, respectively, in the process models presented in Figure 26 and Figure 28.

The *Parallel Insert* operation (Figure 29) was derived from the adaptation pattern *Insert Process Fragment* (presented in Figure 9), and the *Specialize* operation (Figure 30) was essentially derived from the refinement pattern *Change Activity Type* (presented in Figure 10). Comparing definitions of these BPMN*t* operations with definitions of the patterns that originated them, one can observe that our operation definitions are significantly more detailed than their original patterns. They explicitly define parameters provided by each operation, all of them related to BPMN meta-model's elements, and, in special, stand out for presenting rules of pre- and post-conditions associated to each high-level operation, which ensure its correct use and the validity of the resulting process model concerning semantic rules of the BPMN specification.

PARALLEL INSERT

Purpose: Insert a new process element or fragment that is executed *while* another task or fragment is also executed.

Motivation: In a specific process, a task (or fragment) that has not been modeled in the base process needs to be performed and it can be executed *while* another task (or fragment) is executed.

Description: A variant process defines a process element or fragment X which is inserted in the reused base process workflow in parallel to the single element indicated by the parameter *parallelToElement* **or** in parallel to the process fragment contained between the elements indicated by *after* and *before*.
The parameter *additionIsFragment* is relevant only when this operation is defined to a subprocess (source element X). When its value is *true,* it indicates that the content of the source subprocess of the operation represents a process fragment to be inserted directly in the workflow of the reused base process. Otherwise, the source subprocess itself will be inserted in the target process' workflow.

Source Element Type(s): Task, Subprocess, or Event.

Parameters: parallelToElement(BPMN:FlowNode), after(BPMN:FlowNode), before(BPMN:FlowNode), additionIsFragment(boolean = false).

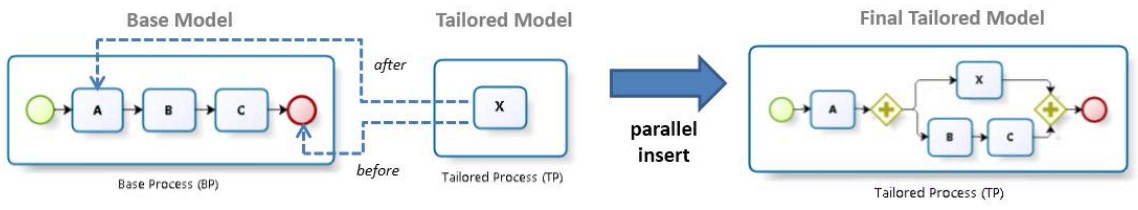
Pre-conditions:

1. The parameter *parallelToElement* must not point to a start or end event or a gateway;
2. Parameters *after* and *before* must *not* point to directly succeeding elements;
3. The base process fragment between the parameters *after* and *before* cannot contain incomplete flow branches;
4. Gateways cannot be inserted singly;
5. Process fragments containing incomplete flow branches cannot be inserted;
6. If source element of the operation is a Subprocess, its content cannot violate any rule in Appendix 1;
7. If source element of the operation is an Event, it must be an intermediate event of specific type (None, Message, Timer, Escalation, Conditional, or Signal).

Post-conditions:

1. A BPMN parallel gateway is included soon after the element pointed by the parameter *after* (or soon before the element pointed by *parallelToElement*) to diverge the process flow and another parallel gateway is included soon before the element pointed by the parameter *before* (or soon after the element pointed by *parallelToElement*) to converge again the process flow;
2. Sequence flows are adjusted to connect to the inserted elements.

Representation:



Related Pattern: Insert Process Fragment (WEBER, REICHERT and RINDERLE-MA, 2008).

Example: In a check-in process (AYORA *et al.*, 2015), a variant of the traditional process it is necessary when the passenger is an unaccompanied minor. In this case, the new task *Print Duplicated Boarding Card for the Relative* is performed in parallel to the base process' task *Drop off Regular Luggage*.

Figure 29. Tailoring operation *Parallel Insert*

SPECIALIZE

Purpose: Change the type (and optionally the name) of a basic process element (e.g., a plain task) for a more specific type.

Motivation: The type of a process element can be changed due to a technical specification decision. According to BRANCO *et al.* (2014), it is easier for business people to stick with basic modeling elements (such as plain tasks), while other types of elements are more suitable to implement the business intent.

Furthermore, elements are sometimes renamed to better reflect some technical aspects.

Description: A variant process defines a specific type of process modeling element that specializes/implements the base process element identified by the parameter *targetElement*. If a name is provided for this element, it will replace the name of the target element.

Source Element Type: Task, or Event.

Parameters: targetElement (BPMN:FlowNode).

Pre-conditions:

1. Parameter *targetElement* CANNOT point to gateways;
2. Parameter *targetElement* CANNOT point to a Start Event of a base process' Subprocess (ref. rule 2 in Appendix 1);

<u>If target element is a Task:</u>

3. If parameter *targetElement* points to a Task, source element of the operation MUST be a task of specific type (i.e., *SendTask, ReceiveTask, ServiceTask, UserTask, ManualTask, ScriptTask* or *BusinessRuleTask*) <u>or</u> an event of specific type *Message* or *Signal;*
4. If parameter *targetElement* points to a Task, source element of the operation CAN be a Start Event ONLY IF predecessor element of *targetElement* is a Start Event;
5. If parameter *targetElement* points to a Task, source element of the operation CAN be an End Event ONLY IF successor element of *targetElement* is an End Event;

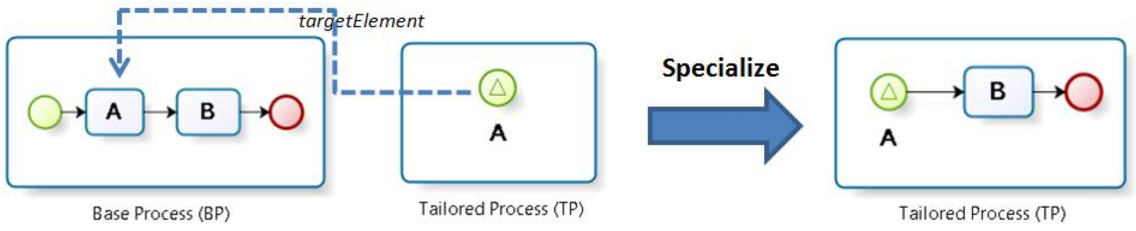
<u>If target element is an Event:</u>

6. If parameter *targetElement* points to the base process' Start Event, then source element of the operation MUST be a Start Event of specific type Message, Timer, Conditional, or Signal;
7. If parameter *targetElement* points to an End Event, then source element of the operation MUST be an End Event of specific type Message, Error, Escalation, Signal, or Terminate;
8. If parameter *targetElement* points to an Intermediate Event, then source element of the operation MUST be an Intermediate Event of specific type Message, Timer, Escalation, Conditional, or Signal;

Post-conditions:

1. If target element is a Task <u>and</u> source element of the operation is a Start Event, then the start event that precedes the target element in the base process will be removed from the tailored process;
2. If target element is a Task <u>and</u> source element of the operation is an End Event, then the end event that succeeds the target element in the base process will be removed from the tailored process;
3. If target element has other elements linked to it (e.g., boundary events), then these elements must be re-linked to the substitute element.

Representation:



Related Pattern: Change Activity Type; Change Activity Name (BRANCO et al., 2014).

Example: In a technical-level claim handling process, the plain task *Validate Claim* (from the business-level process) has been specialized by a task of type *BusinessRuleTask* (see Figure 37 and Figure 38), which indicates it has some associated business rule. In technical-level process models, the use of specialized tasks is a common practice when using BPMN.

Figure 30. Tailoring operation *Specialize*

In the next section, we demonstrate how operations from our catalog can be used in different application contexts to represent some tailoring scenarios, including scenarios from the SIGA Project that has already been considered in Chapter 5.

6.4 Use Cases applying High-Level Operations

In this section, we use tailoring scenarios from two different domains, Software Process Engineering (SPE) and Business Process Management (BPM), to demonstrate the application of high-level tailoring operations presented in the previous section.

6.4.1 Tailoring Scenarios from the SPE domain

In this section, we use two tailoring scenarios of the SIGA Project (described in Chapter 5), which apply most of the control-flow adaptation operations, and another scenario from the technical literature of the area that adapts a Requirements Engineering process for supporting activities of MDD (Model-Driven Engineering). This last one is used for exemplifying operations that have not been applied in the processes from the SIGA Project.

6.4.1.1 Academic Management Process (SIGA Project)

Tailoring scenarios from the SIGA project correspond to changes performed on the base process model of the project to represent specific execution cases. In this context, performed changes modify significantly the process structure, corresponding to the application of control-flow adaptation operations. Figure 31 presents the base process model of the project SIGA and the first tailoring scenario, which has already been described in Section 5.6.5. This scenario now applies operations *extend, delete, split, parallelize,* and *move* on the base process model. Figure 31 represents tailoring specification in a tailored process of schematic way (on the bottom) in order to facilitate the understanding of performed adaptations. Tailoring operations are illustrated in the figure by a dotted arrow on the top right corner of BPMN elements. Tailoring operations shown in the figure are applied on the base process model following the order from left to right, which represents the order in which the process engineer specifies such operations in the tailored process model. The operation specification order is important to obtain the intended result, since an incorrect order can cause conflicts (e.g., when an operation removes a process element that is referred by another subsequent) or result in a different semantically process model from the expected one. In this case, the application order of the operations is determined by the user due to the configuration of the parameter *applicationOrder* of the *Extend* operation (number 1 in

Figure 31), which establishes the reuse of the base process' structure and enables tailoring operations. This parameter was set to the value *Free*, meaning the application order follows the same in which operations were specified in the tailored process model.

Numbers associated to Figure 31 and Figure 32 trace correspondences between the specification of tailoring operations in the tailored process model (Figure 31) and their results after interpretation in the final tailored process model (Figure 32). The following operations are applied in the tailoring scenario of Figure 31 to obtain the final tailored process in Figure 32:

1. **Extend** process Specification and Design: This operation specifies the reuse relationship, enables tailoring operations, and determines the criterion for application order of these operations.
2. **Delete** task *Specify Report*: Removes this task from the final tailored process and the two exclusive gateways that now have only one input flow and one output flow (operation post-condition 2); also connects task *Review Use Case Description* to *Design Screen* and *Define Test Cases* to *Elaborate Physical Model* (operation post-condition 3).
3. **Split** task *Design Screen*: Replaces this task from the base process by the subprocess of same name that details its procedure in the tailored process (see region 3 in Figure 32).
4. **Parallelize** tasks *Review Use Case Specification*, *Define Test Cases*, and *Elaborate Physical Model:* A parallel gateway is included before these tasks to diverge the process flow and another parallel gateway is included after the tasks to converge (synchronize) again the process flow (post-condition 1) as well as sequence flows are adjusted (post-condition 2) (see regions of number 4 in Figure 32).
5. **Move** *Update Database:* After the interpretation of the *Parallelize* operation (described above), *Update Database* is executed only after the synchronization of parallelized tasks. However, it should be performed soon after the task *Elaborate Physical Model*. For this reason, the operation *move* shifts the task (identified by the operation parameter *movedElement*) to this new position (indicated by the operation parameter *newPositionAfter*) (see region 5 in Figure 32).

Comparing the tailoring specification using our SPEM-based operations presented in Chapter 5 (Figure 25) with the tailoring specification based on high-level operations presented in this section, one can perceive that using the new operations the adaptation intention of the modeler is more evident, since the name of these operations is more fine-grained. Moreover, the number of manual specifications required from the modeler is significantly decreased, e.g., he/she does not need anymore to specify each sequence flow connecting inserted elements to the process workflow as well as does not need to specify gateways that diverge and converge process flows, because they are automatically created during the interpretation of high-level operations.

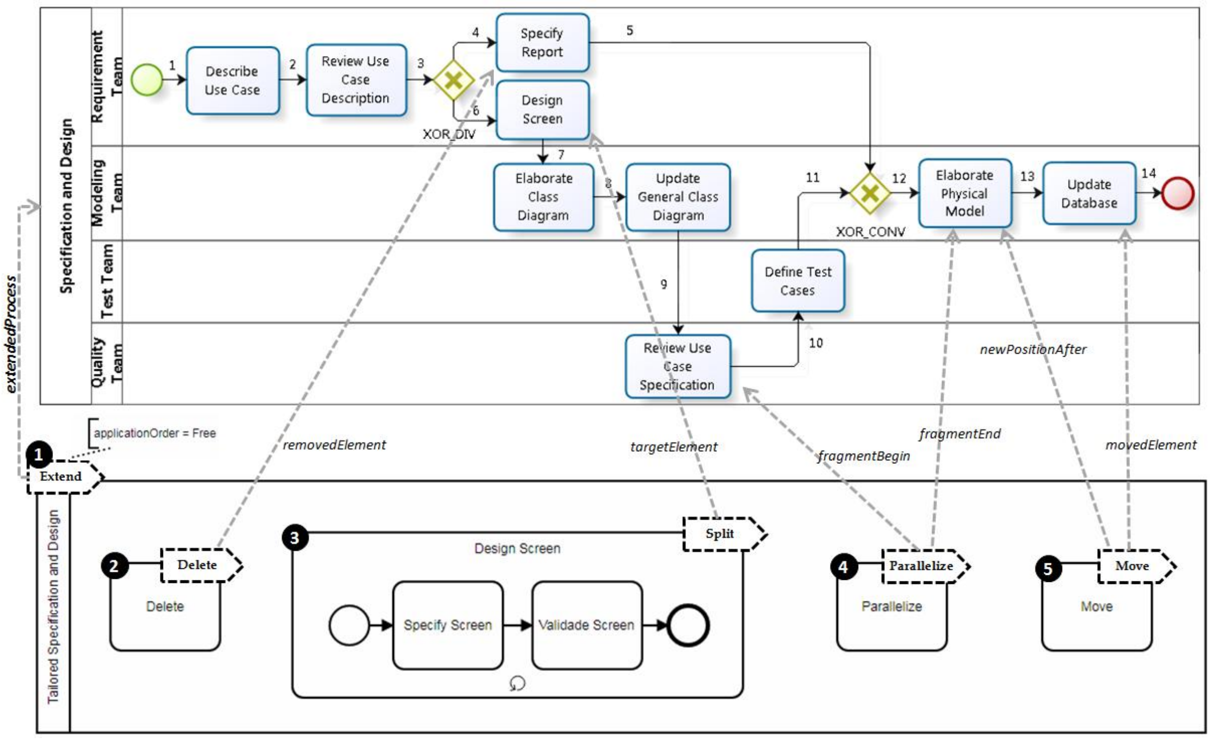


Figure 31. Scenario 1: Base process of the SIGA Project (top) and schematic specification of tailoring (bottom)

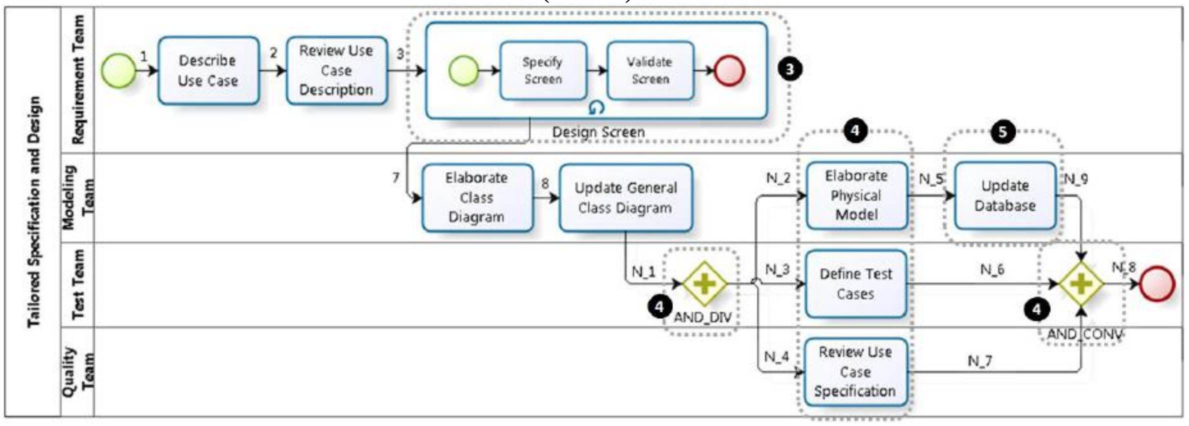


Figure 32. Scenario 1: Final tailored process

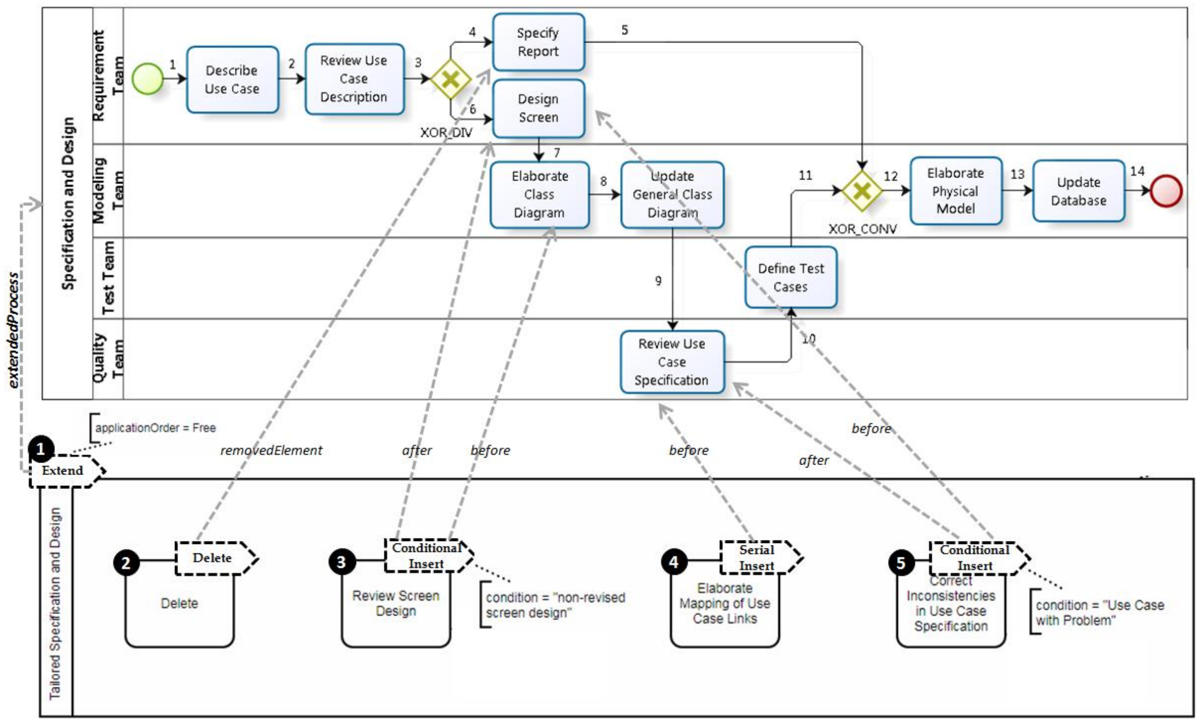


Figure 33. Scenario 2: Base process of the SIGA Project (top) and schematic specification of tailoring (bottom)

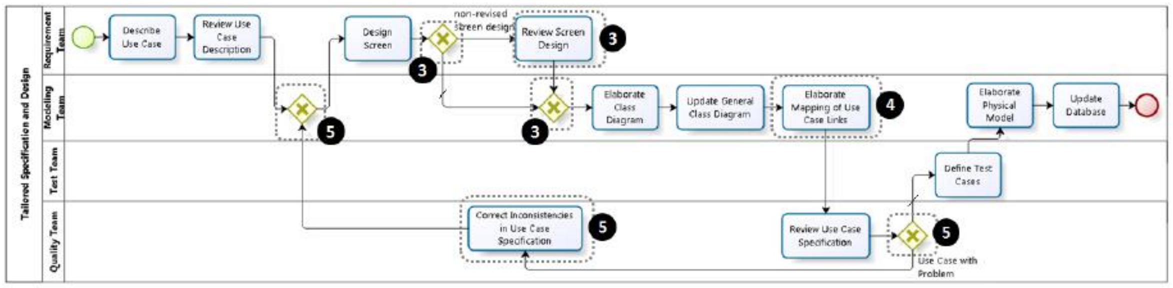

Tailored Specification and Design
Requirement Team
Modeling Team
Test Team
Quality Team
Describe Use Case
Review Use Case Description
Design Screen
non-revised screen design
Review Screen Design
Elaborate Class Diagram
Update General Class Diagram
Elaborate Mapping of Use Case Links
Elaborate Physical Model
Update Database
Define Test Cases
Correct Inconsistencies in Use Case Specification
Review Use Case Specification
Use Case with Problem

Figure 34. Scenario 2: Final tailored process

Figure 33 shows another tailoring scenario from the SIGA Project, which demonstrates the use of other operations of the catalog proposed in the previous section. Again, numbers trace correspondences between the specification of tailoring operations in Figure 33 and their results in the final tailored process model (Figure 34). The following operations are applied in the tailoring scenario of Figure 33:

1. **Extend** process Specification and Design: This operation specifies the reuse relationship, enables tailoring operations, and determines the criterion for application order of these operations (free).
2. **Delete** task *Specify Report*: Removes this task from the final tailored process as well as the two exclusive gateways; also connects task *Review Use Case Description* to *Design Screen* and *Define Test Cases* to *Elaborate Physical Model*.

3. **Conditional Insert** of task *Review Screen Design*: This task should be executed only if the screen project produced by the previous task has not been revised yet. Therefore, the execution of this task is conditioned to the previous situation. For this reason, we have used an operation *Conditional Insert* to add it between the tasks *Design Screen* and *Elaborate Class Diagram* of the base process. This operation inserts the task to the process workflow between one pair of exclusive gateways, as determines the operation post-condition 1 (see regions 3 in Figure 34). If the condition associated to the task is true (i.e., non-revised screen design), it is executed. Otherwise, the process only skips the execution of this task and follows to the next one, i.e., task *Elaborate Class Diagram*. The operation post-condition 4 sets the alternative flow (that skips the execution of the added task) as default path (for the case of no flow condition to be true).
4. **Serial Insert** of task *Elaborate Mapping of Use Case Links*: Adds this task to the final tailored process between directly succeeding tasks *Update General Class Diagram* and *Review Use Case Specification*, adjusting sequence flows of the process (see region 4 in Figure 34).
5. **Conditional Insert** of task *Correct Inconsistencies in Use Case Specification*: This task should be executed only if problems are found in a use case specification during the previous task. When its condition is true, the new task is executed. However, unlike the previous operation *Conditional Insert*, this one has its parameter *inLoop* set to true. This means that the new task must be inserted into a conditional loop that returns the process execution flow to a previous step. In this case, after the task *Review Use Case Specification* the process flow returns to soon before the task *Design Screen* (see regions of number 5 in Figure 34). Again, post-condition 4 sets the existing flow for *default*.

In this second tailoring scenario, again the modeler does not have to worry about adjusting process' sequence flows after adaptation operations or manually inserting gateways to diverge and converge the process flow after operations *Conditional Insert*. In Figure 34, gateways highlighted by dotted boxes are automatically generated during interpretation of operations *Conditional Insert*. These characteristics of high-level operations help the modeler to abstract model's configuration details and focus on the adaptation itself.

6.4.1.2 Model-Driven Requirements Engineering Process

LONIEWSKI, ARMESTO and INSFRAN (2011) proposed an adaptation of OpenUP-based Requirements Engineering process to incorporate processes of Model-Driven Development (MDD). According to authors, the adapted process model can be useful for software engineers who need to guide software development projects following an MDD approach from the requirements elicitation.

Figure 35 presents the base process model of this scenario modeled in BPMN (on the top) and a schematic representation of its tailoring specification by using our BPMN extension (on the bottom). This scenario applies operations *Encapsulate* and *Serial Insert*. The operation *Encapsulate* promotes a hierarchical restructuration by separating existing process parts into a subprocess. According to KÜSTER *et al.* (2016), besides better readability and reuse, there are several other technical reasons motivating such changes, e.g., performance, dependability, and security requirements.

Again, as one can observe in the figure, the tailored process (bottom) contains only its new flow elements (differences regarding the base process) and each of them has an associated tailoring operation (illustrated in the figure by a dotted arrow on the top right corner), which is actually a BPMN*t* extension element. Each tailoring operation has specific parameters (traceability links) that determine which element(s) from the base process its source element (new one) is related to. In order to avoid possible conflicts, operations involving insertions (*Serial Insert* for this case) are executed first and after are executed operations involving deletions (*Encapsulate* for this case). This criterion of operations application is determined through the parameter *applicationOrder = FirstAddition* of the *Extension* operation. Although we did not represent this operation in the following tailoring examples, it must always be the first operation applied when creating a new derived process with our approach, since it is responsible by establishing an extension/reuse relationship with a base process.

Therefore, the following tailoring operations are applied in the scenario of Figure 35 to obtain the final tailored process (Figure 36):

- **Serial Insert** of process fragment: Since the subprocess defined in the tailored model (on bottom part of Figure 35) has an associated tailoring operation whose binary parameter *additionIsFragment* was set to *true*, only the content of the subprocess (fragment) is inserted in the final tailored process (see Figure 36), directly into the main process workflow. Afterwards, sequence flows are reconfigured to connect to the process fragment inserted (operation post-condition).

- **Encapsulate** process fragment: The encapsulated fragment contains all the four original tasks of the base process, since the first one is pointed by the operation parameter *fragmentBegin* and the last one is pointed by the parameter *fragmentEnd.* These tasks are encapsulated into the subprocess *Capture and Analyze Requirements*, which defines the operation in the tailored process and, therefore, is its source element (see left bottom part of Figure 35). In Figure 36, the final content of this subprocess is shown in the detail. As one can observe, a start and end events were added in the beginning and end of the encapsulated process fragment (action performed by operation post-condition 1). Finally, the subprocess is placed in the same workflow position of the extracted fragment.

