FlexSPMF: A Framework for Modelling and Learning Flexibility in Software Processes

Ricardo Martinho[1], João Varajão[2], and Dulce Domingos[3]

[1] School of Technology and Management, Polytechnic Institute of Leiria, Portugal
rmartin@estg.ipleiria.pt
[2] Department of Engineering, University of Trás-os-Montes e Alto Douro, Portugal
jvarajao@utad.pt
[3] Department of Informatics, Faculty of Sciences, University of Lisboa, Portugal
dulce@di.fc.ul.pt

Abstract. Software processes are dynamic entities that are often changed and evolved by skillful knowledge workers such as software development team members. Consequently, flexibility is one of the most important features within software process representations and related tools. However, in the everyday practice, team members do not wish for total flexibility. They rather prefer to learn about and follow previously defined advices on which, where and how they can change/adapt process representations. In this paper we present FlexSPMF: a framework for modelling controlled flexibility in software processes. It comprises three main contributions: 1) identifying a core set of flexibility concepts; 2) extending a Process Modelling Language (PML)'s metamodel with these concepts; and 3) providing modelling resources to this extended PML. This enables process engineers to define and publish software process models with additional (textual/graphical) flexibility information. Other team members can then visualise and learn about this information, and change processes accordingly.

1 Introduction

Software process modelling involves eliciting and capturing informal process descriptions, and converting them into process models. A process model is expressed by using a suitable Process Modelling Language (PML). A type of Process-Aware Information Systems (PAISs) called Process-centred Software Engineering Environments (PSEEs) supports the modelling, instantiation, execution, monitoring and management of software process models and instances.

Software processes are commonly held as dynamic entities that must evolve in order to cope with changes occurred in: the real-world software project (due to changing requirements or unforeseen project-specific circumstances); the software development organization; the market; and in the methodologies used to produce software [1]. Therefore, it should be possible to quickly implement new processes, to enable on-the-fly adaptations of running ones, to defer decisions regarding the exact process logic to runtime, and to evolve implemented processes over time.

M.D. Lytras et al. (Eds.): WSKS 2009, LNAI 5736, pp. 78–87, 2009.

Consequently, process flexibility has been identified as one of the most important features that both PMLs and PSEEs should support [2].

However, allowing for total process flexibility questions the usefulness of models themselves as guidance to a software development team work plan. Unlimited and unclassified flexibility hampers seriously the learning and reuse of information on changes made in past processes that can be useful in the future. Moreover, in the everyday business practice, most people do not want to have much flexibility, but would like to follow very simple rules to complete their tasks, making as little decisions as possible [3,4].

To corroborate this, case studies on flexibility in software processes (see, e.g., [5]) make evidence on the need of having (senior) process participants expressing and controlling the amount of changes that other process participants are allowed to make in the software process. This *controlled flexibility* can be defined as *the ability to express, by means of a PML,* which*,* where *and* how *certain parts of a software process should change, while keeping other parts stable* [4]. This faces strong requirements on PML comprehensibility, both because some users initially lack experience with modelling, and because evolution and learning will cause more frequent updates to the models.

We present in this paper the Flexibility Software Process Modelling Framework (FlexSPMF). It comprises the three following contributions:

1. *Identification* - consists in defining and systematising concepts that will be used to express this controlled flexibility in software processes;
2. *Metamodelling* - maps those concepts onto an existing PML's metamodel. Here, we adopt the Software & Systems Process Engineering Metamodel (SPEM) [6], and extend it by adding a flexibility sub-metamodel;
3. *Modelling* - concerns to the PML representation and PSEE tool support for modelling controlled flexibility. From our new metamodel structure, we derived a UML profile called FlexUML, which enhances UML Activity Diagrams (ADs) as the PML with a set of flexibility-related stereotypes and tagged values.

FlexSPMF provides process engineers the power of modelling controlled flexibility as guidance to software development team changes within software processes.

The rest of the paper is organised as follows: the next section provides an overview of FlexSPMF. Then, sections 3, 4 and 5 describe the details of each of the aforementioned contributions. Section 6 refers most prominent related work and section 7 concludes the paper.