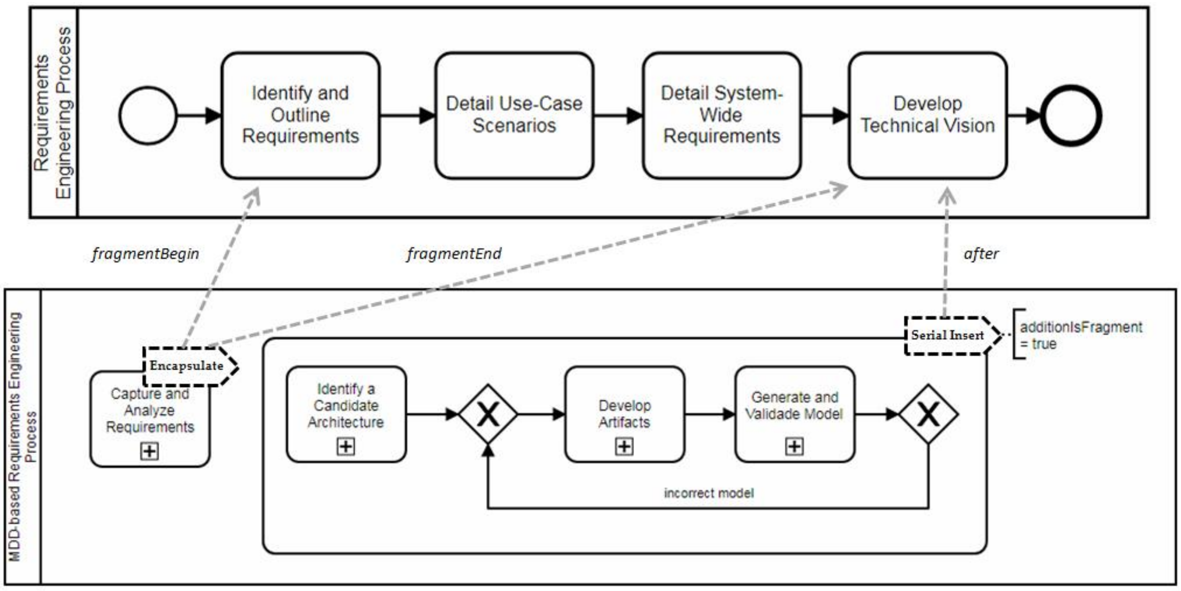

Requirements Engineering Process
Identify and Outline Requirements
Detail Use-Case Scenarios
Detail System-Wide Requirements
Develop Technical Vision
fragmentBegin
fragmentEnd
after
MDD-based Requirements Engineering Process
Encapsulate
Capture and Analyze Requirements
Serial Insert
additionIsFragment = true
Identify a Candidate Architecture
Develop Artifacts
Generate and Validade Model
incorrect model

Figure 35. Requirements Engineering base process (top) and schematic specification of tailoring (bottom)

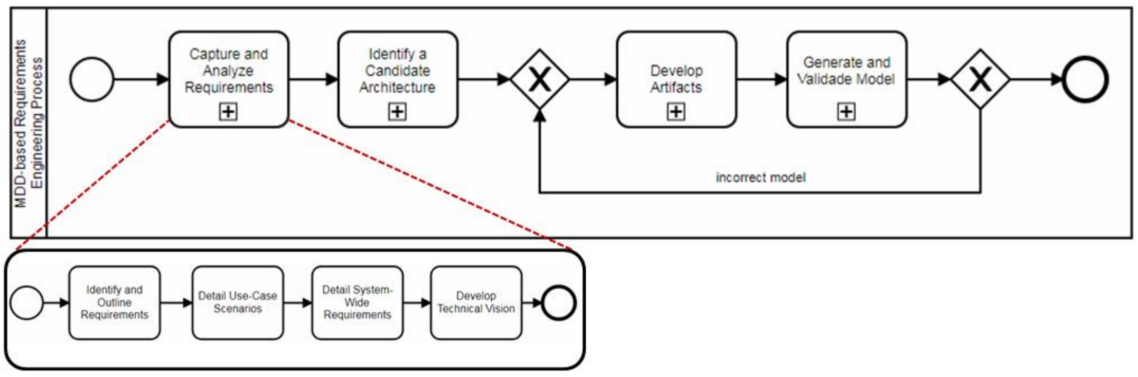

MDD-based Requirements Engineering Process
Capture and Analyze Requirements
Identify a Candidate Architecture
Develop Artifacts
Generate and Validade Model
incorrect model
Identify and Outline Requirements
Detail Use-Case Scenarios
Detail System-Wide Requirements
Develop Technical Vision

Figure 36. MDD-based Requirements Engineering tailored process after tailoring interpretation

6.4.2 Tailoring Scenarios from the BPM domain

6.4.2.1 Claim Handling Process

In this section we consider a scenario reported by KÜSTER *et al.* (2016), which, according to the authors, demonstrates the main changes performed when producing a technical-level process model from its business-level specification. In this context, most of performed changes do not essentially modify the process behavior, corresponding to the application of refinement operations. Figure 37 shows these changes in a claim handling process.

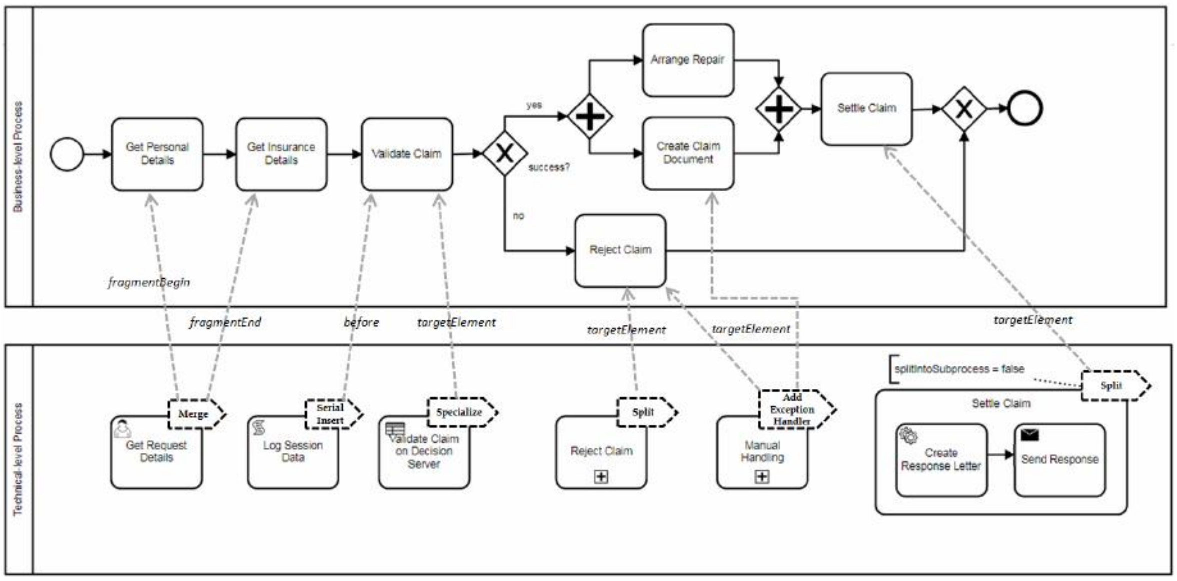


Figure 37. Business-level base process (top) and schematic specification of tailoring (bottom)

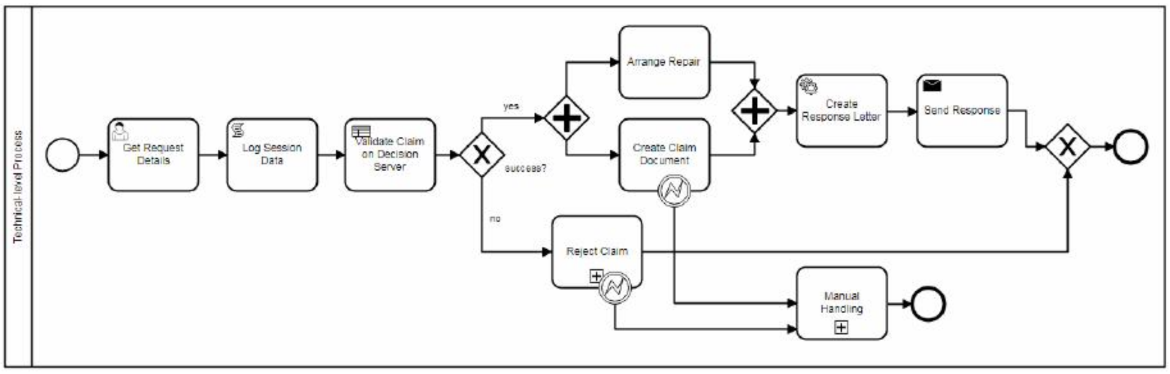


Figure 38. Technical-level tailored process after tailoring interpretation

This scenario applies operations *Merge*, *Serial Insert*, *Specialize*, *Split*, and *Add Exception Handler* on the (business-level) base process model, which is shown on top of Figure 37. Among these operations, only *Serial Insert* modifies in significant way the normal behavior of the process. The same figure also represents the tailoring specification for the (technical-level) tailored process of schematic way (on the bottom) in order to facilitate the understanding of performed adaptations. As one can observe in the figure, the tailored process (bottom) contains only its new flow elements (differences regarding the base process) and each of them have an associated tailoring operation. In general, tailoring operations are applied on the base process model following the order in which the process engineer specifies such operations in the tailored process model, i.e., from left to right in the figure. However, in order to avoid possible conflicts, all operations involving only insertions (e.g., *Serial Insert* and *Add Exception Handler*) are executed first and afterwards are executed operations involving deletions (e.g., *Merge, Specialize,* and *Split*).

Therefore, the following tailoring operations are applied in the scenario of Figure 37 to obtain the final tailored process (Figure 38):

- **Serial Insert** task *Log Session Data*: Inserts this task soon before the task named *Validate Claim* (identified by the operation parameter *before*); also reconfigures sequence flows for connecting the new task to the process workflow (operation post-condition 1). According to the authors of this model, task *Log Session Data* is a merely technical task, and for this reason it was not represented in the business-level base process.
- **Add Exception Handler** subprocess *Manual Handling*: Adds this subprocess as exception handler for tasks *Reject Claim* and *Create Claim Document* (identified by the operation parameter *targetElement*). In this case, the default value for the parameter *exceptionType* is taken on, i.e., *expectionType = Error*. This means the subprocess *Manual Handling* must deal with error exceptions that can occur while any of the two aforementioned tasks is executing. The occurrence of an error event in the context of these tasks diverts the normal flow of the process for an exception flow that takes to the added subprocess. According to the BPMN specification, error events must always interrupt the process' normal flow and this constraint is ensured by the operation pre-condition 1. Moreover, since our high-level operations aim preventing structural failures in the process, first post-condition 1 adds an event of Error type to the boundary of tasks *Reject Claim* and *Create Claim Document*. After, post-condition 2 adds a Sequence Flow outgoing from each added boundary event and incoming to the subprocess *Manual Handling* (see Figure 38). At last, post-condition 3 adds an end event after the exception handler, which is a good

practice recommended by the BPMN specification. In this way, the operation ensures the tailored process model remains correct after its application.

- **Merge** tasks *Get Personal Details* and *Get Insurance Details*: Combines these tasks from the base process (corresponding to the process fragment delimited by operation parameters *fragmentBegin* and *fragmentEnd*) into a single (human) task named *Get Request Details*, which defines the operation in the tailored process. In this context, such a practice means the separate steps of the human action are described elsewhere (e.g., a screenflow). At last, operation post-condition 1 reconnects the process flow by adjusting Sequence Flows to tie the new task in the same position of the grouped fragment. As the task *Log Session Data* was inserted earlier soon after the grouped fragment, it now succeeds the merged task.
- **Specialize** task *Validate Claim*: This plain task (identified by the operation parameter *targetElement*) is represented in the technical-level tailored process by a specialized BPMN task (of type BusinessRuleTask) that indicates it has some associated business rule. In process models of more technical nature, the use of specialized tasks, subprocesses, or events is a common practice when using BPMN. However, this tailoring operation involves several pre- and post-conditions because there are many constraints in the BPMN specification related to the specialization of each type of supported element (i.e., task, subprocess, or event). In order to support the correct specialization of basic process elements, we have specified the main BPMN well-formedness rules related to this operation (presented in Figure 30).
- **Split** task *Reject Claim*: Replaces this task from the base process by a subprocess that details its procedure in the tailored process. Besides the parameter *targetElement*, which identifies the task to be splitted, this operation has another parameter named *splitIntoSubprocess* that is binary (i.e., accepts values *true* or *false*). The default value of this parameter is true, which was taken on in this case. This means the task *Reject Claim* is detailed into a subprocess, which defines the operation. In this case, the subprocess has the same name of the task from the base process (i.e., *Reject Claim*). In this case, operation post-condition 2 reconfigures sequence flows to connect to the substitute subprocess whereas post-condition 3 links to this subprocess the error event (boundary event) attached to the original task, which was added earlier by the operation *Add Exception Handler*.

- **Split** task *Settle Claim*: Replaces this task from the base process by a process fragment (composed by the tasks *Create Response Letter* and *Send Response*) that details its procedure in the tailored process. Unlike the previous *Split* operation, in this one the parameter *splitIntoSubprocess* is set to *false*. This means the task *Settle Claim* is detailed by a process fragment (defined into the subprocess *Settle Claim* in the tailored process model) that must be part of the main process workflow directly (i.e., not enclosed in a subprocess' scope) (see Figure 38). In this case, only the content of the subprocess defining the operation is considered during the tailoring interpretation. Finally, operation post-condition 2 reconfigures sequence flows to connect the new process fragment into the main workflow, in the same position of the splitted task.

6.4.2.2 Loan Process

In this section we present a simple example in order to demonstrate the use of the operation *Event-Based Insert*, which we have created for meeting specific adaption needs of BPMN. To this end, we consider a loan process from DUMAS and PFAHL (2016), which has been modeled in BPMN by the authors. A part of this process consists of the sub-process "*Loan Offer*". In order to better expose the high-level operation, suppose that initially only the success case was modeled (see top part of Figure 39), in which a customer accepts the offered loan (identified by the process through the *Message* receiving event "*offer accepted*"). In such a case, the process will wait for this event to occur, which can never happen. Then, we want to adapt the process for identifying also a negative response from the customer (*Message* receiving event "*offer refused*") and send him/her a form to understand his reasons (new task "*Send form to understand refusal*"). This adaptation is reached through an operation *Event-Based Insert* defined by the new task of the tailored process (named "*Tailored Loan Offer*") as represented on the top of Figure 39. In other words, the new task "*Send form to understand refusal*" is the source element of the tailoring operation *Event-Based Insert* that adds this task as an event-based alternative to the process fragment between the start event and the end event in the tailored process (see bottom part of Figure 39). The type and name of the event that takes the new task to run are determined by the parameters *eventType* and *eventName* of the operation, respectively. Since all pre-conditions of this operation are met (specially pre-condition 2 that requires the first element of the target process fragment to be an event of type *Message*, *Signal*, *Timer*, or *Conditional*), the operation interpretation automatically adds to the final tailored process a BPMN Event-Based Gateway (event-based XOR-Split) before the alternative fragments starting by events and a BPMN Exclusive Gateway (XOR-Join) after these fragments (post-condition 2). The first gateway means the choice between alternative ways is determined by the first of the following events that occurs when the execution of the process arrives in this gateway. If the message "*offer accepted*" is received first, the execution flow proceeds to the task "*Record loan contract*". If the message "*offer refused"* is received first, then a form is sent to the customer. The second gateway only merges two exclusive branches into a single one. Moreover, post-condition 3 also adds to the final tailored process an intermediate event of specific type informed by the operation parameter *eventType* and name defined by the parameter *eventName*. This event will be responsible by the activation of the new task. Finally, post-condition 4 adds and adjusts sequence flows to connect the new elements to the tailored process workflow. The definition of this operation is in accordance with rules #21, #24, #25 and #26 from Appendix 1, which specify constraints of the BPMN standard related to the use of the Event-Based gateway.

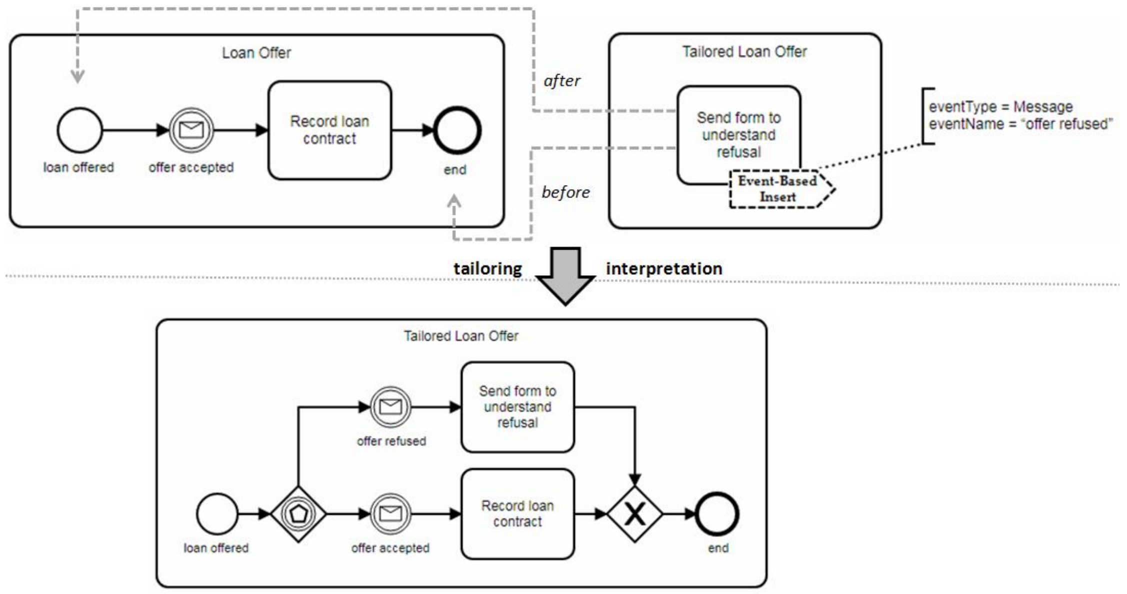


Figure 39. Tailoring the "Loan Offer" process – example 1

In a second example by using the same process (Figure 40), we now consider that the original process is equal to the resulting process of example 1. In this case, we want to add a new task that will cancel the loan offer when its expiry date is reached and the customer has yet not provided any response. This way, we specify the new task "*Cancel loan offer*" in the tailored process (at the top right of the figure) and add to it an operation *Event-Based Insert* defining how this task will be inserted in the tailored process workflow after tailoring interpretation. As parameters *after* and *before* were configured to link to both gateways of the base process, post-condition 1 is applied by connecting the new task to these gateways as a new event-based alternative path. Moreover, post-condition 3 adds immediately before the new task in the final tailored process a *time* intermediate event (according to configurations specified for operation parameters *eventType* and *eventName*) that will be responsible by its activation (see bottom part of Figure 40).

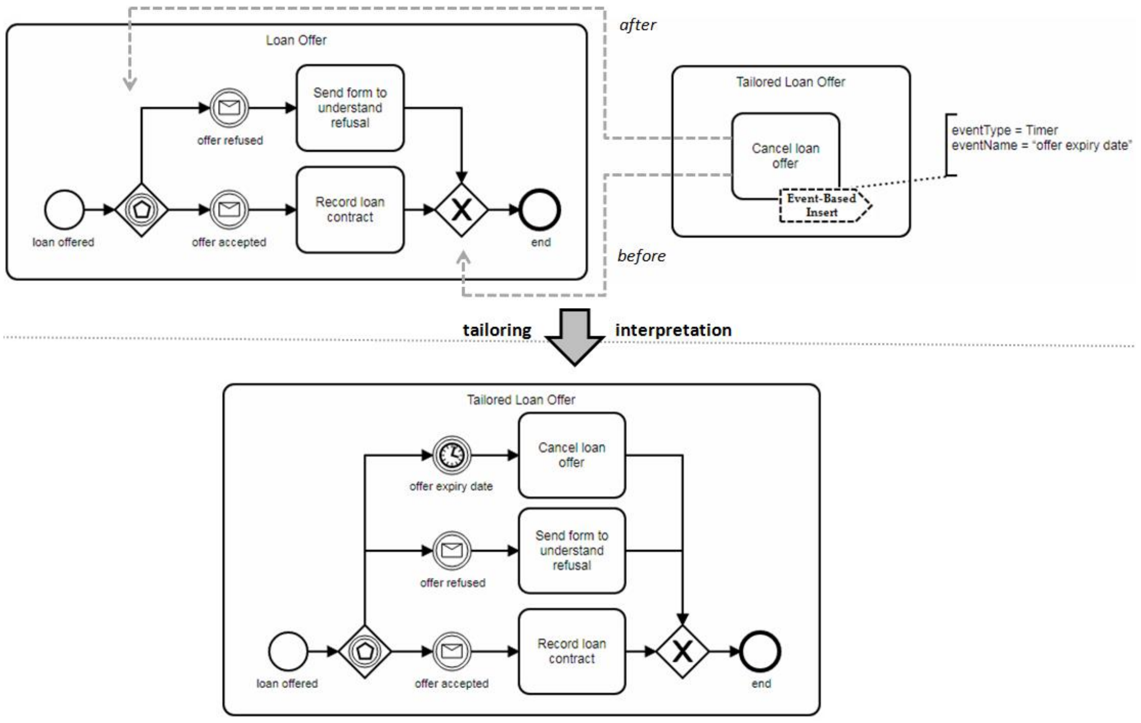


Figure 40. Tailoring the “Loan Offer” process – example 2

6.5 Conceptual Representation of the BPMNt Extension

Following our BPMN extension presented in the previous chapter, we used the BPMN’s built-in extension mechanism (OMG, 2011) to add support for high-level concepts. The standard extension mechanism consists of a set of extension meta-classes that allows attaching additional elements and attributes to BPMN elements.

The BPMN extension mechanism comprises four extension meta-classes. In summary, the *ExtensionDefinition* meta-class allows defining a new extension concept for BPMN elements whereas the *ExtensionAttributeDefinition* meta-class defines new extension attributes for an *ExtensionDefinition* element. The *Extension* meta-class binds/imports an extension definition and its attributes into a BPMN model and, finally, the *ExtensionAttributeValue* meta-class stores the values for additional attributes of a BPMN element instance defined in a model.

Our BPMN extension defines tailoring operations as language's extension concepts and their configuration parameters as extension attributes, generally bound to BPMN process elements. Relationship attributes of these operations are used by variant process models to configure tailoring operations and track adapted base process elements, since they maintain links between elements from the base and tailored process models. Such relationship attributes correspond to operation *parameters* in the presented catalog.

Figure 41 shows the structure of the new BPMN*t* extension into the *BPMNt Tailoring Extension* package and its relationships to standard BPMN elements, shown into the *BPMN* package. According to the aforementioned BPMN extension mechanism, tailoring concepts defined in the *BPMNt Tailoring Extension* package can be bound to any standard BPMN element as extension elements. However, we have limited their application to relevant process elements through additional rules.

All concepts defined by the BPMN*t* tailoring extension correspond to instances of the BPMN extension meta-class *ExtensionDefinition*. However, we have added the stereotype *<<ExtensionDefinition>>* only to the BPMN*t* extension super-class *TailoringOperation* to not hamper the understanding of the model (see Figure 41). Similarly, all extension attributes (including relationships) of BPMNt concepts correspond to instances of the BPMN extension meta-class *ExtensionAttributeDefinition*, identified in the figure by the stereotype of same name that has been added only to the aforementioned super-class.

We structured concepts representing tailoring operations hierarchically according to two classifications, high-level operations and basic operations (see Figure 41), in which the first operation group reuses a set of basic operations to build more complex adaptation patterns that better represent user's intentions.

We refactored the BPMN*t* extension model presented in the previous chapter, which applied the same meta-class structure used by SPEM to represent tailoring concepts. We now represent each basic tailoring operation as a separate class that derives from the common super-class *BasicOperation* such as shown in Figure 41. The set of basic operations comprises the tailoring operations *contribute* and *suppress* presented in the previous chapter, but with names slightly modified to correspond to verbs instead of nouns and without rules associated (pre- and post-conditions). Such rules were related to corresponding high-level operations, since basic operations do not represent a whole and independent adaptation. The *replace* operation, also presented in the previous chapter, was classified as high-level operation because it is essentially a composition of operations *suppress* and *contribute*.

Extend (which corresponds to the *extension* operation presented in the previous chapter) is a special type of operation, since it determines a relationship of extension/reuse in which a variant (tailored) process extends/reuses a given base process (identified by the operation parameter *extendedProcess*), enabling the use of tailoring operations that modify the base process' structure. Thus, we represent the concept *Extend* in the same hierarchical level of *BasicOperation* (see Figure 41). The enumeration *OrderType* defines available options for one to configure the application order of tailoring operations through the parameter *applicationOrder* of *Extend*. The value *Free* indicates the operation application order is defined by the specification order of the operations in the tailored process model. On the other hand, the value *FirstAddition* determines that operations adding process elements must be executed before the others.

Therefore, the proposed set of basic (primitive) operations comprises the following tailoring operations:

1. *Modify: Changes property values of a reused process element from the base process*. Description: An element from the base process (identified by the operation parameter *modifiedElement*) has a given property modified (identified by the operation parameter *property*). The new property value is provided by the operation parameter *value*, for simple properties, or by the parameter *valueRef*, for association properties (see definition of *Modify* concept in Figure 41). This operation has not been previously considered by the BPMN*t* approach, but it is necessary to support some high-level operations.
2. *Suppress: Removes an element from the structure of the reused base process*. Description: An element of the base process (identified by the operation parameter *suppressedElement*) is removed from its reused process structure (see definition of *Suppress* concept in Figure 41).
3. *Contribute: Adds a new element to the reused base process*. Description: A variant process defines a new process element that should be added to the reused structure of the base process itself or to some of its sub-processes

(identified by the operation parameter *targetProcess*). However, this operation does not support the direct specification of a workflow position to addition of the new node element; such position must be determined by *Sequence Flow* elements connecting the new element to other ones. The operation parameters *newElement* and *newElementRef* reference the added element. The former is used when the new element has been specified by the process engineer and the second parameter is used when the new element is automatically created by the *interpreter* component of the BPMN*t* approach, during the execution of a high-level operation (see definition of *Contribute* concept in Figure 41). Elements generated by the approach are typically sequence flows or gateways.

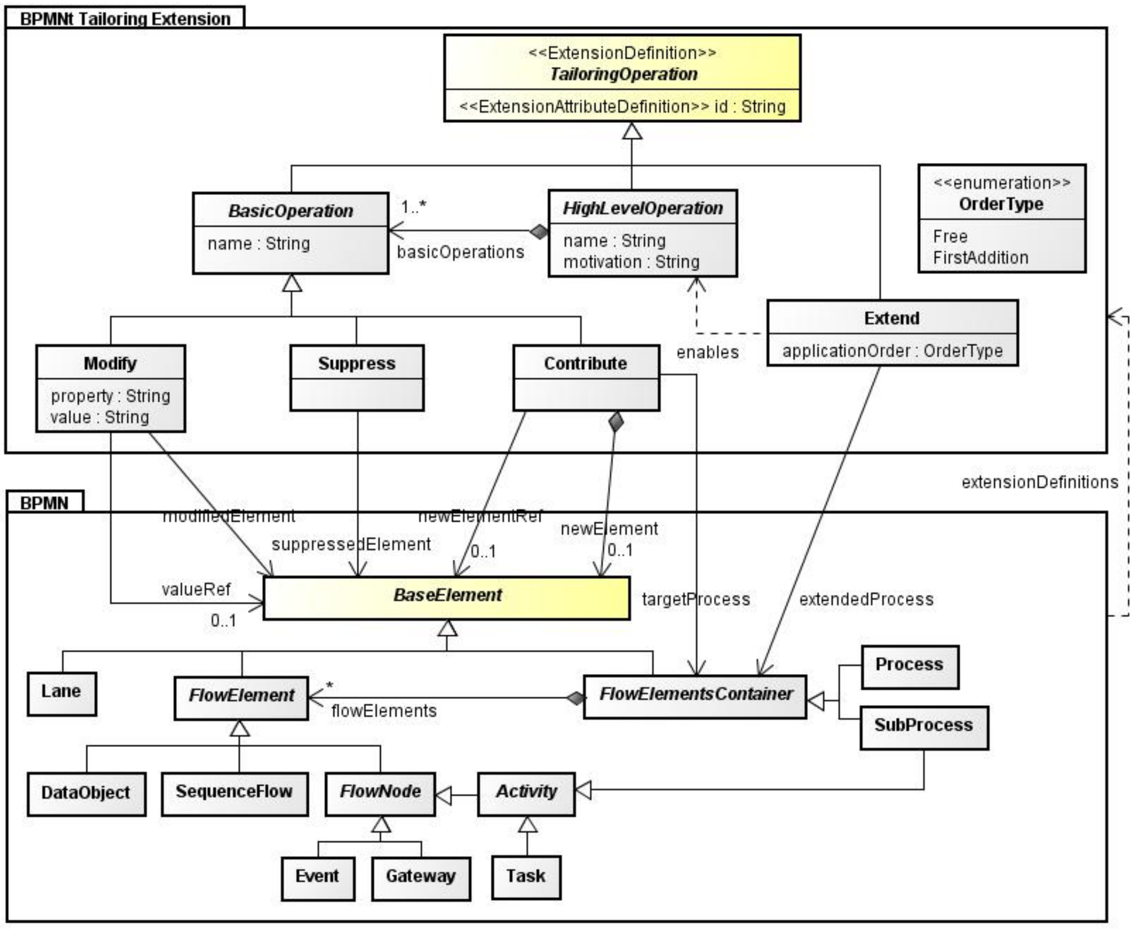


Figure 41. Main meta-classes of the BPMN*t* extension (top) and their relation to BPMN meta-classes (bottom)

High-level operations represent abstractions of a set of basic operations and must be defined by subclasses of *HighLevelOperation*. The operations of our catalog are presented in the class diagram of Figure 43. Each subclass needs to define the parameters of the high-level operation whereas its execution semantics must be defined in the *interpreter* component that integrates the proposed solution structure. The execution semantics of high-level operations shoud be defined by composing basic operations.

In order to allow the comprehension of the rationale of each adaptation applied to a variant model, the super-class *HighLevelOperation* also provides the attribute *motivation*. Providing information to such attribute along with a tailoring operation allows one to understand more easily *why* an adaptation has been performed.

The relationship attribute *basicOperations* of the class *HighLevelOperation* shown in Figure 41 is used by variant process models to record the sequence of basic operations applied by an instance of high-level operation. Such a sequence of operations can be determined only during the execution of tailoring operations by the *interpreter* component, because it depends of the state of the model at the moment in which the operation is executed. For example, when an operation *delete* is applied to remove a task from a process workflow, it can also be necessary to remove gateways (i.e., splits or joins) related to the removed task that have become inconsistent after the operation.

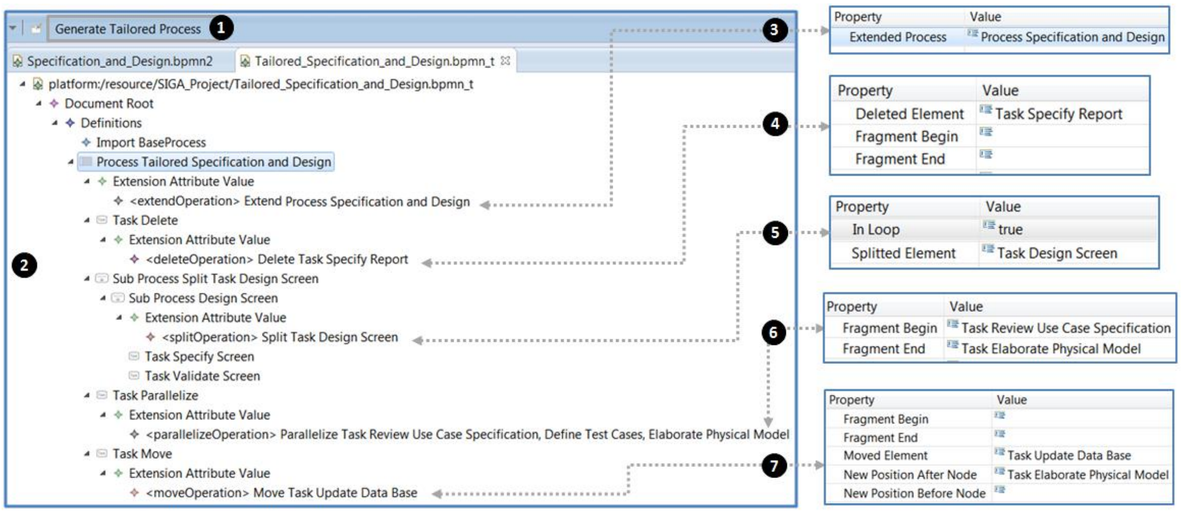
It is important to highlight the BPMN*t* meta-class structure for defining tailoring operations can be extended according to needs of a given user, environment or context. To support a new high-level operation, one should: (1) provide a new subclass of *HighLevelOperation* with specific operation parameters; (2) define the execution semantics of operation into the *interpreter* component; and (3) define well-formedness rules (pre- and post-conditions) related to the new operation.

High-level operations can be defined with the single purpose of making an existing operation more intuitive for the user. For example, the operation *rename* presented in our catalog of high-level operations, which is often used in practice to customize process elements, consists of a single basic operation, of type *modify*, in which the element's property to be modified is previously defined by the operation *rename*, i.e., the name of the element. Moreover, the name of the operation *rename* makes evident its purpose.

Finally, in order to facilitate the reuse and application of the BPMN*t* extension for adapting BPMN process models, we have designed it to be compliant with the BPMN standard extension mechanism (discussed in Section 2.3.2), which allows assigning new extension elements for BPMN elements. The complete specification in XML Schema of the BPMN*t* extension supporting high-level operation concepts is presented in Appendix 3. From this specification, process engineers can import and use our extension into BPMN standard-compliant tools. In the next section, we briefly present our support prototype, which consists of an EMF-based domain specific language derived from the XML Schema specification presented in Appendix 3.

6.6 Support Prototype

To support high-level operations, we have extended our prototype presented in the previous chapter, which is based on the MDT/BPMN2 Project (MDT, 2012). We used a simple model editor from this project for manipulating BPMN process models, which are represented as hierarchical structures (trees). Figure 42 shows the main graphic interface of the prototype representing one of the tailoring scenarios of the SIGA Project (scenario 1, represented graphically in Figure 31).


Generate Tailored Process
1
2
3
4
5
6
7
Specification_and_Design.bpmn2
Tailored_Specification_and_Design.bpmn_t

```
platform:/resource/SIGA_Project/Tailored_Specification_and_Design.bpmn_t
  Document Root
    Definitions
      Import BaseProcess
      Process Tailored Specification and Design
        Extension Attribute Value
          <extendOperation> Extend Process Specification and Design
        Task Delete
          Extension Attribute Value
            <deleteOperation> Delete Task Specify Report
        Sub Process Split Task Design Screen
          Sub Process Design Screen
            Extension Attribute Value
              <splitOperation> Split Task Design Screen
            Task Specify Screen
            Task Validate Screen
        Task Parallelize
          Extension Attribute Value
            <parallelizeOperation> Parallelize Task Review Use Case Specification, Define Test Cases, Elaborate Physical Model
        Task Move
          Extension Attribute Value
            <moveOperation> Move Task Update Data Base
```

Property	Value
Extended Process	Process Specification and Design

Property	Value
Deleted Element	Task Specify Report
Fragment Begin	
Fragment End	

Property	Value
In Loop	true
Splitted Element	Task Design Screen

Property	Value
Fragment Begin	Task Review Use Case Specification
Fragment End	Task Elaborate Physical Model

Property	Value
Fragment Begin	
Fragment End	
Moved Element	Task Update Data Base
New Position After Node	Task Elaborate Physical Model
New Position Before Node	

Figure 42. Tailoring specification based on high-level operations using our support prototype

To represent process adaptations and to obtain the final tailored process in Figure 32, we have created a variant process model named *Tailored Specification and Design* (presented in Figure 42 using our prototype) that contains new process elements (e.g., the subprocess Design Screen) and definitions of tailoring operations. Operation definitions are always contained by elements *Extension Attribute Value* associated to a standard BPMN element. The BPMN element containing the tailoring operation definition represents a new element of the tailored process (contribution) for the case of operations *Replace, Split, Serial Insert, Conditional Insert* and *Parallel Insert*. For other operations, such an element will not be included in the final tailored process model.

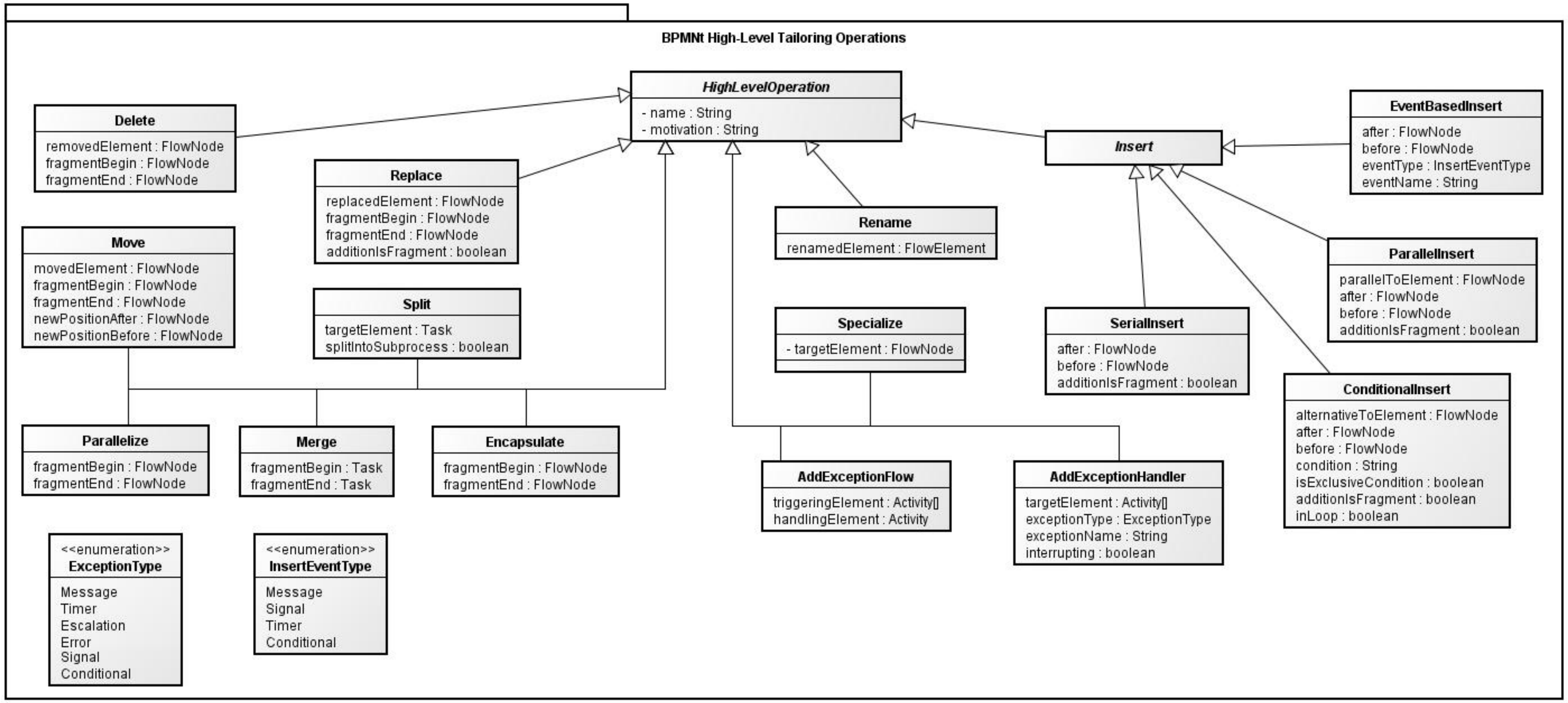


Figure 43. BPMN*t* meta-classes representing high-level tailoring operations

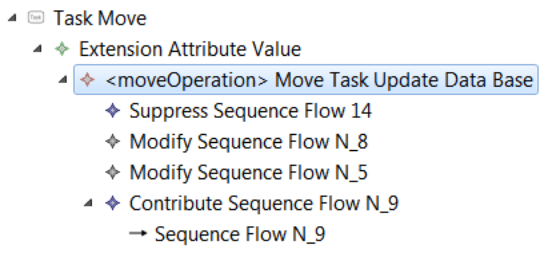
The figure also shows the main parameters (on the right part) of high-level tailoring operations applied by this running example and their configuration values. The tailored process model *Tailored Specification and Design* represented in the figure has the following specification of tailoring operations:

- *Delete task Specify Report* (see specification of number 4 in Figure 42): The execution of the operation removes this task from the final tailored process as well as the two exclusive gateways (XOR_DIV and XOR_CONV) that after the task removal have only one input flow and one output flow (incorrect model state); also connects the task Review Use Case Description to Design Screen and Define Test Cases to Elaborate Physical Model.
- *Split task Design Screen* (see specification 5 in Figure 42): The execution of the operation transforms this general task of the base process in a subprocess that details its procedure in the tailored process.
- *Parallelize tasks Review Use Case Specification, Define Test Cases, and Elaborate Physical Model* (see specification 6 in Figure 42): The execution of the operation includes a parallel gateway (AND_DIV) before these tasks to diverge the process flow and another parallel gateway (AND_CONV) after the tasks to converge (synchronize) again the process flow.
- *Move task Update Database* (see specification 7 in Figure 42): After the execution of the operation parallelize (described above), the task Update Database is performed only after the synchronization of parallelized tasks. However, it should be performed soon after the task Elaborate Physical Model. Thus, the operation move is applied to shift the task Update Database to the new position (indicated by the parameter newPositionAfterNode).

Before executing any tailoring operation from a specified scenario, the prototype checks all pre-conditions of these operations. The transformation of the base process model to the tailored process model only starts when no operation's pre-condition is violated by the current specification.

The execution order of tailoring operations generally corresponds for the same order in which they were specified in the process structure of the BPMNt file, represented by the model *Tailored Specification and Design* in Figure 42, but prioritizing insertion operations in order to avoid operation conflicts. This is the default behavior when interpreting a BPMN*t* tailoring specification. However, this behavior can be altered by the user by configuring the parameter *applicationOrder* of the *Extend* operation, which is always the first operation specified in a BPMN*t* tailoring scenario.

During the execution (interpretation) of high-level operations specified by the user in the tailored process model, the interpreter component of the BPMNt approach records in this model the basic tailoring operations that have been executed to achieve the expect result of each high-level operation. Basic operations that assembly a given high-level operation are added in the tailored process model (BPMNt file) as sub-elements of the one that represents the high-level operation, as such shown in Figure 44 for the high-level operation *Move*. Such an operation is achieved by executing the following set of basic operations: one *suppress* operation removing a sequence flow; two *modify* operations changing connection properties of sequence flows; and one *contribute* operation adding a new sequence flow to the process.



```
Task Move
  Extension Attribute Value
    <moveOperation> Move Task Update Data Base
      Suppress Sequence Flow 14
      Modify Sequence Flow N_8
      Modify Sequence Flow N_5
      Contribute Sequence Flow N_9
        Sequence Flow N_9
```

Figure 44: Set of basic operations automatically generated

6.7 Concluding Remarks

Well-defined language constructs specifying tailoring possibilities, i.e., adaptation operations, and their constraints are important in order to support process engineers in specifying process adaptation scenarios. However, such constructs should also be semantically meaningful for humans in order to better represent user’s adaptation intentions, facilitating the posterior understanding of performed changes.

In order to tackle these challenges, in this chapter we proposed a BPMN meta-model extension and a catalog defining a set of tailoring operations (language concepts) based on BPM adaptation and refinement patterns that intend to improve our previous proposal (presented in Chapter 5) regarding the following issues: (1) understandability of the performed changes; (2) abstraction of model transformation details; (3) reduction of the number of necessary operations for specifying a tailoring scenario; and (4) conservation of the structural correctness and well-formedness of tailored models regarding semantic rules of the BPMN standard.

High-level operations proposed in this chapter realize a user's adaptation intent. This means that all necessary elements and connections for producing such an adaptation associated with the operation must be correctly added to the process and configured, resulting in a valid BPMN process model. To achieve this objective, our operations impose constraints on the way as a base process can be adapted (through operation pre-conditions) and provide "action rules" (operation post-conditions) that modify the resulting process model, whenever necessary, to avoid disconnected elements, remove trivial gateways and sequence flows or add gateways and events related to the correct configuration of the operation semantics.

CHAPTER VII

7. Evaluation

7.1. Introduction

In this thesis, we presented a BPMN extension and support mechanisms for dealing with adaptations in process models represented in this language. Our research involved two target domains, Software Process Engineering (SPE) and Business Process Management (BPM).

In Chapter 5, we proposed an extension of the BPMN meta-model for specifying process tailoring inspired on widespread approaches in the SPE domain. This extension had software processes as its target domain. In the same chapter, we also showed the feasibility of the proposal by using real-world tailoring data from this domain (Section 5.6). This evaluation study demonstrated the effectiveness of the proposed tailoring concepts (operations) in representing tailoring scenarios from the software development project SIGA, since all extension concepts could be applied.

Next, in Chapter 6 we presented an extension of our previous proposal aiming to solve limitations identified from the conducted evaluation study (in Section 5.6) as well as expanding our solution scope for covering also business processes. Therefore, the BPMN*t* solution supports flexible process tailoring in both domains as well as in different application scenarios in which BPMN-based process adaptation can be useful. For example, covering not only the derivation of business-level processes, but also the refinement of these processes to technical-level models.

Thus, our concern here is on assessing the feasibility of the BPMN*t* solution in different situations from SPE and BPM domains that lead to the production of new BPMN models by adapting existing ones. More specifically, our research problem concerns the completeness and correctness of BPMN process models generated by our proposal.

In this sense, we judged to be more interesting to evaluate our research through different studies based on process models and tailoring scenarios from the real-world. In this way, we can use varied data sources and process models that preserve their original characteristics.

In Section 7.2 we present our evaluation plan. Section 7.3 presents the first study, the second one is presented in Section 7.4 and, at last, Section 7.5 depicts our third evaluation study. Next, Section 7.6 presents threats to validity of the studies whereas in Section 7.7 some conclusions are drawn.

7.2. Evaluation Plan

Next, we present the evaluation plan that guided our studies. The general objective is described in Section 7.2.1 and our research questions are presented in Section 7.2.2. An overview of evaluated contexts is given in Section 7.2.3 whereas the procedure for selection of such contexts and data collection is presented in Section 7.2.4.

7.2.1 General Objective

We intend to conduct evaluation studies of *descriptive purpose* (ROBSON, 2011). Our aim is to evaluate the feasibility of the BPMN*t* solution based on high-level operations to represent real-world adaptation needs in different application scenarios from SPE and BPM domains. Moreover, we want to evaluate the capacity of BPMN*t* to produce correct tailored process models, regarding a structural perspective (i.e., with absence of flow breaks) as well as regarding BPMN specification's constraints.

This general objective is detailed bellow according to the structure proposed by BASILI *et al.* (1994):

Analyze	the specification of tailoring scenarios and BPMN process models produced by applying BPMN*t* high-level operations implemented in our support prototype.
For the purpose of	Characterizing
With respect to	Feasibility of the proposed BPMN*t* solution and effectiveness of performed adaptations
From the viewpoint of	Researcher (and domain expert in one of studies)
In the context of	Three real processes from different domains and application contexts

7.2.2 Research Questions

Bellow, we present the research questions (RQ) that this evaluation aims to answer.

- **RQ1:** *Is the BPMNt extension based on high-level tailoring operations capable of specifying the adaptation needs from the evaluated contexts?*

  With this question we intent to assess the feasibility of the set of BPMN*t* high-level tailoring operations. In other words, we wanted to verify the capacity of BPMN*t* to express process variations required in the evaluated contexts. This question will be evaluated by comparing the number of adaptations required from each tailoring scenario with the number of adaptations supported by BPMN*t*.

- **RQ2:** *Are the BPMN process models generated by the BPMNt solution correct in the evaluated contexts?*

  This question aims to assess if BPMN process models produced by our solution are correct considering two different analysis perspectives: (1) structural correctness (i.e., without flow fails) and (2) BPMN specification-based correctness (i.e., if semantic rules of the language are met). The structural correctness will be evaluated considering the number of flow breaks (disconnected nodes) found in the produced models. The BPMN specification-based correctness, i.e. model well-formedness, will be evaluated by observing the number of BPMN's rule violations in the models produced by BPMN*t*. Since we have not found an available implementation for the research conducted by CORREIA (2014), we have used the BPMN model checker provided by the system *Signavio*[11], which had the best score in identifying violations of rules prescribed by the BPMN specification in the study conducted by CORREIA (2014).

[11] https://academic.signavio.com/

7.2.3 Description of Contexts

We will present three evaluation studies based on real process models from the following contexts:

1) **Software Development Process (SIGA Project):** The SIGA project involves Brazilian federal institutions aiming at developing an academic management system. Each identified variation in this context can be seen as a possible improvement for the base process model of the project (SANTOS, OLIVEIRA and ABREU, 2015).
2) **ATM Process in Banking:** The ATM process was modeled within a Brazilian Banking (BRANCO et al., 2014). In this context, variations represent changes from a business-level BPMN process to a technical-level BPMN process.
3) **Picture Postproduction Process in Film Industry**: This context represents a typical case of organizational process variability. Process variants from the Australian Film, Television and Radio School (AFTRS) were modeled and validated by ROSA *et al.* (2017). Such variants share commonalities while also showing differences. In this case, a base process has not been pre-defined. Therefore, we needed to define it before applying our tailoring approach.

7.2.4 Contexts Selection and Data Collection

Since our proposal requires process variants derived from a base process model, we have found difficulty to obtain models for our evaluation. Software or business organizations usually do not share their processes, since they represent a business differential. For this reason, we have based our evaluation on real use cases obtained by convenience (i.e., easier availability). Nevertheless, we could obtain case data from different domains (in SPE and BPM) and applying process tailoring to varied purposes. In the evaluation study based on the SIGA project (SPE domain), adaptations represent possible improvements for the base process model (SANTOS, OLIVEIRA and ABREU, 2015). In the study from banking, adaptations are applied in order to refine a business-level process model to a technical-level model. Finally, in the study from the film industry, tailoring can be applied to derive new variants of a picture postproduction process from a base model composed by activities commonly performed in this process. Despite the little number of case studies and evaluated models, we believe evaluations based on these contexts provide initial evidence of the applicability and effectiveness of our solution for adapting BPMN process models in different application contexts and from both SPE and BPM domains.

Process models from the SIGA project were obtained directly with a member of the project’s development time that collaborated with this research. On the other hand, models used for the other evaluation studies reported in this chapter were collected from the literature (BRANCO, 2014) (ROSA et al., 2017). In all the cases, we have based our evaluation on already available process data, i.e., *archival data* (RUNESON and HÖST, 2009).

7.3. Evaluation Study 1: Software Development Process SIGA

In this study, we have adopted software process models and tailoring scenarios from the SIGA-EPCT Project (SIGA, for short). SIGA-EPCT stands for *Academic Management Integrated System for Technological, Scientific and Professional Education* and is intended to develop an academic management system to be used by public universities in Brazil.

The project began in 2008 and is developed collaboratively by researchers and developers from Federal Institutes of Education, Science and Technology through research centers geographically distributed throughout Brazil. The project has a defined software development process, which comprises the following phases: Planning, Requirements, Specification and Design, Implementation, Test, and Deployment. However, driven by the aim to evaluate our work, we have focused on the Specification and Design phase, since this is core to the project. This phase is composed by the following activities: *Describe Use Case, Review Use Case Description, Design Screen, Elaborate Class Diagram, Update General Class Diagram, Review Use Case Specification, Define Test Cases, Elaborate Physical Model,* and *Update Database*.

The representation in BPMN notation of the process associated with the Specification and Design phase has already been shown in Figure 19 (Chapter 5). The SIGA project is centered on use cases and as a result a new process is instantiated for each use case of the project. The project records all process instances in a task management system for each use case (i.e., tasks, roles and sequence flows) in order to better allow orchestration of the people involved. Using the records from the task management system, another research project (SANTOS, OLIVEIRA and ABREU,

2015) has applied data mining and compliance analysis techniques to identify variances between the defined project process and the process executed by the project team in some core use cases. Such variances would be useful to the process engineer to evaluate the possibility of adaptations on the project base process in order to better reflect the actual execution behavior since, according to FERRATT and MAI (2010), tailoring can also be defined in terms of modifications that emerge from the results of monitoring a project and providing feedback on the development process.

7.3.1 Execution Procedure

A specialist from the SIGA Project team has modeled in BPMN the process of the Specification and Design project phase, used as base process in this evaluation study, and we have specified variances in its execution from six real use cases of the project, whose BPMN models were produced by the specialist by mining the project's task repository. Thus, we have created six BPMN*t* files in our prototype, each one containing the differences of a specific process execution case and tailoring operations linking to base process elements. Figure 42 (presented in Section 6.6), for example, shows the BPMN*t* tailoring specification for one of the SIGA project's use cases (scenario 3 in Table 13). Finally, to evaluate the completeness and correctness of the tailored process models generated by our prototype, they have been checked by the domain specialist from the SIGA project. The correctness regarding rules from the BPMN specification has been checked through the automatic support provided by the *Signavio* system.

7.3.2 Results

All six variant process cases evaluated in this study from the SIGA project are presented in Appendix 4 (Figure 64 to Figure 69).

Table 13 presents all high-level tailoring operations applied to each variant case (following the application order top-down) when using the BPMNt support prototype. In this table, each evaluated tailoring scenario (execution case) is identified by a number in the column ID. The table also details high-level tailoring operations in terms of the number of basic adaptations (additions, removals, and modifications) performed on specific process elements. The most of these basic adaptations are automatically executed by rules of the operation in an effort to ensure the correctness of BPMN models after tailoring. The last column of the table, named *Result*, uses the symbol '+' for indicating that a BPMN*t* adaptation operation was totally effective at executing an adaptation pattern. That is, modifications produced by the operation resulted in a correct BPMN model (without structural fails and concerning the BPMN's specification). As one can observe from the table, all adaptations required from the evaluated scenarios of the SIGA project were successfully supported and executed by BPMN*t*, resulting in complete and correct tailored models.

7.4. Evaluation Study 2: ATM Process in Banking

In this section, we have adopted business process models and tailoring scenario for evaluating our proposal from the Bank of Northeast of Brazil (BNB), which have been collected and published in a case study reported by BRANCO *et al.* (2014).

BNB is controlled by the federal government and oriented towards regional development. The Information Technology (TI) area of the bank contains over 300 professionals, responsible for maintaining more than 200 information systems in operation. Since 2007, BNB has used Business Process Management (BPM) based on the WebSphere family of products from IBM, including Business Modeler, Integration Developer, Business Monitor, and Process Server. The development process is based on the Rational Unified Process (RUP), extended to include business process modeling, and entails iterative and multi-staged model refinement, resulting in three types of models: business specifications, technical specifications, and executable specifications (BRANCO *et al.*, 2014).

Table 13. Results of BPMN*t* process tailoring involving the SIGA Project

ID	*BPMNt High-level Operation*	*Added (Contribute)*					*Removed(Suppress)*					*Modified (Modify)*					*Result*
		Task	*Subproc.*	*Gateway*	*Event*	*Flow*	*Task*	*Subproc.*	*Gateway*	*Event*	*Flow*	*Task*	*Subproc.*	*Gateway*	*Event*	*Flow*	
1	Extend Process *Specification and Design*	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	+
	Delete *Specify Report*	-	-	-	-	-	1	-	2	-	4	-	-	-	-	2	+
	Split *Design Screen*	2	1	-	2	3	1	-	-	-		-	-	-	-	2	+
	Serial Insert *Elaborate Mapping of Use Case Links*	1	-	-	-	2		-	-	-	1	-	-	-	-	-	+
2	Extend Process *Specification and Design*	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	+
	Delete *Specify Report*	-	-	-	-	-	1	-	2	-	4	-	-	-	-	2	+
	Split *Design Screen*	2	1	-	2	3	1	-	-	-	-	-	-	-	-	2	+
	Parallel Insert *Elaborate Mapping of Use Case Links*	1	-	2	-	4	-	-	-	-	-	-	-	-	-	2	+
3	Extend Process *Specification and Design*	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	+
	Delete *Specify Report*	-	-	-	-	-	1	-	2	-	4	-	-	-	-	2	+
	Split *Design Screen*	2	1	-	2	3	1	-	-	-	-	-	-	-	-	2	+
	Parallelize *Elaborate Physical Model, Define Test Cases* and *Review UC Specif.*	-	-	2	-	6	-	-	-	-	-	-	-	-	-	2	+
	Move *Update Database*	-	-	-	-	1	-	-	-	-	1	-	-	-	-	2	+
4	Extend Process *Specification and Design*	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	+
	Delete *Specify Report*	-	-	-	-	-	1	-	2	-	4	-	-	-	-	2	+
	Conditional Insert *Review Screen Desing*	1	-	2	-	5	-	-	-	-	1	-	-	-	-	-	+
	Serial Insert *Elaborate Mapping of Use Case Links*	1	-	-	-	2	-	-	-	-	1	-	-	-	-	-	+
	Conditional Insert *Correct Inconsistencies in Use Case Spefic.*	1	-	2	-	4	-	-	-	-	-	-	-	-	-	2	+
5	Extend Process *Specification and Design*	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	+
	Delete *Specify Report*	-	-	-	-	-	1	-	2	-	4	-	-	-	-	2	+
	Delete *Review Use Case Description*	-	-	-	-	-	1	-	-	-	1	-	-	-	-	1	+
	Delete *Define Test Cases*	-	-	-	-	-	1	-	-	-	1	-	-	-	-	1	+
	Parallelize *Design Screen, Elaborate Class Diagram* and *Update General Class Diagram*	-	-	2	-	6	-	-	-	-	-	-	-	-	-	2	+
	Parallelize *Elaborate Physical Model* and *Review Use Case Specification*	-	-	2	-	4	-	-	-	-	-	-	-	-	-	2	+
	Move *Update Database*	-	-	-	-	1	-	-	-	-	1	-	-	-	-	2	+
6	Extend Process *Specification and Design*	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	+
	Delete fragment from *Design Screen* to *Define Test Cases*	-	-	-	-	-	5	-	2	-	8	-	-	-	-	2	+
	Serial Insert *Validate Report*	1	-	-	-	2	-	-	-	-	1	-	-	-	-	-	+

In our evaluation study, we focus on business and technical models that are completely presented in the original research. We do not consider here executable models, because dealing with models at this level of abstraction is out of the scope of our research. Probably due to issues related to organizational information secrecy, only a business process of BNB has been completely presented, i.e., the *Automated Teller Machine* (ATM) process. Moreover, we have not found other researches reporting case studies in this context of model refinement across levels of abstraction. Therefore, the BNB's ATM process is the single real process case that we have obtained focusing on model refinement rather than purely adaptation. Nevertheless, this model still allowed us to apply and evaluate many of our high-level operations aiming at refinement.