2 Framework Overview

FlexSPMF fits our *identification*, *metamodelling* and *modelling* contributions within the software process model lifecycle, as depicted in Fig. 1. Here we can observe two main (human) roles: the process engineer and the software development team. The former contributes to both identification and metamodelling activities of the lifecycle, and is responsible for modelling software processes with

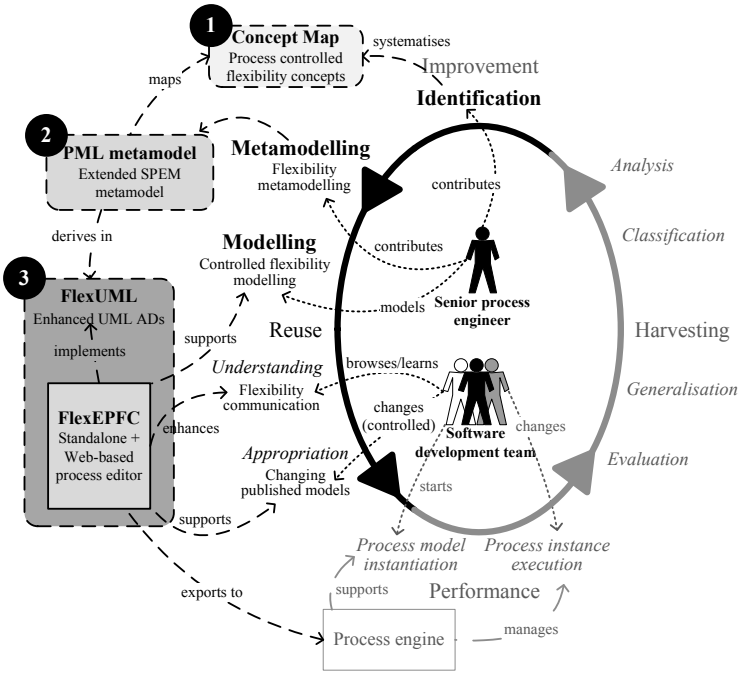

Fig. 1. FlexSPMF in the context of the software process model evolution lifecycle

controlled flexibility. Then, the software development team can browse and learn those models, including flexibility information on their process elements. Hence, they can also change those elements by following advice on previously configured flexibility options.

To better illustrate the framework's purpose, let us consider the example of an organisation that has a process engineer who is responsible for the general modelling and management of software processes and associated models. For a particular process model, s/he requires as mandatory the production of test cases associated with a `Test solution` activity.

On the other hand, s/he also wishes to allow the software testing team to perform a detailed modelling of this activity, which is not completely specified until process model instantiation (e.g., when starting up a software project). S/he does this on the assumption that the testing team members have better skills for the task, and that there is often the need for model adaptation to certain project management constraints (like time, cost or human resources).

Therefore, s/he requires PML resources to express within the process model *which*, *where* and *how* the `Test solution` activity can or cannot be changed. To accomplish this, s/he firstly needs to identify and systematise a delimited set of process controlled flexibility concepts which all other team members will understand and share in the language. Figure 1's first solution (number 1) refers to a Concept Map (Cmap) [7]. Cmaps are used as an informal ontology-based

approach to clarify and improve learning about concepts and relationships of a certain knowledge domain (see, e.g., [8]). In this case, the domain is the modelling of controlled flexibility within software process representations.

Solution number 2 refers to *metamodelling*, i.e., the mapping of the concepts onto a MetaObject Facility (MOF)-compliant metamodel. Here, we adopted SPEM, which serves as a reference metamodel for many Model-Driven Development (MDD) software processes, such as the Rational Unified Process (RUP), Open Unified Process (OpenUP), eXtreme Programming (XP) and Scrum. We extended it by adding our sub-metamodel, which comprises a set of class elements that addresses particularly the flexibility concern.

Solution number 3 refers to *modelling*, more precisely to the FlexUML profile. It introduces UML stereotypes to enhance SPEM ADs as a flexibility-aware PML. It also comprises its implementation in the FlexEPFC PSEE: a customised version of the original IBM's Eclipse Process Framework Composer (EPFC[1]). We added the possibility for a process engineer to express, in a first modelling step, controlled flexibility within a process model. After this, the process engineer can use FlexEPFC to publish the process model to an automatically generated web application. Besides providing an efficient way for the software development team to learn and browse process models, it also includes a web process editor. This allows the team to change the published models, according to the controlled flexibility previously modelled by the process engineer.

The next sections describe in more detail each FlexSPMF contribution.