Figure 45 and Figure 46 show two models representing the process of using an *Automated Teller Machine* (ATM) system at different levels of abstraction. We will use these models, which are versions of real process models from the company, as base (Figure 45) and variant (Figure 46) processes in this tailoring evaluation study.

The two models are the same as the original models, except that their labels were translated from Portuguese to English by the enterprise's member that collected and published such models in the original case study (BRANCO *et al.*, 2014). The first model (Figure 45) represents a business-level process specification, which is created by Business Analysts. The second model (Figure 46) is a refinement of the first one, created by IT Systems Analysts. These stakeholders use such models to align the modeled process with the existing service infrastructure.

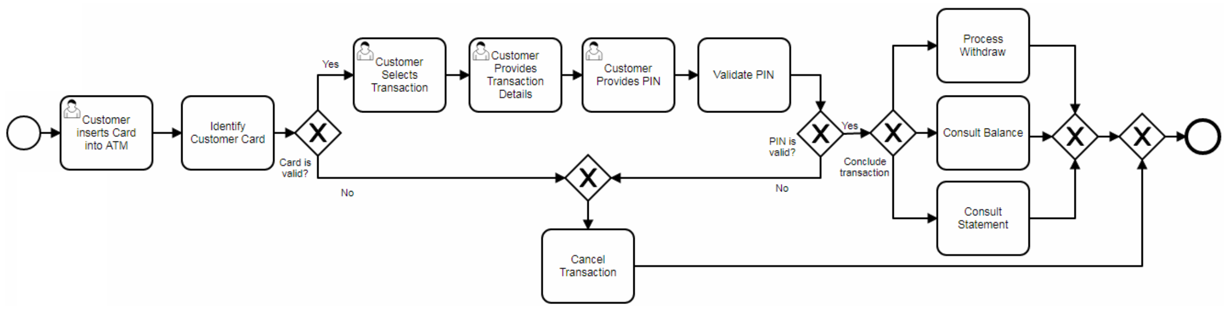


Figure 45: Base process model (Business-level ATM Process)

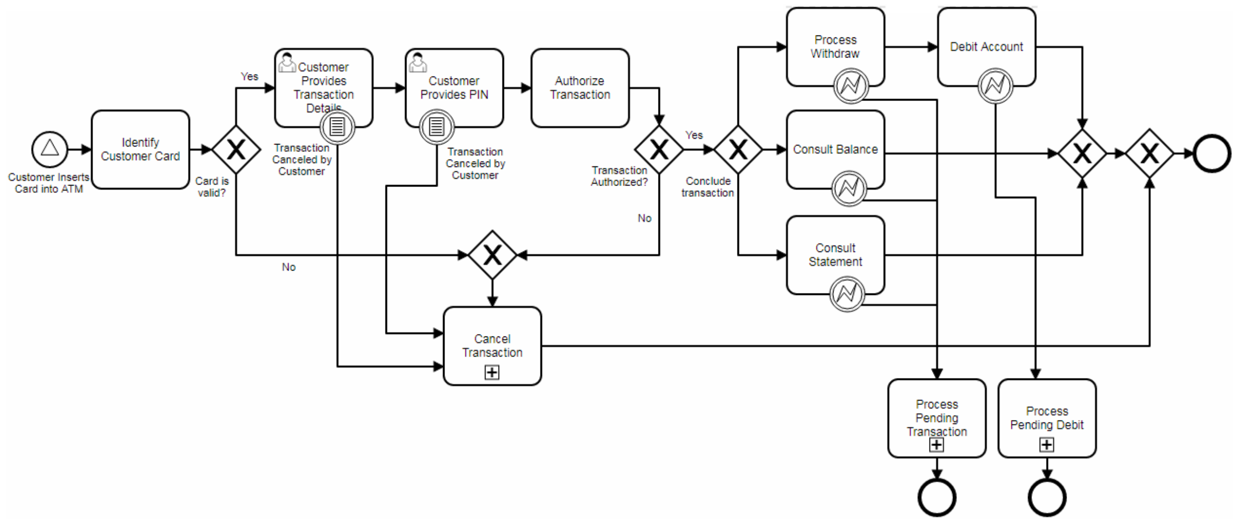


Figure 46: Variant process model (Technical-level ATM Process)

7.4.1 Execution Procedure

A specialist's team from the BNB organization has modeled in BPMN the ATM process of business- and technical-levels, used as base and variant process models, respectively, in our evaluation study. However, we have remodeled such models into the BPMN modeling tool *bpmn.io*[12], since it allows saving them in the BPMN file format, which is the standard format for exchange of BPMN process definitions. Having the base process model specified in this format, we could directly import it into our prototype for specifying and executing necessary adaptations for producing the variant process model.

Next, we have specified variances of the technical-level process (Figure 46) regarding the business-level process (Figure 45) by using the BPMN*t* support prototype. In other words, we have created a BPMN*t* file in our prototype containing the new process elements of the technical model and tailoring operations linking them to base process elements. Finally, in order to evaluate the completeness and correctness of the tailored process model generated by our prototype, we have compared it to the original technical-level process shown in Figure 46. Our model is correct if it has the same functionality and behavior than the original technical process (that does not contain structural fails). Otherwise, there are problems that need to be reported in the presentation of results. The correctness concerning rules from the BPMN standard has been checked through the automatic support provided by the *Signavio* system.

[12] https://bpmn.io/

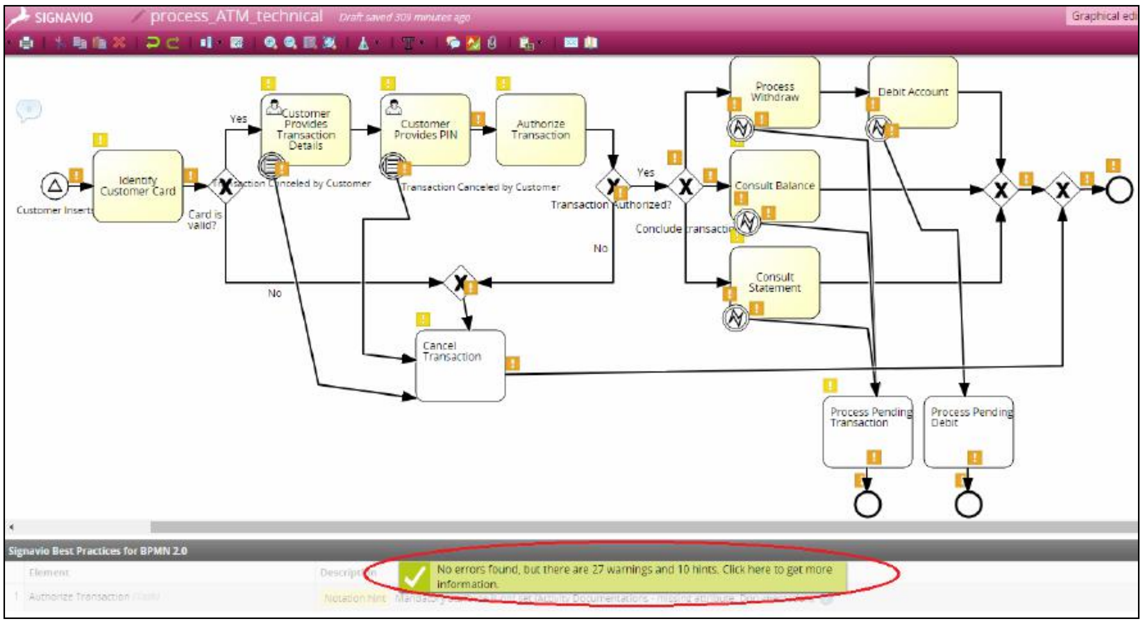
7.4.2 Results

Table 14 presents all high-level tailoring operations applied to the technical-level ATM process (Figure 46) when using the BPMNt support prototype. In this case, all operations involving only additions were executed first, as determinates the configuration *applicationOrder = FirstAddition* of the *Extend* operation. The table also details high-level tailoring operations in terms of the number of basic operations (additions, removals, and modifications) performed on specific process elements. Again, most of these basic adaptations are automatically executed by operation rules in order to ensure the correctness of the BPMN model after tailoring. The *Result* column of the table shows that all adaptations required from this tailoring scenario, involving mainly process element's refinements, were successfully supported and executed, resulting in a complete and correct tailored process model.

Since this model contains varied BPMN process elements, including ones related to exception handling, we chose it for illustrating the result of the automatic checking of BPMN modeling rules by using *Signavio*. As one can observe from the message presented at the bottom of Figure 47, no error was found by the model checking. However, because the tool verifies in the same checking several "best practices" and not only violations of rules from the BPMN standard specification, the result also presents several warnings and hints that do not have relation to the purpose of this study. The important factor for our evaluation is that the tool has found no modeling error.

7.5. Evaluation Study 3: Picture Postproduction Process in Film Industry

In this last study, we adopted real business process models from a process family. These models have been collected and published by ROSA *et al.* (2017) and are the result of a case study in picture postproduction conducted by the authors in the Australian Film, Television and Radio School (AFTRS) in Sydney. In the film industry, picture postproduction (postproduction, for short) is the process that starts after the shooting has been completed, and deals with the creative editing of the motion picture.



No errors found, but there are 27 warnings and 10 hints. Click here to get more information.

1 Authorize Transaction (Task) Notation hint Mandatory attribute not set (Activity Documentations - missing attribute: Documentation)

Figure 47: Result of checking the tailored process model in *Signavio*

Figure 48 shows six variants of the postproduction process from ROSA *et al.* (2017), which we modeled in BPMN based on the original representation using the *Event-driven Process Chains* (EPCs) language. The postproduction process starts with the receipt, from the shooting that needs to be prepared for editing. The footage can either be prepared on film (e.g., as in variant V1 in the figure), on tape (e.g., variant V2) or on both media (variant V4) depending on whether the motion picture was shot on a film roll and/or on a tape. Next, the medium is edited offline to achieve the first rough cut (therefore, activity *Edit offline* appears in all variants). Afterwards, online editing is carried out if the footage was shot on tape (variants V2 and V3), while a negmatching is performed if the footage was shot on film (e.g., variant V1).

Online editing is a cheap editing procedure applied for low-budget movies, typically shot on tape. Negmatching offers better-quality results but involves higher costs, then it is more adequate for high-budget productions, typically shot on film. The choice between online editing and negmatching depends on, e.g., budget, creativity, and type of project. One option or both needs to be chosen. Indeed, each variant in Figure 48 corresponds to a common practice in postproduction. For example, variant V1 is a typical low-budget practice (shooting and releasing on tape), whereas variant V4 represents a more expensive procedure (shooting and releasing on both tape and film).

The final step of postproduction is the finishing of the edited picture. This may involve other activities (e.g., to transfer in a telecine machine) based on the combination of editing type and final medium. The process may conclude with an optional release on a new medium (e.g., DVD or digital stream), which follows the finishing on tape or film.

7.5.1 Execution Procedure

The picture postproduction process and its variants have originally been modeled in the EPCs language with support of domain specialists from the AFTRS organization. In order to use such processes in our evaluation study, we have remodeled them in BPMN by using the modeling tool *bpmn.io*.

ROSA *et al.* (2017) present only the six process variants shown in Figure 48, but they do not mention the existence of a base (or reference) process model. In this case, i.e., when it does not exist a pre-defined base model for derivation of new variants, several strategies can be adopted to obtain a reference process model from a set of existing variants, and thus, finally, applying the BPMN*t* approach. The base model can be the most frequently used process variant, a generic model, the superset of all variants, or their intersection.

We opted by using the variant V1 of Figure 48 as our base model, since this is one of the simplest variants for postproduction. Thus, we imported the BPMN model of V1 into our support prototype and specified a BPMN*t* model for each other process variant (V2 to V6) relating them to the base model through tailoring operations, which create traceability links.

At last, in order to evaluate variant process models generated by our prototype, we have compared them to the original variant processes shown in Figure 48. Our models are complete and correct if they have the same functionality and control flow structure than original variant processes. The correctness concerning BPMN standard rules was again evaluated through automatic verification with *Signavio*.

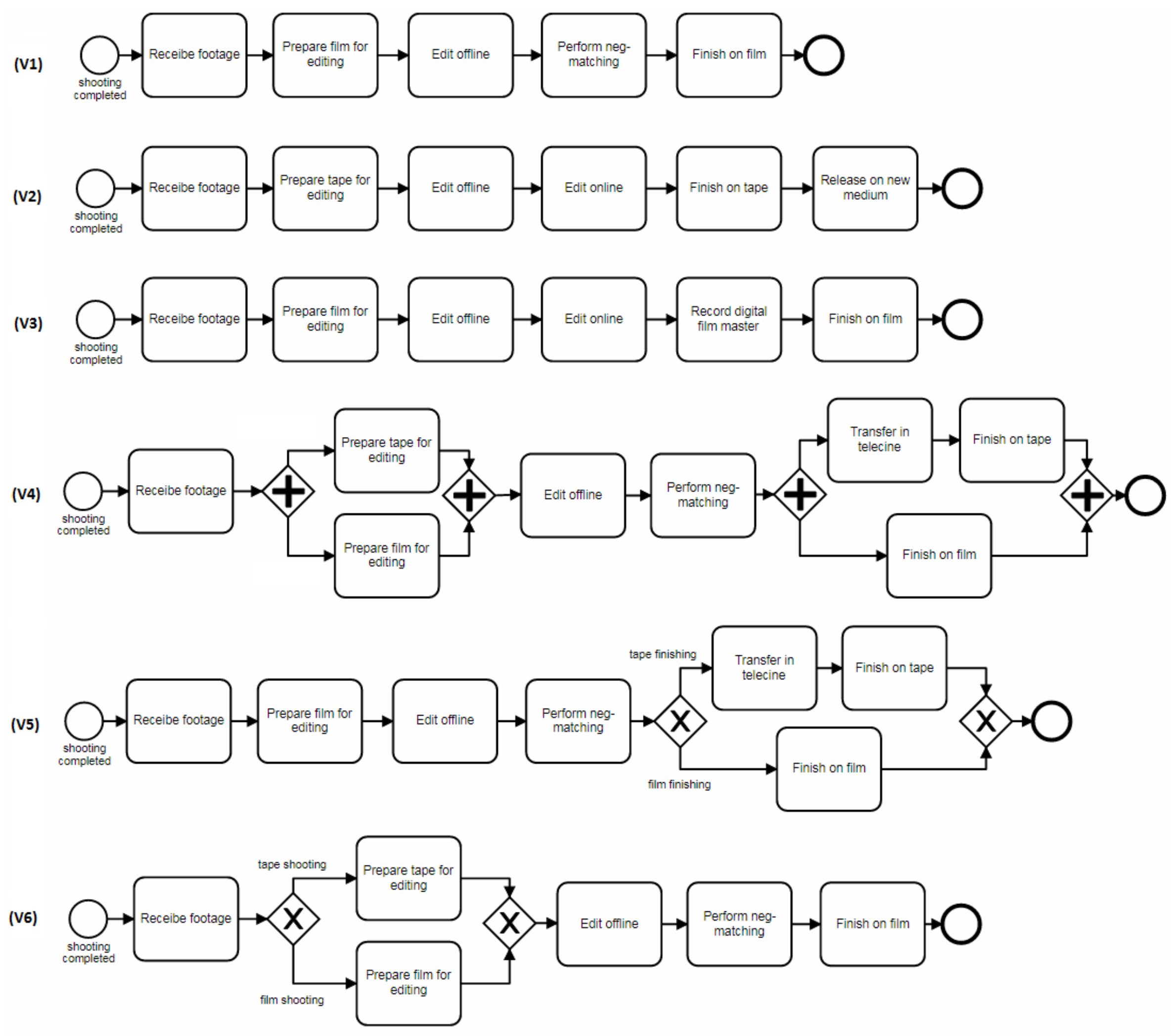


Figure 48: Variants of the picture postproduction process (V1 has been taken as base process)

7.5.2 Results

All five process variants evaluated in this study from the picture postproduction process are presented in Figure 48 (models V2 to V6). Table 15 shows high-level tailoring operations applied to each of these variant processes (following the application order top-down) when using the BPMNt support prototype. Evaluated tailoring scenarios (V2 to V6) are identified in the column ID. The table also details high-level tailoring operations in terms of the number of basic adaptations (additions, removals, and modifications) performed on specific process elements. The last column of the table (*Result*) uses the symbol '+' for showing that BPMN*t* adaptation operation was effective at executing an adaptation pattern. As one can observe from the table, all adaptations applied to evaluated tailoring scenarios in this study (V2 to V6) were successfully executed. The BPMN*t* approach was able to produce variant models with the same functionality and behavior of original variant processes, resulting in complete and correct tailored process models (concerning original models, which do not contain flow breaks). The automatic verification of well-formedness of the tailored models also did not identify no problem. We only highlight here a little difference between models V5 and V6 generated by BPMN*t* and original ones from Figure 48. While in the figure both models have all named conditional paths (i.e., specifying conditions for their execution after exclusive gateways), in models generated by BPMN*t* conditional paths comprising activities that were already in the base model are set as default paths, and, for this reason, they do not receive specific conditions. Specifying a default conditional flow is a best-practice recommended by the BPMN specification, since it avoids that no conditional flow is selected, and does not change the process behavior concerning the specification presented in Figure 48 for V5 and V6 models.

7.6. Threats to Validity

The validity of a study denotes the trustworthiness of the results, to what extent the results are true and not biased by the researcher' subjective point of view (RUNESON and HÖST, 2009). Thus, this evaluation is subject to the following main threats.

Internal validity: The trust on the correct reality representation of the process models obtained from the literature is an uncontrolled factor. To minimize this threat, we have selected as data sources only case studies that describe the target domain and mention the exact source from which process models and variation scenarios were captured. Moreover, regarding the study based on the picture postproduction process, original process models were represented in the modeling language EPCs and were converted to BPMN by the researcher of this thesis. In order to minimize any possible misalignment during this process, we have followed guidelines for conversion between these models (ROSA *et al.*, 2017). Another threat to internal validity is related to the choice of the *Signavio* BPMN model checker, which cannot have identified some rule violations in our models.

Table 14. Results of BPMN*t* process tailoring involving the ATM Process

ID	*BPMNt High-level Operation*	*Added (Contribute)*					*Removed(Suppress)*					*Modified (Modify)*					*Result*
		Task	*Subproc.*	*Gateway*	*Event*	*Flow*	*Task*	*Subproc.*	*Gateway*	*Event*	*Flow*	*Task*	*Subproc.*	*Gateway*	*Event*	*Flow*	
1	Extend *ATM Process* (applicationOrder = FirstAddition)	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	+
	Add Exception Flow *Transaction Canceled by Customer*	-	-	-	2	2	-	-	-	-	-	-	-	-	-	-	+
	Rename task *Validate PIN* (to *Authorize Transaction*)	-	-	-	-	-	-	-	-	-	-	1	-	-	-	-	+
	Rename gateway *PIN is valid?* (to *Transaction Authorized?*)	-	-	-	-	-	-	-	-	-	-	-	-	1	-	-	+
	Serial Insert *Debit Account*	1	-	-	-	2	-	-	-	-	1	-	-	-	-	-	+
	Add Exception Handler *Process Pending Transaction*	-	1	-	4	4	-	-	-	-	-	-	-	-	-	-	+
	Add Exception Handler *Process Pending Debit*	-	1	-	2	2	-	-	-	-	-	-	-	-	-	-	+
	Specialize *Customer Insert Card into ATM*	-	-	-	1	-	1	-	-	1	1	-	-	-	-	1	+
	Delete *Customer Selects Transaction*	-	-	-	-	-	1	-	-	-	1	-	-	-	-	1	+
	Split *Cancel Transaction*	-	1	-	-	-	1	-	-	-	-	-	-	-	-	4	+

Table 15. Results of BPMNt process tailoring involving the Picture Postproduction Process

ID	*BPMNt High-level Operation*	*Added (Contribute)*					*Removed(Suppress)*					*Modified (Modify)*					*Result*
		Task	*Subproc.*	*Gateway*	*Event*	*Flow*	*Task*	*Subproc.*	*Gateway*	*Event*	*Flow*	*Task*	*Subproc.*	*Gateway*	*Event*	*Flow*	
V2	Extend *Postproduction Process*	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	+
	Replace *Prepare film for editing* by *Prepare tape for editing*	1	-	-	-	-	1	-	-	-	-	-	-	-	-	2	+
	Replace fragment (*Perform neg-matching, Finish on film)* by fragment (*Edit online, Finish on tape, Release on new medium*)	3	-	-	-	2	2	-	-	-	1	-	-	-	-	2	+
V3	Extend *Postproduction Process*	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	+
	Replace *Perform neg-matching* by fragment (*Edit online, Record digital film master*)	2	-	-	-	1	1	-	-	-	-	-	-	-	-	2	+
V4	Extend *Postproduction Process*	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	+
	Parallel Insert *Prepare tape for editing*	1	-	2	-	4	-	-	-	-	-	-	-	-	-	2	+
	Parallel Insert fragment (*Transfer in telecine, Finish on tape*)	2			-	5	-	-	-	-	-	-	-	-	-	2	+
V5	Extend *Postproduction Process*	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	+
	Conditional Insert fragment (*Transfer in telecine, Finish on tape*)	2	-	2	-	5	-	-	-	-	-	-	-	-	-	2	+
V6	Extend *Postproduction Process*	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	+
	Conditional Insert *Prepare tape for editing*	1	-	2	-	4	-	-	-	-	-	-	-	-	-	2	+

External validity: Although different domains and use situations have been evaluated in this study, it is not possible to claim that it was exhaustive and its results directly apply to other domains, contexts or use situations. Further studies with a larger number of process models and in different domains and contexts should be performed. However, results of this study provide an indication about the applicability and effectiveness of the BPMN*t* solution in real process tailoring scenarios.

Conclusion validity: The main threat for conclusions of this study is related to the limited number of evaluated scenarios. It is difficult to obtain data to drive researches on process adaptation. Companies which adopt process modeling usually consider process artifacts extremely sensitive and confidential. We obtained access to people and artifacts from the SIGA Project, a collaborative system development project involving several Brazilian research institutions. However, because mining the artifact repository and applying compliance analysis techniques to identify variances between the process models is a laborious and time-consuming effort, we could obtain just a few tailoring scenarios from the aforementioned project. Likewise, we believe other researches reporting process variations from real contexts faced similar difficulties, which would explain the limited number of case studies (and evaluated variants) reported in the literature on process adaptation.

7.7. Conclusions

Finally, we conclude this chapter by answering our initial evaluation questions:

- **RQ1:** *Is the BPMNt extension based on high-level tailoring operations capable of specifying the adaptation needs from the evaluated contexts?*

  Based on scenarios we evaluated from the three studies presented in this chapter, we can state that the BPMN*t* extension based on high-level operations and its support prototype were capable to specify all adaptations needed to produce tailored process models in these contexts, which involved variations related to the control-flow perspective. This conclusion is based on the analysis of results from the three conducted studies, which are presented in Table 13, Table 14, and Table 15. As one can observe from the *Result* column of these tables, all adaptation operations required in the evaluated scenarios were successly executed (as indicated by the symbol '+'). Moreover, in no of the three studies was reported the necessity of some adaptation that is not supported by the BPMN*t* solution. However, more research is necessary to extend the evaluation with additional real-world data and potentially identify test scenarios in which our approach needs to improve besides the known limitations, such as the absence of support for adaptations on other process perspectives (e.g., data and resource ones).

- **RQ2:** *Are the BPMN process models generated by the BPMNt solution correct in the evaluated contexts?*

  Based on scenarios we evaluated from the three presented case studies, we can also state that BPMN models generated by the BPMN*t* solution were correct with respect to their structural correctness. No produced model by BPMNt presented disconnected process nodes.

  Concerning BPMN specification's constraints, according to results of the automatic verification executed by the *Signavio* BPMN model checker, no rule violation was identified from the evaluated contexts. However, since this checker does not verify all control-flow rules of the BPMN specification presented in Appendix 1, we cannot claim that BPMN process models generated by the BPMN*t* solution are fully correct in respect to the BPMN specification. This tool verifies only control-flow BPMN rules related to the correct use of Gateways, according to available information in its website[13]. Other studies are still necessary to evidence these conclusions or identifying improvement opportunities besides the known limitations related to this research question. For example, we know that BPMN*t* does not ensure model correctness regarding some very specific control-flow constraints of the BPMN standard that are related to compensation events and activities and other elements usually applied in executable models.

In summary, despite the little number of case studies and considered models, we believe evaluations based on these contexts provide initial indication of the applicability and effectiveness of our solution for adapting typical BPMN process models from contexts in the SPE and BPM domains. But it is also important to highlight that in practice there may exist contexts requiring tailoring operations uncovered by the BPMN*t*

[13] https://www.modeling-guidelines.org/categories/process-structure/

extension so far, as well as situations that lead to failures in the generation of tailored models.

CHAPTER VIII

8. Conclusion

This chapter presents our final remarks about the presented solution, including main contributions (Section 8.2), publication results (Section 8.3), limitations of the research (Section 8.4) and its implications and future perspectives (Section 8.5).

8.1. Summary

Nowadays, there is a consensus that managing processes is essential for organizational performance and, for this reason, process-oriented approaches are already institutionalized in most organizations (SHARP and MCDERMOTT, 2009). In the context of an organization, a process defines how its activities are structured in a coordinated manner in order to reach business objectives (WESKE, 2007). However, processes are not static and often need to be adapted to specific contexts where they will be applied or improved due to changing requirements or organizational learning. In this scenario, techniques for process adaptation play an important role, since they enable the development of new processes from existing ones by refining and/or modifying their definitions.

In order to effectively adapt organizational processes is recommended to explicitly represent them through models. The development of models contributes reducing ambiguities and facilitating the communication of processes. In this direction, the Business Process Model and Notation (BPMN) is an ISO and OMG standard for modeling business processes and a *de-facto* standard in professional practice (CHINOSI and TROMBETTA, 2012). However, although BPMN has been extensively used for modeling processes, its current specification (OMG, 2011) still does not have any mechanism to support users in activities related to process adaptation (tailoring). Likewise, the most prominent BPM systems based on this technology also do not provide support for such activity.

As a consequence, process variants (which share common process parts) are usually defined and maintained in separate process models without any connection with each other (HALLERBACH, BAUER and REICHERT, 2009). Considering the large number of variants that generally occur in practice, this approach requires significant effort for creating and maintaining process variants.

From a literature review, we have identified researches proposing to extend BPMN for supporting techniques of variability modeling. However, such techniques are not appropriated for application domains in which process variations are difficult to predict, such as in software development processes. Thus, the lacks of support for process adaptation from the BPMN standard and limitations from its extension proposals found currently in the literature motivated this thesis research. We argued BPMN needed a flexible and comprehensive mechanism to address process adaptation in conjunction with specific concepts that could be applied in the different contexts in which BPMN models are produced.

The objective of this research was to provide a BPMN-compliant extension and associated infrastructure for specifying flexible process tailoring on models produced with this language in different application contexts. BPMN is a *de-facto* standard for business process modeling, which focuses on the representation of the process behavior. However, BPMN can also succeed in representing the behavior of software processes (DUMAS and PFAHL, 2016) (CAMPOS and OLIVEIRA, 2013), since they are also a type of business process. In this way, we have designed a tailoring solution optimized for BPMN and have applied it in these two domains of processes, Software Process Engineering (SPE) and Business Process Management (BPM). In particular, we believe focusing on this technology could facilitate the adoption of the solution, since it may more easily be integrated to BPMN-compliant tools.

Based on evaluation studies using realistic process tailoring scenarios, it was possible to obtain indications about the feasibility of the proposed solution, since it was capable of representing all evaluated tailoring scenarios, producing tailored process models in accordance with obtained reference models and structurally correct. However, the well-formedness of the model regarding semantic rules of the BPMN specification could not be entirely evaluated, since we did not find a publicly available automatic model checker that covers all these rules (Appendix 1). Thus, it was possible to confirm the model well-formedness only concerning BPMN rules related to the use of Gateways, which are supported by the *Signavio* checker (used in our studies).

According to CORREIA (2014), this checker has the better rule coverage among available tools for this purpose.

8.2. Contributions

The main contributions of this research include:

1) **The BPMN meta-model extension called SPEM-based BPMN*t*** (depicted in Chapter 5). This proposal extended the BPMN meta-model for including tailoring support similar to the one provided by the SPEM meta-model, which is an OMG standard for software process modeling. This extension is compliant with the standard extension mechanism of BPMN (therefore, conservative) and allows adding SPEM-based tailoring operations to BPMN process elements as extension elements. This proposal has been implemented and applied for representing realistic software process adaptation scenarios in the context of a system development project. Results of this study showed that the proposal could successfully adapt the considered BPMN-based software process models.
2) **The BPMN meta-model extension called Pattern-based BPMN*t*** (depicted in Chapter 6). This proposal extended the BPMN meta-model for including tailoring support based on high-level operations, which encapsulate a set of basic operations in order to abstract the user from details of process model's transformation. These operations have been derived from *adaptation patterns* (WEBER *et al.*, 2008) and *refinement patterns* (BRANCO *et al.*, 2014) identified for the BPM domain. This proposal has also been implemented and applied for representing realistic process adaptation scenarios from different application contexts. Results of this study showed that the proposal was capable of successfully adapting the BPMN process models from the evaluated contexts.
3) **A catalog of high-level tailoring operations for BPMN**: The catalog specifies tailoring possibilities, i.e., adaptation operations, and their constraints (pre- and post-conditions) aiming at supporting process designers in specifying BPMN-based process adaptation scenarios while ensuring the correctness of the tailored process model regarding semantic constraints of the BPMN language and integrity of the process flow. The catalog contains a set of high-level operations that meet the mentioned requirements. Such operations were derived from BPM patterns and aim at supporting different types of adaptation needs, covering structural adaptations of

BPMN models (e.g., move or insert elements) as well as refinements that do not essentially change the process behavior (e.g., split or specialize a task). Each high-level operation from the catalog has been designed to correspond for a user's adaptation intent, aiming at improving the understandability of operation semantics and facilitating the posterior understanding of performed changes. These operations also meet most of control-flow requirements posed by the BPMN standard (OMG, 2011), enforcing well-formedness rules defined in the specification of the language (Appendix 1).

4) **A built-in change traceability mechanism**: Tailoring operations applied to a variant process are recorded into the variant model itself as extension information and configuration parameters of these operations are used as traceability links, connecting new variant process elements to adapted base process elements. These links are created and kept in order to facilitate the identification of the base process elements that are affected by adaptations (operations) after tailoring.
5) **A set of rules associated to tailoring operations that aims at ensuring the well-formedness of tailored models regarding the BPMN specification**. The applicability of tailoring operations in each specific situation is previously checked before executing the operation (e.g., it is not possible to insert a new process element after an end event). A complete tailoring specification is allowed executing only when there is no violation of operation pre-conditions. Afterwards, the application of each tailoring operation can trigger some "action rules" that modify the resulting process model in order to correctly specify an adaptation according to semantic rules of the BPMN language (Appendix 1) or solve any structural fail (i.e., flow break). Thus, the defined set of rules prevents any structural fail or violation of rules of the BPMN specification.
6) **A support prototype for the proposed solution**: A support tool was developed to enable the use of the BPMN*t* (SPEM-based and Pattern-based) extension and its mechanisms in specifying process adaptation scenarios based on BPMN models. The prototype was built by integrating our extension definition with resources of the MDT/BPMN2 Project (MDT, 2012), which is based on the *Eclipse Modeling Framework* (EMF). In order to support the BPMN*t* extension proposed in this research, we have automatically converted its BPMN-compliant specification in XML Schema (Appendix 3) to an EMF model, and its manipulation has been integrated with the BPMN editor of the MDT project. Thus, we enabled the

addition of BPMN*t* extension elements to standard BPMN elements without requiring manual transformations between models.

8.3. Publication Results

During this research, we have achieved the following publications:

- Raquel M. Pillat and Toacy C. Oliveira. **A Representation Structure for Software Process Tailoring Based on BPMN High-Level Operations.** In: *Proceedings of the 31st Annual ACM Symposium on Applied Computing (SAC 2016),* ACM, 2016, p. 1576-1579.
- Raquel M. Pillat, Toacy C. Oliveira, Paulo S. C. Alencar, and Donald D. Cowan. **BPMNt: A BPMN Extension for Specifying Software Process Tailoring**, *Information and Software Technology*, vol. 57, January 2015, p. 95-115.
- Raquel M. Pillat, Fábio P. Basso, Toacy C. Oliveira, and Cláudia L. Werner. **Ensuring Consistency of Feature-based Decisions with a Business Rule System**. In: *Proceedings of the Seventh International Workshop on Variability Modelling of Software-intensive Systems* (VaMoS'13). ACM, Pisa, 2013, p. 1-8.
- Raquel M. Pillat and Toacy C. Oliveira. **Introducing Software Process Tailoring to BPMN: BPMNt**. In: *Proceedings of the International Conference on Software and System Process* (ICSSP'12). IEEE Computer Society, Zurich, 2012, p. 58-62.

Some publications were also achieved from related researches and collaborations:

- Fábio Basso, Raquel Pillat, Toacy Oliveira, Fabricia Roos-frantz, Rafael Frantz. **Automated design of multi-layered web information systems**. *The Journal of Systems and Software*, v. 117, 2016.
- Fábio Basso, Raquel Pillat, Fabricia Roos-frantz, Rafael Frantz. **Combining MDE and Scrum on the rapid prototyping of web information systems**. *International Journal of Web Engineering and Technology*, v. 10, 2015.
- Fabio P. Basso, Raquel M. Pillat, Rafael Frantz and Fabrícia Rooz-Frantz. **Study on Combining Model-driven Engineering and Scrum to Produce**

Web Information Systems. In: *16th International Conference on Enterprise Information Systems* (ICEIS'14), Lisbon, 2014, p. 137-144.

- Fabio P. Basso, Raquel M. Pillat, Rafael Frantz and Fabrícia Rooz-Frantz. **Assisted Tasks to Generate Pre-prototypes for Web Information Systems**. In: *16th International Conference on Enterprise Information Systems* (ICEIS'14), Lisbon, 2014, p. 14-25.
- Fabio P. Basso and Raquel M. Pillat. **Towards a Web Modeling Environment for a Model Driven Engineering Approach**. In: *III Brazilian Workshop on Model-Driven Software Development*, Natal, 2012.

Currently, we are working on an article reporting our Pattern-based BPMN*t* extension (presented in Chapter 6) that has not yet been submitted for publication.

8.4. Limitations

On the scope of the research:

- The proposal presented in this thesis focuses only on the representational aspect of adaptations involving two BPMN process models that share common process elements. We have not investigated or dealt with issues that are around the mentioned aspect, such as reuse concerns. For example, this research does not present techniques for selecting a base process model from a set of available models in a repository as well as it does not concern on the definition of reference process models that will be target of tailoring.
- This proposal has not been intended for managing variants in the context of process families (variability management). It focuses only in the relationship between two similar process models (base and variant ones), so that a model extends the behavior of another by reusing common process parts.

On the solution approach:

- This proposal does not impose constraints defining process parts that can or cannot be target of adaptations, for example, aiming at standard conformance. Since it intends to be useful in situations involving process adaptation beyond the variability management (e.g., model refinement for including technique aspects and changes aiming at process improvement), we judged to be better to let the process engineer to make these decisions.

- Adaptations are not graphically represented in any process model perspective and can only be inferred from the adaptation operations that form a given tailoring scenario. The lack of graphical representation can make difficult for a user to predict the final result of tailoring in some cases.
- The proposal does not provide mechanisms to guide the user during the specification of tailoring scenarios, which can become complex and difficult to manage when there are many operations modifying the process structure.
- Like other approaches based on BPMN models, we consider only the process control-flow perspective in our adaptation approach. This limitation is because other process perspectives are generally not represented in BPMN process models.
- In the current proposal, we do not provide mechanisms for identifying and solving possible conflicts between adaptation operations (e.g., an operation that modifies an element that has been removed). Although we created a configuration option that executes operations involving only insertion before operations involving removals in order to minimize this issue, conflicts can still occur.
- We did not formalize our rules associated with tailoring operations because we did not find a single suitable formal representation for representing them. We considered OCL, but it represents only static model rules and we have also action rules. Then we investigated the representation of business rules with a specific rule management system (Drools) and evaluated its use for managing the configuration process of a Feature Model. This experience was reported in PILLAT *et al.* (2013). However, this type of representation seems adequate only for dynamic rules, which use *runtime* information.

On the evaluation of the proposal:

- Despite we have conducted some evaluation studies based on real process adaptation data, these studies were still limited in their scope and coverage and evaluated only the feasibility of our proposal. New studies evaluating other perspectives of the proposal must be conducted to better understand its benefits and limitations.

8.5. Implications and Future Perspectives

In this research, we have dedicated considerable effort for defining our BPMN extension in a compliant way with its standard extension mechanism, which is rather peculiar and has little available guidance information. However, this initiative was indispensable for achieving standard conformity, better extension comprehensibility, model exchangeability, and support of BPMN tools (BRAUN and ESSWEIN, 2014). In fact, the conformance to the standard made simpler the development of our prototype. Therefore, we believe this characteristic of the proposal can facilitate its adoption in other BPMN tools and projects dealing with process adaptation based on BPMN models. Recently, this initiative has also motivated other researches proposing BPMN extensions to follow standard conformance (e.g., YOUSFI *et al.*, 2017; MANDAL, WEIDLICH and WESKE, 2017).

Our proposal explicitly represents adaptation decisions. This explicit information about performed changes (i.e., applied tailoring operations), and optionally about their motivation, can be especially important in domains where process tailoring decisions are still poorly understood, such as in software development processes (KALUS and KUHRMANN, 2013) (CLARKE and O'CONNOR, 2012). According to KUHRMANN (2014), the explicit representation of process tailoring enables one to analyze and understand how tailoring has been performed, and also allows, for example, to support continuous process improvement.

Indeed, a BPMN*t* model (i.e., a tailored process model) only contains specific elements of the derived process and tailoring operations that modify elements of the base model. Thus, this model can be used as documentation of the adaptation steps and may be transformed to a change list representation, which is an important asset for future revisions of the process model (TERNITÉ, 2010).

Our SPEM-based BPMN*t* extension has been designed based on the tailoring mechanism of an important technology in the SPE domain and has been evaluated with real adaptation data from this domain. Therefore, it can be an important resource for recording, analyzing and understanding behavior variations in software processes. On the other hand, our BPMN*t* extension based on high-level operations (pattern-based) allows representing adaptation operations that have increased semantic value, facilitating the comprehension of performed adaptations and requiring less effort for specification of tailoring scenarios.

Our proposal can be applied to adapt common BPMN process models, i.e. which has not been previously "prepared" for reuse (for example, by containing special configuration semantics), and it also preserves the original model, since no of its elements is directly modified. Instead, all adaptation information is expressed as extension information in the tailored process model (BPMN*t* model). These characteristics provide flexibility for our support of tailoring, which can be applied in different contexts, on traditional BPMN process models, and requiring no time or effort of preparation for the tailoring.

In future works, we intend to extend the thesis research by addressing the following issues:

- Techniques of information visualization could be explored to graphically represent dependency relationships from elements of a variant process to elements of a base process. This graphic representation of dependencies between models would facilitate the impact analysis of changes added in their future revisions. Considering our SPEM-based BPMN*t* extension, since its tailoring operations are similar to primitive ones, techniques of differential analysis *a posteriori* could be investigated to analyze a pair of base and variant models and extract information about the application of our operations, thus automatically creating tailoring relationships of the BPMN*t* extension. These relationships are important for recording, analyzing and understanding behavior variations in related processes.
- In order to meet requirements from organizations that need to retain conformity with a reference (base) model, we can further extend our proposal for including possibilities of restricting the application of certain tailoring operations. For example, in some cases can be forbidden for the user to apply tailoring operations that remove elements from the base process. In other cases, the user can only be allowed to apply refinement operations, which do not modify the process behavior.
- At last, for use in well-defined domains, where process variations can be associated for certain context variables, tailoring scenarios with BPMN*t* could be automatically configured adopting a rule-based adaptation approach. In this

case, pre-defined business rules could create specific tailoring operations that modify a base process model according to certain values in context variables.

References

AABY, A. **Introduction to Programming Languages**. 1996. Disponível em: <http://www.emu.edu.tr/aelci/Courses/D-318/D-318-Files/plbook/>. Acesso em: 10 dez. 2017.

ACTIVITI. **Activiti BPM Platform**. Disponível em: <http://www.activiti.org/>.

ALEIXO, F. A. et al. Automating the Variability Management, Customization and Deployment of Software Processes: A Model-Driven Approach. In: **Enterprise Information Systems**. Berlin, Heidelberg: Springer Berlin Heidelberg, 2011. v. 73, p. 372–387.

ALI, N. B.; PETERSEN, K.; WOHLIN, C. A systematic literature review on the industrial use of software process simulation. **Journal of Systems and Software**, v. 97, p. 65–85, nov. 2014.

ARMBRUST, O. et al. Scoping software process lines. **Software Process: Improvement and Practice**, v. 14, n. 3, p. 181–197, maio 2009.

ARMENISE, P. et al. A Survey and Assessment of Software Process Representation Formalisms. **International Journal of Software Engineering and Knowledge Engineering**, v. 03, n. 03, p. 401–426, set. 1993.

ASSY, N. et al. Deriving configurable fragments for process design. **International Journal of Business Process Integration and Management**, v. 7, n. 1, p. 2, 2014.

AYORA, C. et al. VIVACE: A framework for the systematic evaluation of variability support in process-aware information systems. **Information and Software Technology**, v. 57, p. 248–276, jan. 2015.

BARRETO, A. **Uma Abordagem para Definição de Processos baseada em Reutilização Visando à Alta Maturidade em Processos**. Doctoral Thesis — Rio de Janeiro: COPPE/UFRJ, 2011.

BASILI, V.; CALDIERA, G.; ROMBACH, H. D. Goal Question Metric (GQM) Approach. In: **Encyclopedia of Software Engineering**. Hoboken, NJ, USA: John Wiley & Sons, Inc., 1994. v. 1, p. 528–532.

BAZIRE, M.; BRÉZILLON, P. Understanding Context Before Using It. In: **Modeling and Using Context**. Berlin, Heidelberg: Springer Berlin Heidelberg, 2005. v. 3554, p. 29–40.

BECKER, J.; JANIESCH, C.; PFEIFFER, D. Reuse Mechanisms in Situational Method Engineering. In: **Situational Method Engineering: Fundamentals and Experiences**. Boston, MA: Springer US, 2007, v. 244, p. 79–93.

BENDRAOU, R.; GERVAIS, M.-P. **A Framework for Classifying and Comparing Process Technology Domains**. In: INTERNATIONAL CONFERENCE ON SOFTWARE ENGINEERING ADVANCES (ICSEA 2007). IEEE, 2007. Disponível em: <http://ieeexplore.ieee.org/document/4299888/>

BEUCHE, D.; PAPAJEWSKI, H.; SCHRÖDER-PREIKSCHAT, W. Variability management with feature models. **Science of Computer Programming**, v. 53, n. 3, p. 333–352, dez. 2004.

BRANCO, M. C. **Managing Consistency of Business Process Models across Abstraction Levels**. Doctoral Thesis — Waterloo, Canada: University of Waterloo, 2014.

BRANCO, M. C. et al. A case study on consistency management of business and IT process models in banking. **Software & Systems Modeling**, v. 13, n. 3, p. 913–940, jul. 2014.

BRAUN, R.; ESSWEIN, W. Classification of Domain-Specific BPMN Extensions. In: **The Practice of Enterprise Modeling**. Berlin, Heidelberg: Springer Berlin Heidelberg, 2014. v. 197, p. 42–57.

CAMPOS, A. L. N.; OLIVEIRA, T. Software Processes with BPMN: An Empirical Analysis. In: **Product-Focused Software Process Improvement**. Berlin, Heidelberg: Springer Berlin Heidelberg, 2013, v. 7983, p. 338–341.

CASATI, F. **Models, Semantics, and Formal Methods for the design of Workflows and their Exceptions**. Milan: University of Milano, 1998.

CHINOSI, M.; TROMBETTA, A. BPMN: An introduction to the standard. **Computer Standards & Interfaces**, v. 34, n. 1, p. 124–134, jan. 2012.

CHRISSIS, M. B.; KONRAD, M.; SHRUM, S. **CMMI for development: guidelines for process integration and product improvement**. 3rd ed. Upper Saddle River, NJ: Addison-Wesley, 2011.

CLARKE, P.; O'CONNOR, R. V. The situational factors that affect the software development process: Towards a comprehensive reference framework. **Information and Software Technology**, v. 54, n. 5, p. 433–447, maio 2012.

CLEMENTS, P.; NORTHROP, L. **Software product lines: practices and patterns**. Boston: Addison-Wesley, 2002.

COGNINI, R. et al. **Research challenges in business process adaptability**. In: PROCEEDINGS OF THE 29TH ANNUAL ACM SYMPOSIUM ON APPLIED COMPUTING (SAC'14). ACM Press, 2014. Disponível em: <http://dl.acm.org/citation.cfm?doid=2554850.2555055>

CORREIA, A. **Quality of Process Modeling Using BPMN: A Model-Driven Approach**. Doctoral Thesis — Lisboa, Portugal: Universidade Nova de Lisboa, 2014.

CORREIA, A.; ABREU, F. B. E. Adding Preciseness to BPMN Models. **Procedia Technology**, v. 5, p. 407–417, 2012.

CORREIA, A.; ABREU, F. B. E. Enhancing the Correctness of BPMN Models. In: **Improving Organizational Effectiveness with Enterprise Information Systems**. Advances in Business Information Systems and Analytics. [s.l.] IGI Global, 2015. v. 1, p. 241–261.

CUI, X. **An approach implementing template-based process development on BPMN**. In: INTERNATIONAL CONFERENCE ON COMPUTER AND INFORMATION SCIENCE (ICIS). IEEE, 2017. Disponível em: <http://ieeexplore.ieee.org/document/7960000/>.

DAVENPORT, T. H. **Process innovation: reengineering work through information technology**. Boston, Mass: Harvard Business School Press, 1993.

DERNIAME, J.-C.; KABA, B. A.; WASTELL, D. (EDS.). **Software Process: Principles, Methodology, and Technology**. Berlin, Heidelberg: Springer Berlin Heidelberg, 1999. v. 1500.

DEY, A. K.; ABOWD, G. D.; SALBER, D. A. Conceptual Framework and a Toolkit for Supporting the Rapid Prototyping of Context-Aware Applications. **Human–Computer Interaction**, v. 16, n. 2–4, p. 97–166, dez. 2001.

DIJKMAN, R. M.; DUMAS, M.; OUYANG, C. Semantics and analysis of business process models in BPMN. **Information and Software Technology**, v. 50, n. 12, p. 1281–1294, nov. 2008.

DIJKMAN, R.; ROSA, M. L.; REIJERS, H. A. Managing large collections of business process models — Current techniques and challenges. **Computers in Industry**, v. 63, n. 2, p. 91–97, fev. 2012.

DÖHRING, M.; ZIMMERMANN, B. vBPMN: Event-Aware Workflow Variants by Weaving BPMN2 and Business Rules. In: **Enterprise, Business-Process and Information Systems Modeling**. Berlin, Heidelberg: Springer Berlin Heidelberg, 2011. v. 81, p. 332–341.

DUMAS, M.; GARCÍA-BAÑUELOS, L.; POLYVYANYY, A. Unraveling Unstructured Process Models. In: **Business Process Modeling Notation**. Berlin, Heidelberg: Springer Berlin Heidelberg, 2010. v. 67, p. 1–7.

DUMAS, M.; PFAHL, D. Modeling Software Processes Using BPMN: When and When Not? In: KUHRMANN, M. et al. (Eds.). **Managing Software Process Evolution**. Cham: Springer International Publishing, 2016. p. 165–183.

EPF. **The OpenUP methodology web site**. Disponível em: <http://epf.eclipse.org/wikis/openup/>. Acesso em: 10 dez. 2017.

FERRATT, T. W.; MAI, B. **Tailoring software development**. In: PROCEEDINGS OF THE SPECIAL INTEREST GROUP ON MANAGEMENT INFORMATION SYSTEM’S 48TH ANNUAL CONFERENCE ON COMPUTER PERSONNEL RESEARCH ON COMPUTER PERSONNEL RESEARCH. ACM Press, 2010. Disponível em: <http://portal.acm.org/citation.cfm?doid=1796900.1796963>.

FRANK, U. **Conceptual modeling as the core of the information systems discipline—perspectives and epistemological challenges**. In: FIFTH AMERICAS CONFERENCE ON INFORMATION SYSTEMS. 1999.

FRECE, A.; JURIC, M. B. Modeling functional requirements for configurable content- and context-aware dynamic service selection in business process models. **Journal of Visual Languages & Computing**, v. 23, n. 4, p. 223–247, ago. 2012.

FUGGETTA, A. **Software Process: A Roadmap**. Proceedings of the Conference on The Future of Software Engineering. **Anais**...: ICSE’00. New York, NY, USA: ACM, 2000. Disponível em: <http://doi.acm.org/10.1145/336512.336521>. Acesso em: 15 jun. 2012.

GARCÍA-BORGOÑÓN, L. et al. Software process modeling languages: A systematic literature review. **Information and Software Technology**, v. 56, n. 2, p. 103–116, fev. 2014.

GINSBERG, M. P.; QUINN, L. H. **Process Tailoring and the Software Capability Maturity Model**: CMU/SEI Report Number: CMU/SEI-94-TR-024. Pittsburgh, PA, USA: Software Engineering Institute, nov. 1995. Disponível em: <https://resources.sei.cmu.edu/library/asset-view.cfm?assetid=12261>.

GOTTSCHALK, F. et al. Configurable Process Models: Experiences from a Municipality Case Study. In: **Advanced Information Systems Engineering**. Berlin, Heidelberg: Springer Berlin Heidelberg, 2009. v. 5565, p. 486–500.

GOTTSCHALK, F.; VAN DER AALST, W. M. P.; JANSEN-VULLERS, M. H. Configurable Process Models — A Foundational Approach. In: **Reference Modeling**. Heidelberg: Physica-Verlag HD, 2007. p. 59–77.

HACK, M. **Petri Net Language**. Cambridge, MA, USA: Massachusetts Institute of Technology, 1976.

HALLERBACH, A.; BAUER, T.; REICHERT, M. Capturing variability in business process models: the Provop approach. **Journal of Software Maintenance and Evolution: Research and Practice**, v. 22, n. 6–7, p. 519–546, 19 out. 2009.

HALLERBACH, A.; BAUER, T.; REICHERT, M. Configuration and Management of Process Variants. In: BROCKE, J. VOM; ROSEMANN, M. (Eds.). **Handbook on Business Process Management 1**. Berlin, Heidelberg: Springer Berlin Heidelberg, 2010. p. 237–255.

HAMMER, M.; CHAMPY, J. **Reengineering the corporation: a manifesto for business revolution**. New York: HarperBusiness Essentials, 2003.

HARMON, P. **The State of Business Process Management 2016**: BPTrends Report. [s.l.] BPTrends, mar. 2016. Disponível em: <https://www.bptrends.com/bpt/wp-content/uploads/2015-BPT-Survey-Report.pdf>.

HEBIG, R.; KHELLADI, D. E.; BENDRAOU, R. Approaches to Co-Evolution of Metamodels and Models: A Survey. **IEEE Transactions on Software Engineering**, v. 43, n. 5, p. 396–414, maio 2017.

HENDERSON, P. **Software processes are business processes too**. IEEE Comput. Soc. Press, 1994. Disponível em: <http://ieeexplore.ieee.org/document/344411/>.

HENDERSON-SELLERS, B. et al. **Situational Method Engineering**. Berlin, Heidelberg: Springer Berlin Heidelberg, 2014.

HENDERSON-SELLERS, B.; RALYTÉ, J. Situational Method Engineering: State-of-the-Art Review. **JUCS - Journal of Universal Computer Science**, n. 3, fev. 2010.

HENNINGER, S. et al. Supporting Adaptable Methodologies to Meet Evolving Project Needs. In: **Extreme Programming and Agile Methods — XP/Agile Universe 2002**. Berlin, Heidelberg: Springer Berlin Heidelberg, 2002. v. 2418, p. 33–44.

HERRMANNSDÖRFER, M.; WACHSMUTH, G. Coupled Evolution of Software Metamodels and Models. In: MENS, T.; SEREBRENIK, A.; CLEVE, A. (Eds.). **Evolving Software Systems**. Berlin, Heidelberg: Springer Berlin Heidelberg, 2014. p. 33–63.

HUMPHREY, W. S. The software engineering process: definition and scope. **ACM SIGSOFT Software Engineering Notes**, v. 14, n. 4, p. 82–83, 1989.

HURTADO ALEGRÍA, J. A. et al. **An MDE approach to software process tailoring**. In: INTERNATIONAL CONFERENCE ON SOFTWARE AND SYSTEMS PROCESS (ICSSP). ACM Press, 2011. Disponível em: <http://portal.acm.org/citation.cfm?doid=1987875.1987885>.

ISO. **ISO/IEC 15504-2:2003**. International Organization for Standardization, 2004.

ISO. **ISO/IEC 19510:2013: Information technology -- Object Management Group Business Process Model and Notation.** International Organization for Standardization, jul. 2013. Disponível em: <https://www.iso.org/standard/62652.html>.

KALUS, G.; KUHRMANN, M. **Criteria for software process tailoring: a systematic review**. In: INTERNATIONAL CONFERENCE ON SOFTWARE AND SYSTEM PROCESS (ICSSP). ACM Press, 2013. Disponível em: <http://dl.acm.org/citation.cfm?doid=2486046.2486078>.

KRUCHTEN, P. **The rational unified process: an introduction**. 3rd ed. Boston: Addison-Wesley, 2004.

KUHRMANN, M. **You can't tailor what you haven't modeled**. In: PROCEEDINGS OF THE 2014 INTERNATIONAL CONFERENCE ON SOFTWARE AND SYSTEM PROCESS (ICSSP). ACM Press, 2014. Disponível em: <http://dl.acm.org/citation.cfm?doid=2600821.2600851>.

KUHRMANN, M. et al. Flexible software process lines in practice: A metamodel-based approach to effectively construct and manage families of software process models. **Journal of Systems and Software**, v. 121, p. 49–71, nov. 2016.

KUHRMANN, M.; FERNÁNDEZ, D. M.; STEENWEG, R. **Systematic software process development: where do we stand today?**. In: INTERNATIONAL CONFERENCE ON SOFTWARE AND SYSTEMS PROCESS (ICSSP). ACM Press, 2013. Disponível em: <http://dl.acm.org/citation.cfm?doid=2486046.2486077>.

KUHRMANN, M.; FERNÁNDEZ, D. M.; TERNITÉ, T. **Realizing software process lines: insights and experiences**. In: PROCEEDINGS OF THE 2014 INTERNATIONAL CONFERENCE ON SOFTWARE AND SYSTEM PROCESS. ACM Press, 2014. Disponível em: <http://dl.acm.org/citation.cfm?doid=2600821.2600833>.

KÜSTER, J. et al. Supporting different process views through a Shared Process Model. **Software & Systems Modeling**, v. 15, n. 4, p. 1207–1233, out. 2016.

LA ROSA, M. **Modeling Business Process Variability: Are We Done Yet?**. In: INTERNATIONAL SYSTEMS AND SOFTWARE PRODUCT LINE CONFERENCE (SPLC) - VOLUME A. Sevilla, Spain: ACM Press, 2017. Disponível em: <http://dl.acm.org/citation.cfm?doid=3106195.3106196>.

LANGER, P. et al. A posteriori operation detection in evolving software models. **Journal of Systems and Software**, v. 86, n. 2, p. 551–566, fev. 2013.

LONCHAMP, J. **A structured conceptual and terminological framework for software process engineering**. In: CONFERENCE ON THE SOFTWARE PROCESS. IEEE Comput. Soc. Press, 1993. Disponível em: <http://ieeexplore.ieee.org/document/236823/>

LONIEWSKI, G.; ARMESTO, A.; INSFRAN, E. **An agile method for model-driven requirements engineering**. In: THE SIXTH INTERNATIONAL CONFERENCE ON SOFTWARE ENGINEERING ADVANCES (ICSEA). IARIA, 2011.

MAGDALENO, A. **COMPOOTIM: Em Direção ao Planejamento, Acompanhamento e Otimização da Colaboração na Definição de Processos de Software**. PhD Thesis — Rio de Janeiro: COPPE/UFRJ, 2013.

MANDAL, S.; WEIDLICH, M.; WESKE, M. Events in Business Process Implementation: Early Subscription and Event Buffering. In: **Business Process Management Forum**. Cham: Springer, 2017. v. 297, p. 141–159.

MARTÍNEZ-RUIZ, T. et al. Requirements and constructors for tailoring software processes: a systematic literature review. **Software Quality Journal**, v. 20, n. 1, p. 229–260, mar. 2012.

MARTÍNEZ-RUIZ, T.; GARCÍA, F.; PIATTINI, M. Towards a SPEM v2.0 Extension to Define Process Lines Variability Mechanisms. In: LEE, R. (Ed.). **Software Engineering Research, Management and Applications**. Berlin, Heidelberg: Springer Berlin Heidelberg, 2008. v. 150, p. 115–130.

MDT. **MDT/BPMN2 Project**. Disponível em: <http://wiki.eclipse.org/MDT/BPMN2>.

MENDLING, J. **Metrics for Process Models**. Berlin, Heidelberg: Springer Berlin Heidelberg, 2008. v. 6.

MENS, T.; VAN GORP, P. A. Taxonomy of Model Transformation. **Electronic Notes in Theoretical Computer Science**, v. 152, p. 125–142, mar. 2006.

MOON, M.; HONG, M.; YEOM, K. **Two-Level Variability Analysis for Business Process with Reusability and Extensibility**. In: COMPUTER SOFTWARE AND APPLICATIONS (COMPSAC). IEEE, 2008. Disponível em: <http://ieeexplore.ieee.org/document/4591567/>.

NUNES, V. **Dynamic Process Adaptation: Planning in a Context-Aware Approach**. PhD Thesis — Rio de Janeiro: COPPE/UFRJ, 2014.

OLIVEIRA, E. A. et al. SMartySPEM: A SPEM-Based Approach for Variability Management in Software Process Lines. In: **Product-Focused Software Process Improvement**. Berlin, Heidelberg: Springer Berlin Heidelberg, 2013. v. 7983, p. 169–183.

OMG. **Software Process Engineering Metamodel (SPEM) 2.0 Specification**. Object Management Group, 2008. Disponível em: <http://www.omg.org/spec/SPEM/2.0/PDF/>.

OMG. **Business Process Model and Notation (BPMN) Version 2.0**. Object Management Group, jan. 2011. Disponível em: <http://www.omg.org/spec/BPMN/2.0>.

OMG. **Unified Modeling Language version 2.5**. Object Management Group, maio 2015. Disponível em: <http://www.omg.org/spec/UML/2.5/>.

OSTERWEIL, L. **Software processes are software too**. In: PROCEEDINGS OF THE 9TH INTERNATIONAL CONFERENCE ON SOFTWARE ENGINEERING (ICSE ’87). Monterey, California, USA: IEEE, 1987.

PAIGE, R. F.; MATRAGKAS, N.; ROSE, L. M. Evolving models in Model-Driven Engineering: State-of-the-art and future challenges. **Journal of Systems and Software**, v. 111, p. 272–280, jan. 2016.

PANDE, P. S.; NEUMAN, R. P.; CAVANAGH, R. R. **The Six Sigma way: how GE, Motorola, and other top companies are honing their performance**. New York: McGraw-Hill, 2000.

PASCALAU, E. et al. Partial process models to manage business process variants. **International Journal of Business Process Integration and Management**, v. 5, n. 3, p. 240, 2011.

PEDREIRA, O. et al. A systematic review of software process tailoring. **ACM SIGSOFT Software Engineering Notes**, v. 32, n. 3, p. 1, 2007.

PEFFERS, K. et al. A Design Science Research Methodology for Information Systems Research. **Journal of Management Information Systems**, v. 24, n. 3, p. 45–77, dez. 2007.

PEREIRA, E. B.; BASTOS, R. M.; OLIVEIRA, T. **Process tailoring based on well-formedness rules**. In: PROCEEDINGS OF THE 20TH INTERNATIONAL CONFERENCE ON SOFTWARE ENGINEERING AND KNOWLEDGE ENGINEERING (SEKE'08). San Francisco, CA, USA: 2008.

PEREIRA, E. B.; BASTOS, R. M.; OLIVEIRA, T. C. **A Systematic Approach to Process Tailoring**. IEEE, mar. 2007.

PILLAT, R. M. et al. **Ensuring consistency of feature-based decisions with a business rule system**. In: INTERNATIONAL WORKSHOP ON VARIABILITY MODELLING OF SOFTWARE-INTENSIVE SYSTEMS (VAMOS). Pisa - Italy, ACM Press, 2013.

PILLAT, R. M. et al. BPMNt: A BPMN extension for specifying software process tailoring. **Information and Software Technology**, v. 57, p. 95–115, jan. 2015.

PILLAT, R. M.; OLIVEIRA, T. C. **A representation structure for process tailoring based on BPMN high-level operations**. In: ANNUAL ACM SYMPOSIUM ON APPLIED COMPUTING (SAC). ACM Press, 2016, p. 1576-1579.

PILLAT, R. M.; OLIVEIRA, T. C.; FONSECA, F. L. **Introducing Software Process Tailoring to BPMN: BPMNt**. In: INTERNATIONAL CONFERENCE ON SOFTWARE AND SYSTEM PROCESS (ICSSP). IEEE, jun. 2012.

POLYVYANYY, A. et al. **Restructuring BPMN diagrams using BPStruct**. Signavio BPM. Disponível em: <https://www.signavio.com/bpm-academic-initiative/>. Acesso em: 12 dez. 2017.

POPP, R.; KAINDL, H. **Automated refinement of business processes through model transformations specifying business rules**. In: INTERNATIONAL CONFERENCE ON RESEARCH CHALLENGES IN INFORMATION SCIENCE (RCIS). IEEE, 2015.

PRESSMAN, R. S. **Software engineering: a practitioner's approach**. Eighth edition ed. New York, NY: McGraw-Hill Education, 2015.

RAJABI, B. A.; LEE, S. P. **Change Management in Business Process Modeling Survey**. Information Management and Engineering, 2009. In: INTERNATIONAL CONFERENCE ON INFORMATION MANAGEMENT AND ENGINEERING (ICIME '09). IEEE, 2009.

RALYTÉ, J.; DENECKÈRE, R.; ROLLAND, C. Towards a Generic Model for Situational Method Engineering. In: **Advanced Information Systems Engineering**. Berlin, Heidelberg: Springer Berlin Heidelberg, 2003. v. 2681, p. 95–110.

RECKER, J. et al. **Do Process Modelling Techniques Get Better? A Comparative Ontological Analysis of BPMN**. In: 16TH AUSTRALASIAN CONFERENCE ON INFORMATION SYSTEMS, SYDNEY AND AUSTRALIA, AUSTRALASIAN CHAPTER OF THE ASSOCIATION FOR INFORMATION SYSTEMS. 2005.

REICHERT, M. et al. **Adaptive Process Management with ADEPT2**. In: PROCEEDINGS OF THE 21ST INTERNATIONAL CONFERENCE ON DATA ENGINEERING (ICDE'05). Tokoyo, Japan: IEEE, 2005.

REICHERT, M.; WEBER, B. **Enabling flexibility in process-aware information systems: challenges, methods, technologies**. Berlin, New York: Springer, 2012.

REIS, C. **Uma Abordagem Flexível para Execução de Processos de Software Evolutivos**. PhD Thesis - Porto Alegre: PPGC- UFRGS, 2003.

ROBSON, C. **Real world research: a resource for users of social research methods in applied settings**. 3. ed. Chichester: Wiley, 2011.

ROSA, M. L. et al. Business Process Variability Modeling: A Survey. **ACM Computing Surveys**, v. 50, n. 1, p. 1–45, mar. 2017.

RUIZ-RUBE, I. et al. Uses and applications of Software & Systems Process Engineering Meta-Model process models. A systematic mapping study: Uses and Applications of SPEM process models. **Journal of Software: Evolution and Process**, v. 25, n. 9, p. 999–1025, set. 2013.

RUNESON, P.; HÖST, M. Guidelines for conducting and reporting case study research in software engineering. **Empirical Software Engineering**, v. 14, n. 2, p. 131–164, abr. 2009.

SANTOS, E. et al. **A Goal-Oriented Approach for Variability in BPMN**. In: PROCEEDINGS OF THE 13TH WORKSHOP ON REQUIREMENTS ENGINEERING (WER). 2010.

SANTOS, R. M. S.; OLIVEIRA, T. C.; ABREU, F. B. **Mining software development process variations**. In: ACM SYMPOSIUM ON APPLIED COMPUTING (SAC). ACM Press, 2015.

SCHMIDT, D. C. Model-Driven Engineering. **Computer**, v. 39, n. 2, p. 25–31, fev. 2006.

SCHNIEDERS, A.; PUHLMANN, F. **Variability mechanisms in e-business process families**. In: PROCEEDINGS OF THE 9TH INTERNATIONAL CONFERENCE ON BUSINESS INFORMATION SYSTEMS (BIS). GI, 2006.

SCHONENBERG, M. H. et al. **Towards a taxonomy of process flexibility (extended version)**: BPM Center Report No. BPM-07-11. [s.l.] BPM Center, 2007. Disponível em: <http://bpmcenter.org/wp-content/uploads/reports/2007/BPM-07-11.pdf>.

SCHUNSELAAR, D. M. M. et al. Creating Sound and Reversible Configurable Process Models Using CoSeNets. In: **Business Information Systems**. Berlin, Heidelberg: Springer Berlin Heidelberg, 2012. v. 117, p. 24–35.

SEI. **Standard CMMI Appraisal Method for Process Improvement (SCAMPI), version 1.1: Method definition document (CMU/SEI-2001-HB-001)**. Software Engineering Institute, 2001.

SELIC, B. The pragmatics of model-driven development. **IEEE Software**, v. 20, n. 5, p. 19–25, set. 2003.

SHARP, A.; MCDERMOTT, P. **Workflow modeling: tools for process improvement and applications development**. 2ª ed. Boston: Artech House, 2009.

SUTTON, S. M.; OSTERWEIL, L. J. **Product families and process families**. In: PROCEEDINGS OF THE 10TH INTERNATIONAL SOFTWARE PROCESS WORKSHOP. IEEE Comput. Soc, 1998.

TEIXEIRA, E. **OdysseyProcessReuse: Uma Metodologia para Engenharia de Linha de Processos de Software Baseada em Componentes**. PhD Thesis — Rio de Janeiro: COPPE/UFRJ, 2016.

TERNITÉ, T. **Process Lines: A Product Line Approach Designed for Process Model Development**. In: 35TH EUROMICRO CONFERENCE ON SOFTWARE ENGINEERING AND ADVANCED APPLICATIONS. IEEE, 2009.

TERNITÉ, T. **Variability of Development Models**. PhD Thesis — TU Clausthal, 2010.

TERNITÉ, T.; KUHRMANN, M. **Das V-Modell XT 1.3 Metamodell**. Technische Universität München, 2009.

THOM, L.; IOCHPE, C. **BPMN 2.0 Poster**. Berliner BPM-Offensive, [s.d.]. Disponível em: <http://www.bpmb.de/images/BPMN2_0_Poster_EN.pdf>.

VAN DER AALST, W. M. P. Business Process Management: A Comprehensive Survey. **ISRN Software Engineering**, v. 2013, p. 1–37, 2013.

VAN DER AALST, W. M. P.; JABLONSKI, S. Dealing with workflow change: identification of issues and solutions. **International Journal of Computer Systems Science & Engineering**, v. 15, n. 5, p. 267–276, set. 2000.

VAN DER AALST, W. M. P.; TER HOFSTEDE, A. H. M. YAWL: yet another workflow language. **Information Systems**, v. 30, n. 4, p. 245–275, jun. 2005.

VAN DER AALST, W. M. P.; TER HOFSTEDE, A. H. M.; WESKE, M. Business Process Management: A Survey. In: TER HOFSTEDE, A. (Ed.). **Business Process Management**. Berlin, Heidelberg: Springer Berlin Heidelberg, 2003. v. 2678, p. 1–12.

VARA, J. M. et al. Dealing with Traceability in the MDDof Model Transformations. **IEEE Transactions on Software Engineering**, v. 40, n. 6, p. 555–583, jun. 2014.

VERGIDIS, K.; TIWARI, A.; MAJEED, B. Business Process Analysis and Optimization: Beyond Reengineering. **IEEE Transactions on Systems, Man, and Cybernetics, Part C (Applications and Reviews)**, v. 38, n. 1, p. 69–82, jan. 2008.

WANG, H. J.; ZHAO, J. L. Constraint-centric workflow change analytics. **Decision Support Systems**, v. 51, n. 3, p. 562–575, jun. 2011.

WASHIZAKI, H. **Deriving Project-Specific Processes from Process Line Architecture with Commonality and Variability**. In: IEEE INTERNATIONAL CONFERENCE ON INDUSTRIAL INFORMATICS. IEEE, ago. 2006.

WEBER, B. et al. Refactoring large process model repositories. **Computers in Industry**, v. 62, n. 5, p. 467–486, jun. 2011.

WEBER, B.; REICHERT, M.; RINDERLE-MA, S. Change patterns and change support features – Enhancing flexibility in process-aware information systems. **Data & Knowledge Engineering**, v. 66, n. 3, p. 438–466, Setembro 2008.

WESKE, M. **Business process management: concepts, languages, architectures**. New York: Springer, 2007.

XU, P. **Knowledge Support in Software Process Tailoring**. In: PROCEEDINGS OF THE 38TH ANNUAL HAWAII INTERNATIONAL CONFERENCE ON SYSTEM SCIENCES (HICSS '05). Big Island, HI, USA: IEEE, 2005.

XU, P.; RAMESH, B. Using Process Tailoring to Manage Software Development Challenges. **IT Professional**, v. 10, n. 4, p. 39–45, jul. 2008.

YOON, I.-C.; MIN, S.-Y.; BAE, D.-H. **Tailoring and verifying software process**. IEEE Comput. Soc, 2001. Disponível em: <http://ieeexplore.ieee.org/document/991478/>.

YOUSFI, A. et al. Towards uBPMN-Based Patterns for Modeling Ubiquitous Business Processes. **IEEE Transactions on Industrial Informatics**, p. 1–1, 2017.

YOUSFI, A.; SAIDI, R.; DEY, A. K. Variability patterns for business processes in BPMN. **Information Systems and e-Business Management**, v. 14, n. 3, p. 443–467, ago. 2016.

ZAKARIA, N. A.; IBRAHIM, S.; MAHRIN, M. N. **The state of the art and issues in software process tailoring**. In: INTERNATIONAL CONFERENCE ON SOFTWARE ENGINEERING AND COMPUTER SYSTEMS (ICSECS). IEEE, ago. 2015. Disponível em: <http://ieeexplore.ieee.org/document/7333097/>.

ZAVE, P. Feature interactions and formal specifications in telecommunications. **Computer**, v. 26, n. 8, p. 20–28, ago. 1993.

ZHANG, H.; HAN, W.; OUYANG, C. Extending BPMN for Configurable Process Modeling. **Advances in Transdisciplinary Engineering**, p. 317–330, 2014.

Appendix 1: BPMN Well-Formedness Rules

Table 16 lists all BPMN well-formedness rules related to the process' control-flow perspective that we have taken into account for producing correct tailored process models regarding the BPMN specification when using BPMN*t* tailoring operations. It is important to highlight that our solution works only with BPMN models that explicitly represent start and end events, which is a best-practice recommendation (OMG, 2011).

Our list does not contain some control-flow rules from the original list in (CORREIA, 2014), which represent limitations of our solution. Such missing rules are:

- Rules related to *Message Flow* (element used in the modeling of Collaborations);
- Rules related to event of type *Link*;
- Rules related to matching between *Catch* and *Throw Events*.
- Rules for merging exception flows with the normal flow of the process;
- Rules related to BPMN elements usually represented in executable models (BRANCO, 2014): Compensation activities and events, Transaction Sub-Process and Event Sub-Process.

Table 16. Subset of BPMN control-flow well-formedness rules considered in this research (adapted from CORREIA, 2014)

ID	BPMN Control-Flow Well-formedness Rules
1	**Process:** A Top-Level Process can only be instantiated by a restricted set of Start Event types (None, Message, Timer, Conditional, or Signal).
2	**Sub-Process:** A Sub-Process can only have one None Start Event.
3	**Flow Node:** A Flow Node, in a container that includes start and end events, must have at least one incoming or one outgoing sequence flow.
4	**Event:** Only some predefined types of Start, Intermediate and End Events are allowed in specific contexts.
5	Incoming Sequence Flow not allowed in a Start Event.
6	Outgoing Sequence Flow not allowed in an End Event.
7	Intermediate Events used within normal flow require incoming and outgoing Sequence Flows.
8	Explicit Start/End Events do not allow Activities or Gateways without incoming/outgoing Sequence Flow.
9	Error intermediate events can only be attached to activity boundaries.
10	Catch Error Event must trigger an exception flow.
11	

12	A Throwing Error Event must be an End Event.
13	Catch Escalation Events can only be attached to activity boundaries.
14	Catch Escalation Event must trigger an exception flow.
15	A Boundary Event must have exactly one outgoing Sequence Flow (unless it has the Compensation type) A Boundary Event must not have incoming Sequence Flow
	Gateway:
16	A Parallel Gateway joins only non-exclusive Sequence Flows
17	A join Exclusive Gateway must merge only exclusive Sequence Flows
18	A Gateway must have either multiple incoming Sequence Flow or multiple outgoing Sequence Flow (i.e., it must merge or split the flow).
19	A Gateway with a *gatewayDirection* of converging must have multiple incoming Sequence Flow, but must not have multiple outgoing Sequence Flow.
20	A Gateway with a *gatewayDirection* of diverging must have multiple outgoing Sequence Flow, but must not have multiple incoming Sequence Flow.
21	An Event-Based Gateway must have two or more outgoing Sequence Flow.
22	A Conditional Sequence Flow must not be used if the source Gateway is of type Event-Based.
23	A condition Expression must be defined if the Source of the Sequence Flow is an Exclusive or Inclusive Gateway.
24	Target of the Event-Based Gateway must be Receive Task or specific Intermediate Catch Event (Message, Signal, Timer, or Conditional).
25	If Message Intermediate Catch Events are used as Target for the Gateway's outgoing Sequence Flow, then Receive Tasks must not be used and vice versa.
26	Target elements in an Event-Based Gateway configuration must not have any additional incoming Sequence Flow (other than that from the Event Gateway).
27	A Parallel Gateway must not have outgoing Conditional Sequence Flow.
	Sequence Flow:
28	A conditional Sequence Flow cannot be used if there is only one sequence flow out of the element.
29	Sequence Flows cannot cross container boundaries.
30	The source and target must not be the same.

Appendix 2: Catalog of BPMN*t* Tailoring Operations

EXTEND

Purpose: Reuse the elements structure of an existing process model to derive a new one and enable the use of tailoring operations.

Motivation: A new process TP shares with another existing one part of its elements structure, therefore the new process should be defined by adapting the existing one through tailoring operations.

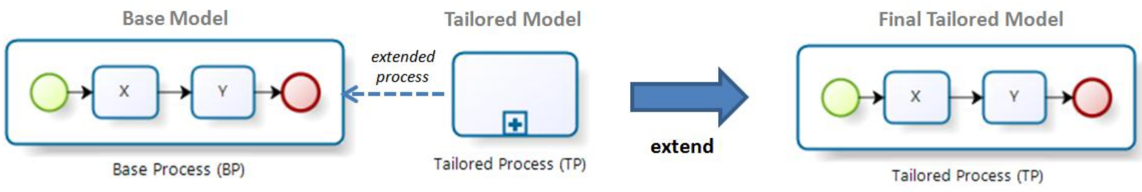
Description: A process or subprocess (*Tailored Process* - TP) reuses the structure of another (*Base Process* - BP), inheriting its complete elements structure. The parameter *applicationOrder* specifies the criterion used for applying tailoring operations defined in a *Tailored Process (TP)*. If *applicationOrder = FirstAddition* (default), then all operations involving only insertions of elements are executed first. On the other hand, if the value of the parameter is *Free*, then the order of operation application is given by the order in which they were specified in the *Tailored Process.*

Source Element Type(s): Process, or Subprocess.

Parameters: extendedProcess(BPMN:FlowElementsContainer), applicationOrder (OrderType = FirstAddition {Free | FirstAddition })

Post-conditions: Enables the application of other tailoring operations from the *Tailored Process* (source) to the *Base Process* (target).

Representation:



Example: In the SIGA project, specific execution cases (i.e., instances) of the defined process for the project (i.e., base process) represent tailored processes from this last one (e.g., see Figure 31, item number 1).

Figure 49. Tailoring operation *Extend*

DELETE

Purpose: Remove a process element or fragment from the reused base process.

Motivation: In a specific process, an element or fragment of the base process does not need to be executed.

Description: A variant process defines an element B which removes another one (identified by *removedElement*) **or** a fragment of elements (from *fragmentBegin* to *fragmentEnd*) from the reused base process.

Source Element Type(s): Task, Subprocess, or Event.

Parameters: removedElement (BPMN:FlowNode), fragmentBegin (BPMN:FlowNode), fragmentEnd (BPMN:FlowNode).

Pre-conditions:

1) It is not allowed to suppress the start or end event of a process;
2) It is not allowed to suppress gateways directly;
3) It is not allowed to suppress process fragments containing incomplete flow branches.

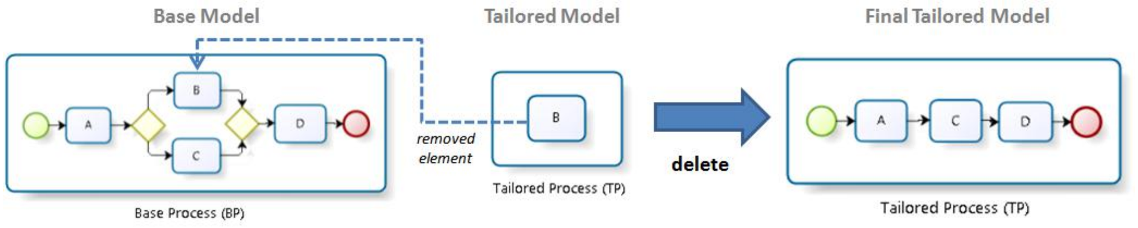
Post-conditions:

1) Elements linked to the removed one that are not related to other elements should also be removed;
2) By deleting an entire flow to or from a gateway, if there is only one flow entering or leaving this

gateway, then the gateway will also be removed;

3) The element immediately preceding the removed one will be connected to the element immediately following it.

Representation:



Related Pattern: Delete Process Fragment (WEBER, REICHERT and RINDERLE-MA, 2008); Suppress Specification Activity (BRANCO et al., 2014).

Example: In a variant software process (SIGA Project), use cases that do not describe a report specification, but a functionality with associated GUI screen, do not require the task *Specify Report* (see Figure 32).

Figure 50. Tailoring operation *Delete*

REPLACE

Purpose: Replace a process element or fragment from the reused base process by another element or fragment.

Motivation: A process element or fragment from the base process is no longer adequate to the specific process, but it can be replaced by another one.

Description: A variant process defines an element X (or process fragment into a subprocess) which replaces another element B (identified by *replacedElement*) **or** a fragment of elements (from *fragmentBegin* to *fragmentEnd*) from the reused base process.
The parameter *additionIsFragment* is relevant only when this operation is defined to a subprocess (source element X). When its value is *true* indicates that the content of the source subprocess of the operation represents a process fragment to be inserted directly in the workflow of the reused base process. Otherwise, the source subprocess itself will be inserted in the target process' workflow.

Source Element Type(s): Task, Subprocess, or Event.

Parameters: replacedElement (BPMN:FlowNode), fragmentBegin (BPMN:FlowNode), fragmentEnd (BPMN:FlowNode), additionIsFragment(boolean).

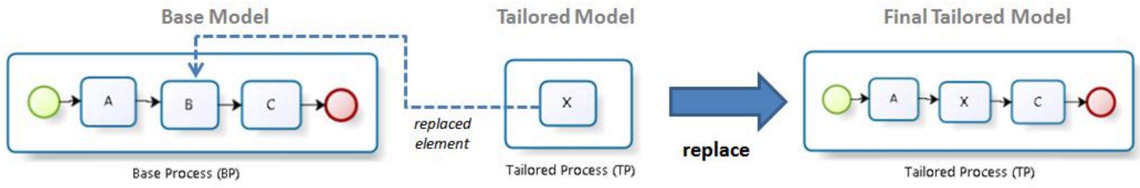
Pre-conditions:

1) Start or end events cannot be replaced;
2) Gateways cannot be replaced directly;
3) Process fragments containing incomplete flow branches cannot be replaced.
4) If source element of the operation is a Subprocess, its content cannot violate any rule in Appendix 1;
5) If source element of the operation is an Event, it must be an intermediate event of specific type (None, Message, Timer, Escalation, Conditional, or Signal).

Post-conditions:

1) Elements linked to the replaced one that are not related to other elements should be removed;
2) Sequence flows connected to the replaced element must be reconfigured to connect in the same way to the substitute element.

Representation:



Related Pattern: Replace Process Fragment (WEBER, REICHERT and RINDERLE-MA, 2008).

Example: The SIGA project's base process could specify only the workflow used to use cases with associated GUI screen (alternative path on the bottom part of Figure 31), i.e., without representing the alternative flow that contains the task *Specify Report*, since these are the most common types of use case of the project. In this case, the base process could be easy adapted to represent the workflow of use cases related to reports by replacing the process fragment between tasks *Design Screen* and *Define Test Cases* by the task *Specify Report.*

Figure 51. Tailoring operation *Replace*

MOVE

Purpose: Move a process element or fragment from its current position in the base process to another position within the variant process.

Motivation: The predefined order to elements in the base process cannot be completely satisfied in the specific process for a given process element or fragment.

Description: A variant process element specifies through this operation that an element D (identified by *movedElement*) **or** a fragment of elements (from *fragmentBegin* to *fragmentEnd*) is moved to a new workflow position immediately after (*newPositionAfter*) or immediately before (*newPositionBefore*) another element C within the same process.

Source Element Type(s): Task.

Parameters: movedElement (BPMN:FlowNode), fragmentBegin (BPMN:FlowNode), fragmentEnd (BPMN:FlowNode), newPositionAfter (BPMN:FlowNode), newPositionBefore (BPMN:FlowNode).

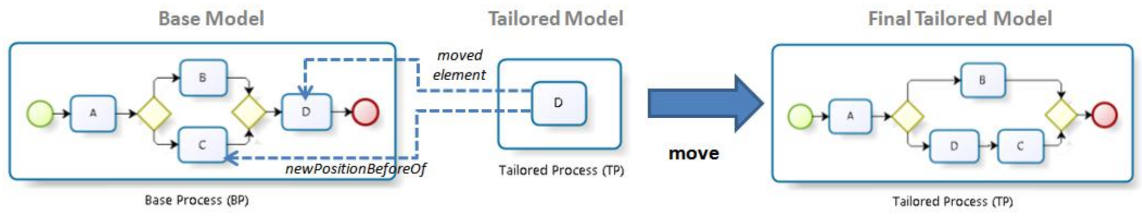
Pre-conditions:

1) If only the parameter *newPositionAfter* is provided by the user, it cannot link to a diverging gateway or end event;
2) If only the parameter *newPositionBefore* is provided by the user, it cannot link to a converging gateway or start event;
3) Start or end events cannot be moved;
4) Gateways cannot be moved singly;
5) Process fragments containing incomplete flow branches cannot be moved.

Post-conditions:

1) Elements linked to the moved one should be moved together;
2) Sequence flows must be reconfigured to connect the predecessor to the successor of the moved element (or fragment) in its old position as well as to reconnect it in its new position.

Representation:



Related Pattern: Move Process Fragment (WEBER, REICHERT and RINDERLE-MA, 2008).

Example: In a particular variant process from the SIGA Project, the task that precedes *Update Database* has been parallelized with other tasks. So, in order to continue being executed soon after its original predecessor, the task *Update Database* needs to be moved (see Figure 32).

Figure 52. Tailoring operation *Move*

PARALLELIZE

Purpose: Parallelize the execution of elements from a process fragment that has been defined as a sequential flow in the base process.

Motivation: The predefined sequential order for a fragment of the base process does not correspond to its real-world execution order, which is concomitant. Typically, tasks assigned to different roles and without dependencies to each other can be performed in parallel.

Description: A variant process element specifies through this operation that a process fragment identified by the parameters *fragmentBegin* and *fragmentEnd* has its execution parallelized in the reused base process. To this end, the operation includes a parallel gateway before the target fragment's elements to diverge the process flow and another parallel gateway after the fragment's elements to converge again the process flow.

Source Element Type(s): Task.

Parameters: fragmentBegin (BPMN:FlowNode), fragmentEnd (BPMN:FlowNode).

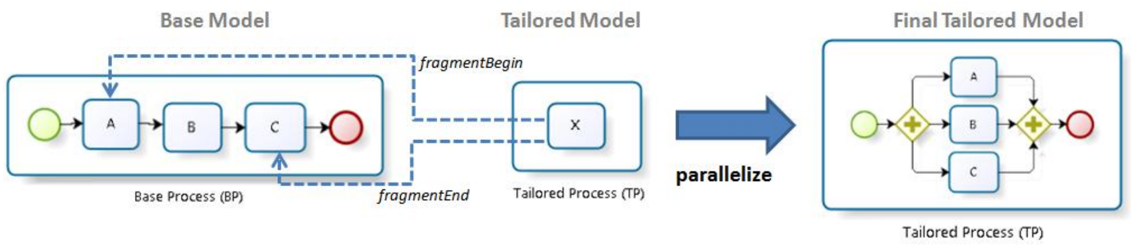
Pre-conditions:

1) Parameters *fragmentBegin* and *fragmentEnd* must point to succeeding different elements;
2) The target process fragment cannot include start or end events;
3) The target process fragment cannot include gateways.

Post-conditions:

1) A BPMN parallel gateway is included before the target fragment's elements to diverge the process flow and another BPMN parallel gateway is included after the target fragment's elements to converge again the process flow;
2) Sequence Flows are inserted and adjusted to connect target fragment's elements to the parallel gateways included by this operation.

Representation:



Related Pattern: Parallelize Activities (WEBER, REICHERT and RINDERLE-MA, 2008).

Example: In a particular variant process from the SIGA Project, the tasks *Review Use Case Specification*, *Define Test Cases*, and *Elaborate Physical Model* have been parallelized to represent their real-world execution order (see Figure 32).

Figure 53. Tailoring operation *Parallelize*

SERIAL INSERT

Purpose: Insert a new process element or fragment between two directly succeeding elements of the reused base process.

Motivation: In a more specific process, a task has to be performed which has not been modeled in the more general process, i.e. the base process.

Description: A variant process defines a process element or fragment X which should be inserted in the reused base process workflow after the element indicated by the parameter *after* **or** before the element indicated by the parameter *before*.
The parameter *additionIsFragment* is relevant only when this operation is defined to a subprocess (source element X). When its value is *true* indicates that the content of the source subprocess of the operation represents a process fragment to be inserted directly in the workflow of the reused base process. Otherwise, the source subprocess itself will be inserted in the target process' workflow.

Source Element Type(s): Task, Subprocess, or Event.

Parameters: after(BPMN:FlowNode), before(BPMN:FlowNode), additionIsFragment(boolean = false).

Pre-conditions:

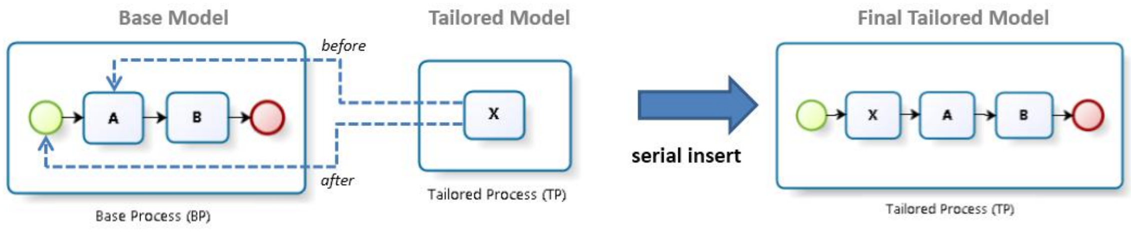
1. If the parameter *aftef* is provided by the user, it cannot link to a diverging gateway or end event;
2. If the parameter *before* is provided by the user, it cannot link to a converging gateway or start event;
3. Gateways cannot be inserted singly;
4. Process fragments containing incomplete flow branches cannot be inserted;

5. If source element of the operation is a Subprocess, its content cannot violate any rule in Appendix 1;
6. If source element of the operation is an Event, it must be an intermediate event of specific type (None, Message, Timer, Escalation, Conditional, or Signal).

Post-conditions:

1. Sequence flows must be reconfigured to connect to the element or fragment inserted.

Representation:



Related Pattern: Insert Process Fragment (WEBER, REICHERT and RINDERLE-MA, 2008).

Example: In a particular variant process from the SIGA Project, the task *Elaborate Mapping of Use Case Links*, which has not been modeled in the base process, needed to be performed in executed processes between the directly succeeding tasks *Update General Class Diagram* and *Review Use Case Specification* (see Figure 33).

Figure 54. Tailoring operation *Serial Insert*

CONDITIONAL INSERT

Purpose: Insert in the reused base process a conditional process element or fragment that is executed only when a given condition is true.

Motivation: In a specific process, a task that has not been modeled in the base process needs to be performed when a given condition (situation) is met.

Description: A variant process defines a process element or fragment X which is inserted in the reused base process workflow as an alternative to the single element indicated by the parameter *alternativeToElement* **or** to the process fragment contained between the elements indicated by *after* and *before*. When these parameters point to directly succeeding elements means the inserted task or fragment is optional, i.e., its execution can be skipped. The parameter *condition* must contain a unique expression determining when the inserted element is executed and *isExclusiveCondition* indicates if the *true* evaluation of the provided condition excludes the evaluation of other conditional alternatives. The parameter *additionIsFragment* is relevant only when this operation is defined to a subprocess (source element X). When its value is *true* indicates that the content of the source subprocess of the operation represents a process fragment to be inserted directly in the workflow of the reused base process. Otherwise, the source subprocess itself will be inserted in the target process' workflow. The parameter *inLoop* indicates if the inserted element is into a loop.

Source Element Type(s): Task, Subprocess, or Event.

Parameters: alternativeToElement(BPMN:FlowNode), after(BPMN:FlowNode), before(BPMN:FlowNode), condition(String), isExclusiveCondition(boolean = true), additionIsFragment(boolean = false), inLoop (boolean = false).

Pre-conditions:

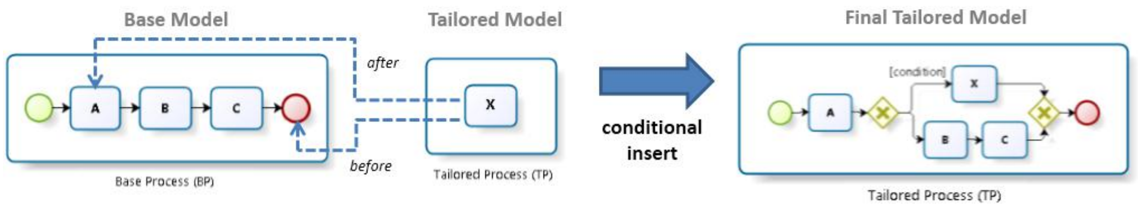
1. The parameter *alternativeToElement* must *not* point to a start or end event or a gateway;
2. The base process fragment between the parameters *after* and *before* cannot contain incomplete flow branches;
3. Parameters *after* and *before* cannot both link to non-conditional gateways (i.e., different of exclusive or inclusive ones);
4. Gateways cannot be inserted singly;
5. Process fragments containing incomplete flow branches cannot be inserted;
6. The parameter *condition* must always be provided;
7. If source element of the operation is a Subprocess, its content cannot violate any rule in Appendix 1;

8. If source element of the operation is an Event, it must be an intermediate event of specific type (None, Message, Timer, Escalation, Conditional, or Signal).

Post-conditions:

1. If the parameter *isExclusiveCondition* is true, a BPMN Exclusive Gateway is included soon after the element pointed by the parameter *after* (or soon before the element pointed by *alternativeToElement*) to diverge the process flow and another exclusive gateway is included soon before the element pointed by the parameter *before* (or soon after the element pointed by *alternativeToElement*) to converge again the process flow;
2. If the parameter *isExclusiveCondition* is false, a BPMN Inclusive Gateway is included soon after the element pointed by the parameter *after* (or soon before the element pointed by *alternativeToElement*) to diverge the process flow and another inclusive gateway is included soon before the element pointed by the parameter *before* (or soon after the element pointed by *alternativeToElement*) to converge again the process flow;
3. Sequence flows are adjusted to connect to the inserted elements.
4. The sequence flow outgoing from the added diverging Gateway (split) and leading to the original base process fragment must become the default flow from the Gateway.

Representation:



Related Pattern: Insert Process Fragment (WEBER, REICHERT and RINDERLE-MA, 2008).

Example: In a variant process from the SIGA Project, a system screen design needs to be reviewed only if it has not been reviewed yet in a previous stage of the process (i.e., the review task is conditional and it has not been modeled in the project's base process). In another situation, when are found problems in a use case specification, a new task to correct its inconsistencies needs to be performed and soon after the process execution flow must return to before the task *Design Screen* in order to alter the artifacts produced previously (see Figure 33).

Figure 55. Tailoring operation *Conditional Insert*

EVENT-BASED INSERT

Purpose: Insert in the reused base process a conditional process fragment that is executed only when a given event (of message, signal, time, or condition) occurs before other alternative events. The occurrence of this event cancels the others, i.e., event-based alternatives are mutually exclusive.

Motivation: In a specific process, a process fragment that has not been modeled in the base process needs to be performed when a given event occurs before other alternative events. The event, usually the receipt of a message or time expiration, determines the execution of this fragment instead of other ones.

Description: A variant process defines a process element or fragment X which is inserted in the reused base process workflow as an event-based alternative to the target process fragment contained between the elements indicated by *after* and *before*. The target fragment MUST have as first element an intermediate catch event of type *Message*, *Signal*, *Timer*, or *Conditional*. The parameter *eventType* identifies the type of event that makes the new process fragment be performed (its default value is *Message*) whereas the parameter *eventName* must contain a unique identifier name for such an event.

Source Element Type(s): Task or Subprocess.

Parameters: after(BPMN:FlowNode), before(BPMN:FlowNode), eventType(InsertEventType = Message {Message | Signal | Timer | Conditional}), eventName(String).

Pre-conditions:

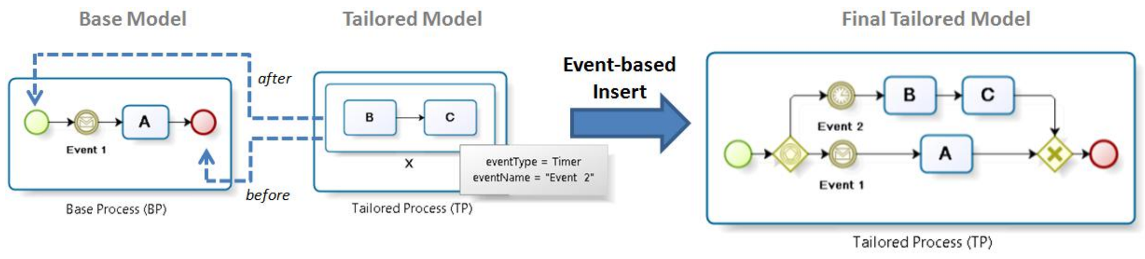
1. Parameters *after* and *before* must *not* point to directly succeeding elements;
2. The target process fragment between the parameters *after* and *before* must have as first element an intermediate catch event of type *Message*, *Signal*, *Timer*, or *Conditional*;
3. The target process fragment cannot contain incomplete flow branches;

4. Parameters *after* and *before* cannot both point to different gateways of event-based and exclusive ones, respectively;
5. Gateways cannot be inserted singly;
6. Process fragments containing incomplete flow branches cannot be inserted.
7. If source element of the operation is a Subprocess, its content cannot violate any rule in Appendix 1.

Post-conditions:

1. If parameter *after* points to a Event-Based Gateway (event-based XOR-Split) and parameter *before* points to a converging Exclusive Gateway (XOR-Join), then new outgoing and incoming sequence flows are added to these gateways, respectively, connecting to the inserted process fragment;
2. If post-condition 1 is false, a BPMN Event-Based Gateway (event-based XOR-Split) is included soon after the element pointed by the parameter *after* to diverge the process flow and a BPMN Exclusive Gateway (XOR-Join) is included soon before the element pointed by the parameter *before* to converge again the process flow;
3. An intermediate event of specific type informed by the parameter *eventType* and name defined by the parameter *eventName* is added immediately before the inserted process fragment, since it will be responsible by the activation of this fragment, and immediately after the Event-Based Gateway.
4. Sequence flows are added and adjusted to connect the new elements to the tailored process workflow.

Representation:



Related Pattern: --

Example: In a loan offer process (DUMAS and PFAHL, 2016), after offering loan for a customer the process waits for a response. It will perform one given task if the customer responds "offer accepted" and another task if the customer responds "offer refused". The identity (name) of the customer's response message determines which task is performed. That is, "offer accepted" and "offer refused" are different messages because they have different identifiers. However, besides these options it is necessary to consider that the customer cannot respond. Then, the task *Cancel Loan Offer* is inserted in the process to be performed when the response date expired (see Figure 40).

Figure 56. Tailoring operation *Event-Based Insert*

ENCAPSULATE

Purpose: Encapsulate a process fragment with related activities into a separate subprocess.

Motivation: The complexity of the variant process increased significantly regarding its base process, then in order to make it simpler and easy to understand and maintain, a part of this process is encapsulated in a separate subprocess.

Description: A variant process defines a subprocess S that through this operation will encapsulate the process fragment identified by the parameters *fragmentBegin* and *fragmentEnd* and take its place in the process workflow.

Source Element Type: Subprocess.

Parameters: fragmentBegin (BPMN:FlowNode), fragmentEnd (BPMN:FlowNode).

Pre-conditions:

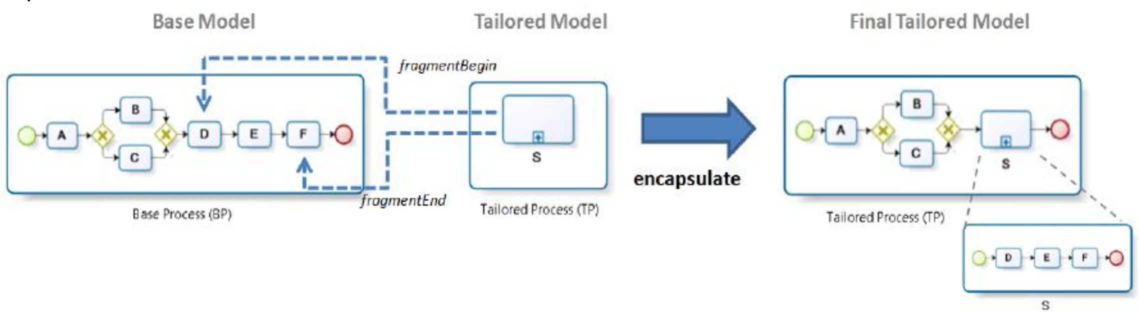
1. Parameters *fragmentBegin* and *fragmentEnd* cannot point to the same element;
2. The target process fragment cannot include start or end events;
3. The target process fragment cannot contain incomplete flow branches;

Post-conditions:

1. Start and End events are added in the beginning and end of the encapsulated process fragment,

respectively;
2. Sequence flows connected to the target process fragment must be reconfigured to connect in the same way to the substitute subprocess.

Representation:



Related Pattern: EXTRACT Process Fragment to Sub-Process (WEBER, REICHERT and RINDERLE-MA, 2008)

Example: When adapting a Requirements Engineering process to incorporate activities of MDD (Model-Driven Development), a process has become too large (LONIEWSKI, ARMESTO and INSFRAN, 2011). Thus, activities from the original process have been encapsuled into a subprocess named *Capture and Analyze Requirements* (see Figure 35 and Figure 36).

Figure 57. Tailoring operation *Encapsulate*

SPLIT

Purpose: Split a single task from the base process to a process fragment or subprocess that details its procedure in the variant process.

Motivation: A task from the base process is too generic to the level of abstraction of the variant process and therefore needs of refinement.

Description: A variant process defines a subprocess X which details steps to perform a generic task identified by the parameter *targetElement* from the base process. If the parameter *splitIntoSubprocess* is true, then task B is replaced by the subprocess X into the variant process. Otherwise, the workflow defined by the subprocess X is directly embedded into the variant process as a process fragment (by automatically removing any start or end event).

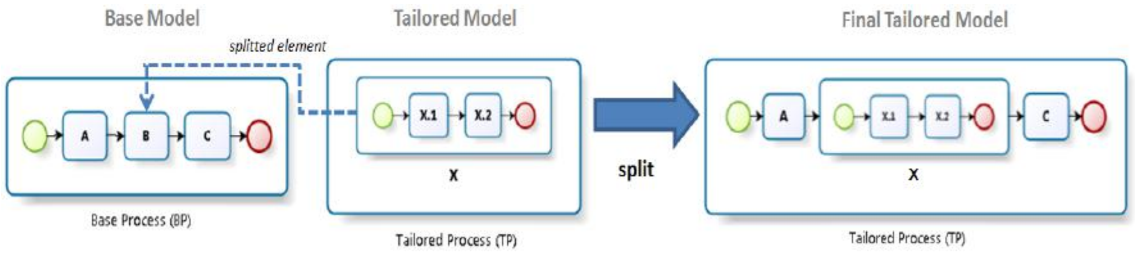
Source Element Type: Subprocess.

Parameters: targetElement(BPMN:Task), splitIntoSubprocess(boolean = true).

Pre-conditions:
1. The subprocess that defines this operation (from the variant process) must contain a valid process workflow or fragment (according to rules in Appendix 1).

Post-conditions:
1. If parameter *splitIntoSubprocess* is false, then start and end events of the workflow defined by the source subprocess of the operation must be removed before embedding it into the variant process;
2. Sequence flows connected to the splitted element must be reconfigured to connect in the same way to the substitute subprocess or fragment.
3. If parameter *splitIntoSubprocess* is true, then elements linked to the splitted task (e.g., boundary events) must be re-linked to the substitute subprocess.

Representation:



Related Pattern: Split Task into Block; Split Workflow (BRANCO et al., 2014).

Example: **(1)** In a variant software process (SIGA Project), the task *Design Screen* of the base process has been splitted into a subprocess called *Design and Validate User Interfaces* that details its steps (see Figure 32). In this case, the default value of the parameter *splitIntoSubprocess* was taken on (i.e., true).

(2) In a technical-level claim handling process, the task *Settle Claim* of the business-level base process has been detailed by a process fragment (composed by tasks *Create Response Letter* and *Send Response*) directly embedded into the workflow of the technical-level process, i.e., not enclosed in a subprocess' scope (see Figure 37 and Figure 38). In this case, the parameter *splitIntoSubprocess* was set to false.

Figure 58. Tailoring operation *Split*

MERGE

Purpose: Merge two or more directly succeeding tasks into a single particular task.

Motivation: Business tasks from the base process can be subsumed in a technical-level specification (tailored process). Typical examples are: (1) Combine several business tasks into a single service task (the service provided would be coarser than the business steps described); (2) Combine human tasks into a single human task in which the separate steps of the human action are described elsewhere (e.g., a screenflow).

Description: A variant process defines a task (DE) that represents the combination (merge) of tasks contained in the base process fragment identified by the parameters *fragmentBegin* and *fragmentEnd* and replaces these tasks in the variant process workflow.

Source Element Type: Task.

Parameters: fragmentBegin (BPMN:Task), fragmentEnd (BPMN:Task).

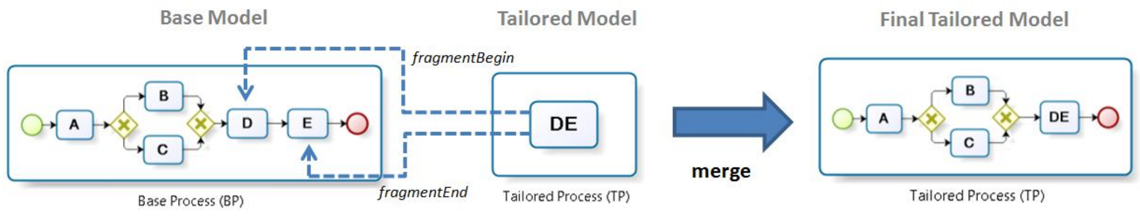
Pre-conditions:

1. Parameters *fragmentBegin* and *fragmentEnd* must point to succeeding different tasks;
2. The target process fragment must contain only task nodes;

Post-conditions:

1. Sequence flows connected to the target process fragment must be reconfigured to connect in the same way to the substitute task.
2. If a task from the target fragment has elements linked to it (e.g., boundary events), then these elements must be re-linked to the substitute task.

Representation:



Related Pattern: SUPPRESS Specification Activity (BRANCO et al., 2014).

Example: In a technical-level claim handling process, tasks *Get Personal Details* and *Get Insurance Details* have been merged into a single (human) task named *Get Request* Details (see Figure 37 and Figure 38). In this context, such a practice means the separate steps of the human action are described elsewhere (e.g., a screenflow).

Figure 59. Tailoring operation *Merge*

RENAME

Purpose: Change the name of a process element.

Motivation: The name of an element from the base process is not adequate or representative enough to the context of the tailored process.

Description: A variant process defines an element Z of same type than an element Y from the base process.

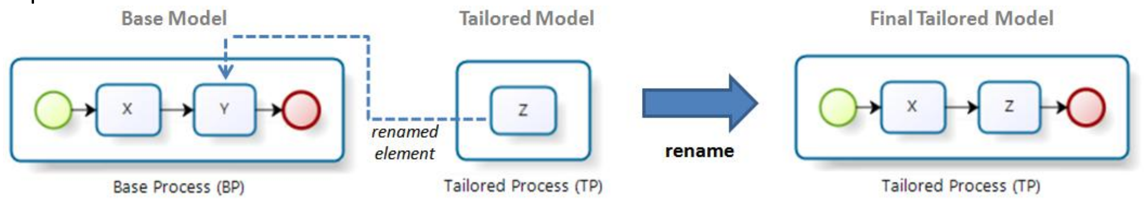
Through this operation, the name of the variant element Z replaces the name of the base element Y.

Source Element Type(s): Task, Subprocess, Event, Gateway, or Sequence Flow.

Parameters: renamedElement(BPMN:FlowElement).

Pre-conditions: Variant and base elements must be of same type.

Representation:



Related Pattern: Change Activity Name (BRANCO et al., 2014); Update Condition (WEBER, REICHERT and RINDERLE-MA, 2008).

Example: In a claim handling process, the task *Validate Claim* has been renamed to *Validate Claim on Decision Server* to better reflect some technical aspects of this task (see Figure 37 and Figure 38).

Figure 60. Tailoring operation *Rename*

ADD EXCEPTION HANDLER

Purpose: Add an exception handler (task or subprocess) to deal with a given type of exception event that can occur in the context of one or more task(s) or subprocess(es).

Motivation: According to BRANCO *et al.* (2014), technical exception handlers are not expected to be represented in a business-level model, because they implement nonfunctional requirements. Therefore, they are generally added when refining these models to technical-level models.

Description: A variant process defines a task or subprocess that will be responsible by dealing with a technical exception of type defined by the parameter *exceptionType* and name defined by the parameter *exceptionName* that can occur while the element pointed by *targetElement* is executing. The parameter *interrupting* determines if the normal flow of the process will be or not interrupted when this exception occurs.

Source Element Type: Task or Subprocess.

Parameters: targetElement (BPMN:Activity[*]), exceptionType(ExceptionType = Error {Message | Timer | Escalation | Error | Signal | Conditional}), exceptionName(String), interrupting(boolean = true).

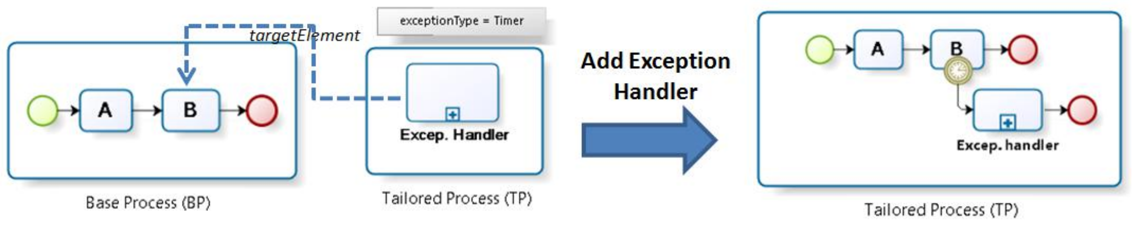
Pre-conditions:

1. If parameter *exceptionType* is of type Error, then the parameter *interrupting* must be true.
2. If source element of the operation is a Subprocess, its content cannot violate any rule in Appendix 1.

Post-conditions:

1. Add an event of the specific type indicated by the parameter *expectionType* to the boundary of the element(s) pointed by *targetElement*.
2. Add a Sequence Flow outgoing from each boundary event added by rule 1 and incoming to the source element of the operation (exception handler).
3. Add an (plain) end event and a sequence flow outgoing from the source element of the operation (exception handler) and incoming to the added end event.

Representation:



Related Pattern: Add Boundary Event; Add Technical Exception Flow (BRANCO *et al.*, 2014).

Example: In a technical-level claim handling process, an exception handler subprocess named *Manual Handling* has been added for tasks *Reject Claim* and *Create Claim Document* (see Figure 37 and Figure 38). In this case, the default value for the parameter *exceptionType* was taken on (i.e., Error). Therefore, the subprocess *Manual Handling* must deal with error exceptions that can occur while any of these tasks is executing.

Figure 61. Tailoring operation *Add Exception Handler*

ADD EXCEPTION FLOW

Purpose: Add an exception flow to deal with a given type of event that is triggered and handled by activities from the base process.

Motivation: Addition of a technical exception flow is required when a given exception event must be catched and handled and the source activity of the exception as well as its handler are already modeled in the base process.

Description: A variant process defines an event of specific type that will be responsible by catching an exception that can occur in the context of one or more activities pointed by the parameter triggeringElement. This exception will be handled by the activity pointed by the parameter handlingElement.

Source Element Type: Event.

Parameters: triggeringElement (BPMN:Activity[*]), handlingElement(BPMN:Activity).

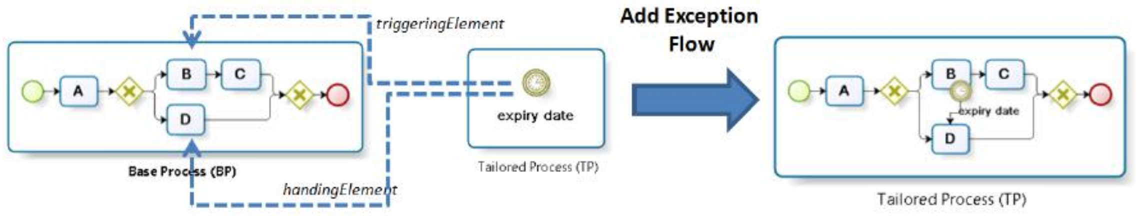
Pre-conditions:

1. The source event of the operation must be of type Message, Timer, Escalation, Error, Signal, or Conditional;
2. If the source event of the operation is of type Error, then it must be an interrupting event.

Post-conditions:

1. Add the source event of the operation to the boundary of the element(s) pointed by the parameter triggeringElement.
2. Add a Sequence Flow outgoing from each boundary event added by rule 1 and incoming to the element pointed by the parameter handlingElement (exception handler).

Representation:



Related Pattern: Add boundary event; Add technical exception flow (BRANCO et al., 2014).

Example: In an Automated Teller Machine (ATM) process, the task Cancel Transaction modeled in the business-level base process is considered an exception handler in the technical-level process. Thus, this task is transformed in a subprocess and afterwards it becomes target of exception flows, added in the technical process, that take to it. The complete model of this example can be seen in Figure 45 and Figure 46.

Figure 62. Tailoring operation *Add Exception Flow*

Appendix 3: XML Schema-based BPMN*t* Extension

Table 17 bellow shows the structure of the High-Level BPMN*t* extension defined in the XML Schema language. Elements *xsd:complexType* group the BPMN*t* extension concepts representing tailoring operations, such as the BPMN extension meta-class *ExtensionDefinition*. Above of the definition of each extension concept type, an element of this specific type is also created for containing its instances in a BPMN model.

Table 17. *BPMNt2.xsd* – BPMN*t* Extension Definition in XML Schema

```
<?xml version="1.0" encoding="UTF-8" standalone="no"?>
<xsd:schema xmlns="http://www.extensions.com/bpmnt_HL"
xmlns:bpmn="http://www.omg.org/spec/BPMN/20100524/MODEL"
xmlns:xsd="http://www.w3.org/2001/XMLSchema" attributeFormDefault="unqualified"
elementFormDefault="qualified" targetNamespace="http://www.extensions.com/bpmnt_HL">

<xsd:import namespace="http://www.omg.org/spec/BPMN/20100524/MODEL"
schemaLocation="BPMN20.xsd"/>

<xsd:element name="extendOperation" type="Extend"/>
<xsd:complexType name="Extend">
<xsd:sequence>
        <xsd:element maxOccurs="1" minOccurs="1" name="extendedProcess"
type="bpmn:tBaseElement"/>
        <xsd:element maxOccurs="1" minOccurs="1" name="applicationOrder" type="OrderType"/>
</xsd:sequence>
</xsd:complexType>

<xsd:simpleType name="OrderType">
        <xsd:restriction base="xsd:string">
                <xsd:enumeration value="Free"/>
                <xsd:enumeration value="FirstAddition"/>
        </xsd:restriction>
</xsd:simpleType>

<xsd:complexType abstract="true" name="BasicOperation">
<xsd:sequence>
        <xsd:element name="name" type="xsd:string"/>
</xsd:sequence>
</xsd:complexType>

<xsd:element name="suppressOperation" type="Suppress"/>
<xsd:complexType name="Suppress">
<xsd:complexContent>
<xsd:extension base="BasicOperation">
<xsd:sequence>
        <xsd:element maxOccurs="1" minOccurs="1" name="suppressedElement"
type="bpmn:tBaseElement"/>
</xsd:sequence>
</xsd:extension>
</xsd:complexContent>
</xsd:complexType>

<xsd:element name="contributeOperation" type="Contribute"/>
<xsd:complexType name="Contribute">
<xsd:complexContent>
<xsd:extension base="BasicOperation">
        <xsd:sequence>
        <xsd:element maxOccurs="1" minOccurs="1" name="targetProcess" type="bpmn:tBaseElement"/>
        <xsd:element maxOccurs="1" minOccurs="0" name="newElement" type="bpmn:tBaseElement"/>
        <xsd:element maxOccurs="1" minOccurs="0" name="newElementRef" type="bpmn:tBaseElement"/>
        </xsd:sequence>
</xsd:extension>
</xsd:complexContent>
</xsd:complexType>

<xsd:element name="modifyOperation" type="Modify"/>
<xsd:complexType name="Modify">
<xsd:complexContent>
<xsd:extension base="BasicOperation">
<xsd:sequence>
        <xsd:element name="property" type="xsd:string"/>
        <xsd:element name="value" type="xsd:string"/>
        <xsd:element maxOccurs="1" minOccurs="1" name="modifiedElement"
type="bpmn:tBaseElement"/>
        <xsd:element maxOccurs="1" minOccurs="0" name="valueRef" type="bpmn:tBaseElement"/>
</xsd:sequence>
</xsd:extension>
</xsd:complexContent>
</xsd:complexType>


<xsd:complexType abstract="true" name="HighLevelOperation">
<xsd:sequence>
        <xsd:element name="name" type="xsd:string"/>
        <xsd:element name="motivation" type="xsd:string"/>
        <xsd:element maxOccurs="unbounded" minOccurs="1" name="basicOperations"
type="BasicOperation"/>
</xsd:sequence>
</xsd:complexType>

<xsd:element name="deleteOperation" type="Delete"/>
<xsd:complexType name="Delete">
<xsd:complexContent>
<xsd:extension base="HighLevelOperation">
<xsd:sequence>
        <xsd:element maxOccurs="1" minOccurs="0" name="removedElement" type="bpmn:tFlowNode"/>
        <xsd:element maxOccurs="1" minOccurs="0" name="fragmentBegin" type="bpmn:tFlowNode"/>
        <xsd:element maxOccurs="1" minOccurs="0" name="fragmentEnd" type="bpmn:tFlowNode"/>
</xsd:sequence>
</xsd:extension>
</xsd:complexContent>
</xsd:complexType>

<xsd:element name="replaceOperation" type="Replace"/>
<xsd:complexType name="Replace">
<xsd:complexContent>
<xsd:extension base="HighLevelOperation">
<xsd:sequence>
        <xsd:element maxOccurs="1" minOccurs="0" name="replacedElement" type="bpmn:tFlowNode"/>
        <xsd:element maxOccurs="1" minOccurs="0" name="fragmentBegin" type="bpmn:tFlowNode"/>
        <xsd:element maxOccurs="1" minOccurs="0" name="fragmentEnd" type="bpmn:tFlowNode"/>
        <xsd:element name="additionIsFragment" type="xsd:boolean"/>
</xsd:sequence>
</xsd:extension>
</xsd:complexContent>
</xsd:complexType>

<xsd:element name="renameOperation" type="Rename"/>
<xsd:complexType name="Rename">
<xsd:complexContent>
<xsd:extension base="HighLevelOperation">
<xsd:sequence>
        <xsd:element maxOccurs="1" minOccurs="1" name="renamedElement"
type="bpmn:tFlowElement"/>
</xsd:sequence>
</xsd:extension>
</xsd:complexContent>
</xsd:complexType>
```

```
<xsd:element name="moveOperation" type="Move"/>
<xsd:complexType name="Move">
<xsd:complexContent>
<xsd:extension base="HighLevelOperation">
<xsd:sequence>
        <xsd:element maxOccurs="1" minOccurs="0" name="movedElement" type="bpmn:tFlowNode"/>
        <xsd:element maxOccurs="1" minOccurs="0" name="fragmentBegin" type="bpmn:tFlowNode"/>
        <xsd:element maxOccurs="1" minOccurs="0" name="fragmentEnd" type="bpmn:tFlowNode"/>
        <xsd:element maxOccurs="1" minOccurs="0" name="newPositionAfter" type="bpmn:tFlowNode"/>
        <xsd:element maxOccurs="1" minOccurs="0" name="newPositionBefore"
type="bpmn:tFlowNode"/>
</xsd:sequence>
</xsd:extension>
</xsd:complexContent>
</xsd:complexType>

<xsd:element name="parallelizeOperation" type="Parallelize"/>
<xsd:complexType name="Parallelize">
<xsd:complexContent>
<xsd:extension base="HighLevelOperation">
<xsd:sequence>
        <xsd:element maxOccurs="1" minOccurs="1" name="fragmentBegin" type="bpmn:tFlowNode"/>
        <xsd:element maxOccurs="1" minOccurs="1" name="fragmentEnd" type="bpmn:tFlowNode"/>
</xsd:sequence>
</xsd:extension>
</xsd:complexContent>
</xsd:complexType>

<xsd:element name="splitOperation" type="Split"/>
<xsd:complexType name="Split">
<xsd:complexContent>
<xsd:extension base="HighLevelOperation">
<xsd:sequence>
        <xsd:element maxOccurs="1" minOccurs="1" name="targetElement" type="bpmn:tTask"/>
        <xsd:element name="splitIntoSubprocess" type="xsd:boolean"/>
</xsd:sequence>
</xsd:extension>
</xsd:complexContent>
</xsd:complexType>

<xsd:element name="mergeOperation" type="Merge"/>
<xsd:complexType name="Merge">
<xsd:complexContent>
<xsd:extension base="HighLevelOperation">
<xsd:sequence>
        <xsd:element maxOccurs="1" minOccurs="1" name="fragmentBegin" type="bpmn:tTask"/>
        <xsd:element maxOccurs="1" minOccurs="1" name="fragmentEnd" type="bpmn:tTask"/>
</xsd:sequence>
</xsd:extension>
</xsd:complexContent>
</xsd:complexType>

<xsd:element name="encapsulateOperation" type="Encapsulate"/>
<xsd:complexType name="Encapsulate">
<xsd:complexContent>
<xsd:extension base="HighLevelOperation">
<xsd:sequence>
        <xsd:element maxOccurs="1" minOccurs="1" name="fragmentBegin" type="bpmn:tFlowNode"/>
        <xsd:element maxOccurs="1" minOccurs="1" name="fragmentEnd" type="bpmn:tFlowNode"/>
</xsd:sequence>
</xsd:extension>
</xsd:complexContent>
</xsd:complexType>

<xsd:element name="serialInsertOperation" type="SerialInsert"/>
<xsd:complexType name="SerialInsert">
<xsd:complexContent>
<xsd:extension base="HighLevelOperation">
<xsd:sequence>
        <xsd:element maxOccurs="1" minOccurs="0" name="after" type="bpmn:tFlowNode"/>
        <xsd:element maxOccurs="1" minOccurs="0" name="before" type="bpmn:tFlowNode"/>
        <xsd:element name="additionIsFragment" type="xsd:boolean"/>
```

```
</xsd:sequence>
</xsd:extension>
</xsd:complexContent>
</xsd:complexType>

<xsd:element name="conditionalInsertOperation" type="ConditionalInsert"/>
<xsd:complexType name="ConditionalInsert">
<xsd:complexContent>
<xsd:extension base="HighLevelOperation">
<xsd:sequence>
        <xsd:element maxOccurs="1" minOccurs="0" name="alternativeToElement"
type="bpmn:tFlowNode"/>
        <xsd:element maxOccurs="1" minOccurs="0" name="after" type="bpmn:tFlowNode"/>
        <xsd:element maxOccurs="1" minOccurs="0" name="before" type="bpmn:tFlowNode"/>
        <xsd:element name="condition" type="xsd:string"/>
        <xsd:element name="isExclusiveCondition" type="xsd:boolean"/>
        <xsd:element name="additionIsFragment" type="xsd:boolean"/>
        <xsd:element name="inLoop" type="xsd:boolean"/>
</xsd:sequence>
</xsd:extension>
</xsd:complexContent>
</xsd:complexType>

<xsd:element name="parallelInsertOperation" type="ParallelInsert"/>
<xsd:complexType name="ParallelInsert">
<xsd:complexContent>
<xsd:extension base="HighLevelOperation">
<xsd:sequence>
        <xsd:element maxOccurs="1" minOccurs="0" name="parallelToElement"
type="bpmn:tFlowNode"/>
        <xsd:element maxOccurs="1" minOccurs="0" name="after" type="bpmn:tFlowNode"/>
        <xsd:element maxOccurs="1" minOccurs="0" name="before" type="bpmn:tFlowNode"/>
        <xsd:element name="additionIsFragment" type="xsd:boolean"/>
</xsd:sequence>
</xsd:extension>
</xsd:complexContent>
</xsd:complexType>

<xsd:element name="eventBasedInsertOperation" type="EventBasedInsert"/>
<xsd:complexType name="EventBasedInsert">
<xsd:complexContent>
<xsd:extension base="HighLevelOperation">
<xsd:sequence>
        <xsd:element maxOccurs="1" minOccurs="1" name="after" type="bpmn:tFlowNode"/>
        <xsd:element maxOccurs="1" minOccurs="1" name="before" type="bpmn:tFlowNode"/>
        <xsd:element maxOccurs="1" minOccurs="1" name="eventType" type="InsertEventType"/>
        <xsd:element name="eventName" type="xsd:string"/>
</xsd:sequence>
</xsd:extension>
</xsd:complexContent>
</xsd:complexType>

<xsd:simpleType name="InsertEventType">
        <xsd:restriction base="xsd:string">
                <xsd:enumeration value="Message"/>
                <xsd:enumeration value="Signal"/>
                <xsd:enumeration value="Timer"/>
                <xsd:enumeration value="Conditional"/>
        </xsd:restriction>
</xsd:simpleType>

<xsd:element name="specializeOperation" type="Specialize"/>
<xsd:complexType name="Specialize">
<xsd:complexContent>
<xsd:extension base="HighLevelOperation">
<xsd:sequence>
        <xsd:element maxOccurs="1" minOccurs="1" name="targetElement" type="bpmn:tFlowNode"/>
</xsd:sequence>
</xsd:extension>
</xsd:complexContent>
</xsd:complexType>
```

```
<xsd:element name="addExceptionHandlerOperation" type="AddExceptionHandler"/>
<xsd:complexType name="AddExceptionHandler">
<xsd:complexContent>
<xsd:extension base="HighLevelOperation">
<xsd:sequence>
        <xsd:element maxOccurs="unbounded" minOccurs="1" name="targetElement"
type="bpmn:tActivity"/>
        <xsd:element maxOccurs="1" minOccurs="1" name="exceptionType" type="ExceptionType"/>
        <xsd:element name="exceptiontName" type="xsd:string"/>
        <xsd:element name="interrupting" type="xsd:boolean"/>
</xsd:sequence>
</xsd:extension>
</xsd:complexContent>
</xsd:complexType>

<xsd:simpleType name="ExceptionType">
        <xsd:restriction base="xsd:string">
                <xsd:enumeration value="Message"/>
                <xsd:enumeration value="Timer"/>
                <xsd:enumeration value="Escalation"/>
                <xsd:enumeration value="Error"/>
                <xsd:enumeration value="Signal"/>
                <xsd:enumeration value="Conditional"/>
        </xsd:restriction>
</xsd:simpleType>

<xsd:element name="addExceptionFlowOperation" type="AddExceptionFlow"/>
<xsd:complexType name="AddExceptionFlow">
<xsd:complexContent>
<xsd:extension base="HighLevelOperation">
<xsd:sequence>
        <xsd:element maxOccurs="unbounded" minOccurs="1" name="triggeringElement"
type="bpmn:tActivity"/>
        <xsd:element maxOccurs="1" minOccurs="1" name="handlingElement" type="bpmn:tActivity"/>
</xsd:sequence>
</xsd:extension>
</xsd:complexContent>
</xsd:complexType>

</xsd:schema>
```

Appendix 4: BPMN Models used in the Evaluation

Bellow, we present all BPMN models used in our evaluation study based on the software development process of the SIGA Project. We present the base process model from the Specification and Design phase of the project (Figure 63) and six of its execution variants (Figure 64 to Figure 69).

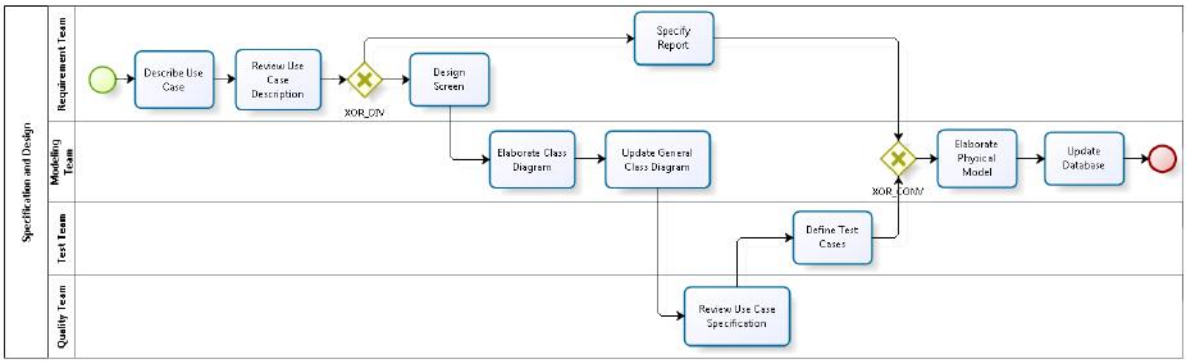


Figure 63: Base process model of the SIGA Project

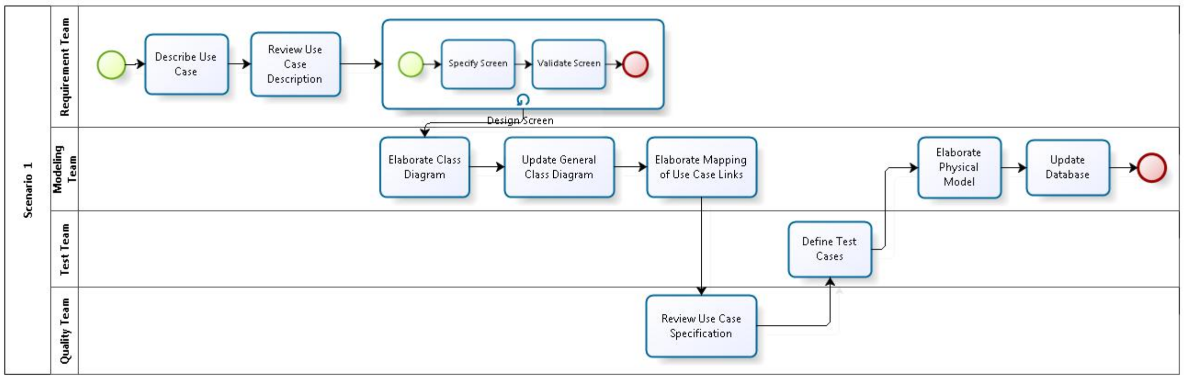


Figure 64: Scenario 1 – Variant process model

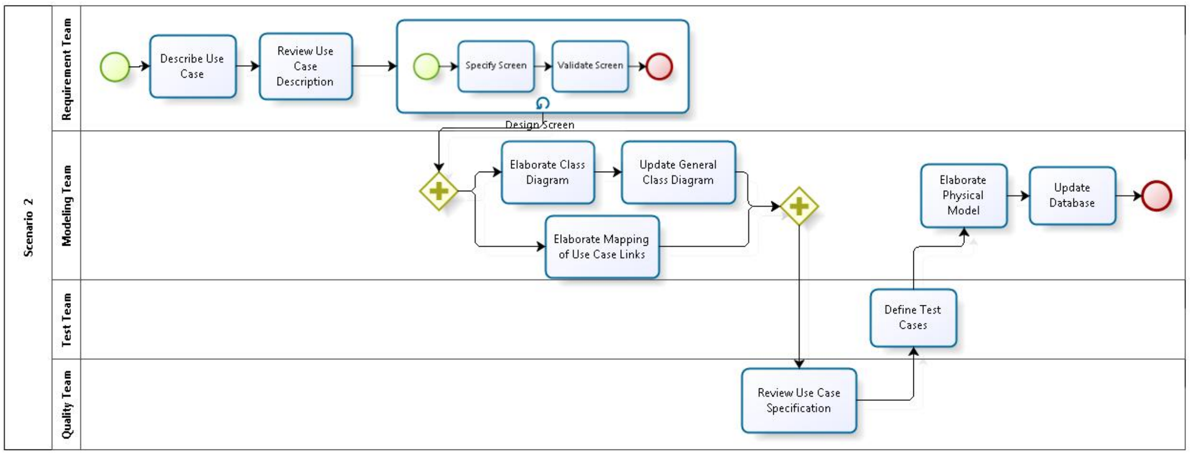


Figure 65: Scenario 2 – Variant process model

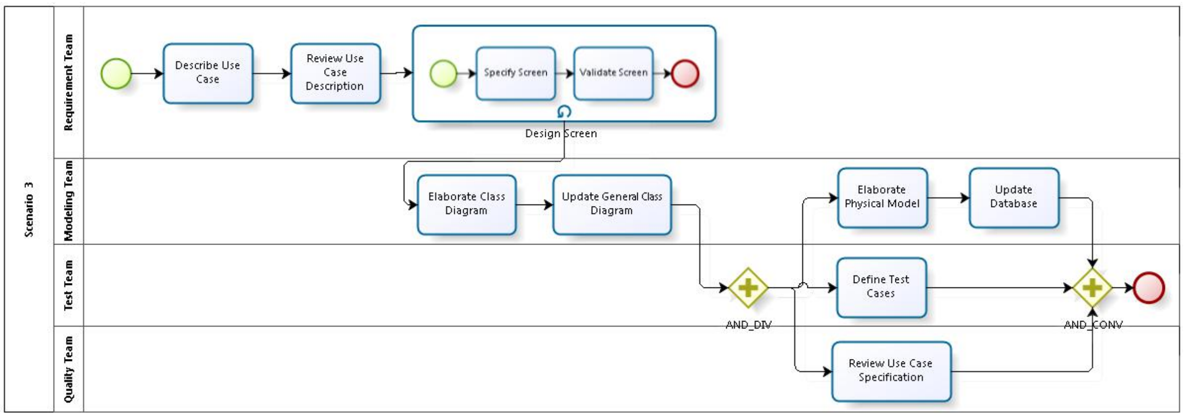


Figure 66: Scenario 3 – Variant process model

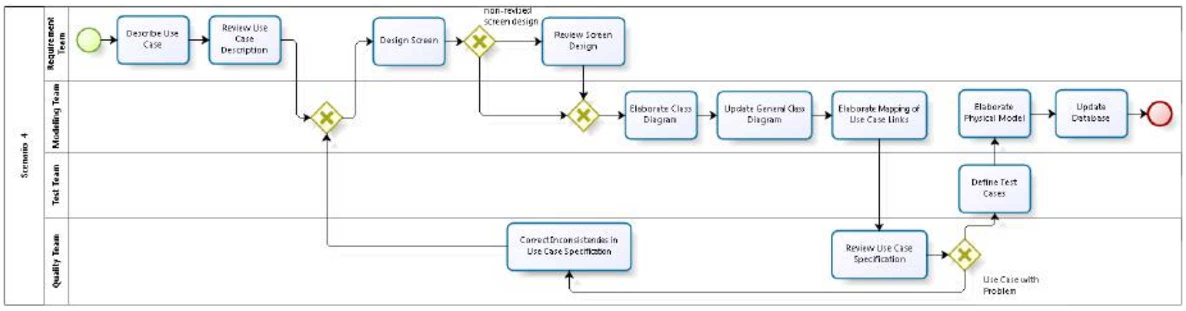


Figure 67: Scenario 4 – Variant process model

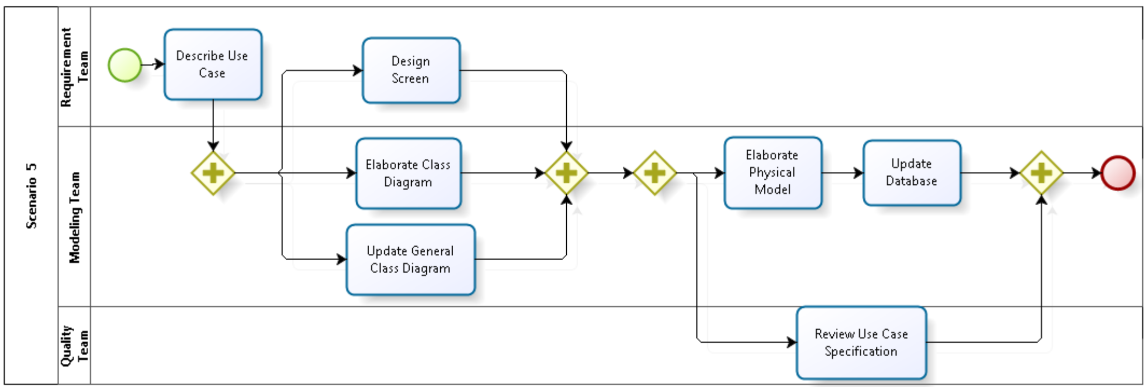


Figure 68: Scenario 5 – Variant process model

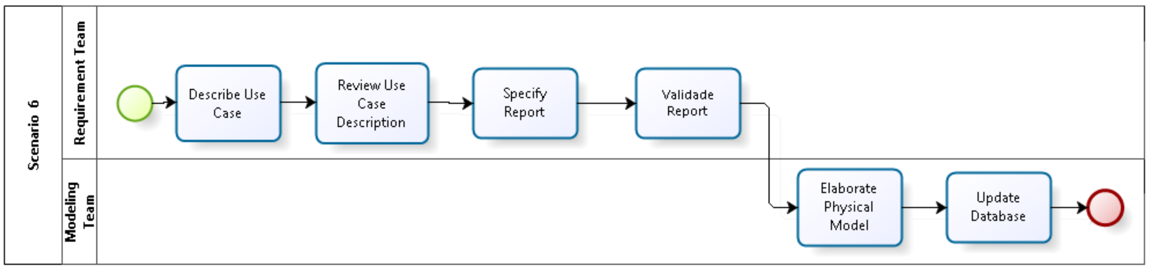


Figure 69: Scenario 6 – Variant process model