3 Identification

Figure 2 illustrates the main Cmap for the process controlled flexibility domain, picturing its core concepts and relationships (please consult [9] for more detailed information on these). Briefly, it states that a *software process* is a combination of *elements* represented by a process *model*.

Software process modelling *elements* include: 1) *functional* elements (e.g. *phases*, *activities* and *steps*); 2) *behavioural* elements (e.g. control flow nodes such as *fork*, *join*, *decision* and *merge* nodes, as also iterative/parallel region elements); 3) *organisational* elements; and 4) informational elements (e.g. *data*, *artifact*, *work product* (intermediate and end) and *object* elements).

A process *model* depends on its *metamodel*, which establishes the structure of concepts, relationships and constraints that can be used in defining a process model. The metamodel usually defines a Process Modelling Language (PML), where all process modelling elements are specified. The Software & Systems Process Engineering Metamodel (SPEM) [6] is an example of a software process metamodel which defines UML ADs as the core PML.

Process models are then created as instances of the metamodel. They represent an arrangement of process elements in more or less specified sequences of activities, resources and resulting work products that will provide guidance for specific project workplans. Process models are templates which represent

[1] http://www.eclipse.org/epf

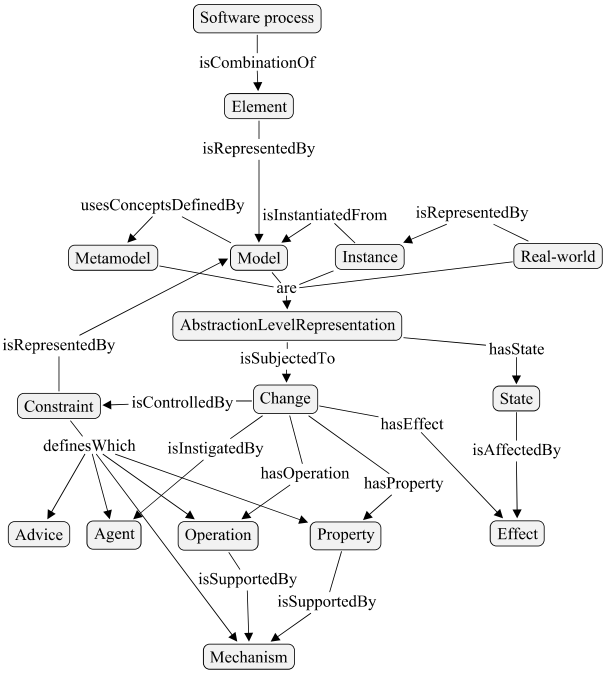

Fig. 2. Core Concept Map for sharing knowledge on process controlled flexibility

reusable process knowledge. Examples of software process models include the process structures comprised by well known software methods like the Unified Process [10].

Applying a process model for a specific software project is called process *instantiation*. An *instance* follows the model and provides specific data for each distinct project, such as activities' duration, (human) resource assignments, cost estimations and monitoring/control data updates. Multiple process instances may share the same process model.

On the contrary, *real-world* processes have a 1:1 multiplicity to process instances, as they reflect the activities, resources and products that are actually performed, used and produced by humans or tools. It describes what is really happening, and process participants may retrieve feedback which is used to update process running instances.

Metamodel, model, instance and real-world are distinct but correlated *abstraction levels* of process element *representations*. These representations are subjected to *changes*, which in turn have *effects* that can affect their *states*.

A *change* is characterised by *properties* and *operations*. Performing *change operations* includes creating, updating and deleting process elements, as well as moving them or realising element- and representation-specific operations such as *undo*, *skip* or *redo* an `Activity` in a process running *instance*. Actually, change operations are the actions that will change the state of the process elements.

Properties of change are not dependent on a process element's type, but characterise multiple and general dimensions of a change. Possible implementations of properties of change commonly referred in literature include [11,12]: 1) *extent* (incremental or revolutionary); 2) *duration* (temporary or permanent); 3) *swiftness* (immediate or deferred propagation between models and running instances) and; 4) *anticipation* (planned or ad-hoc changes). Both operations and properties are supported by corresponding *mechanisms*. For example, executing an *add* operation on a process model implies the use of a software tool that, besides supporting process editing features, also provides verification of conformance, consistency or compliance rules associated with that operation.

Changes are instigated (put into action) by *agents* of change. Agents of change need to have the ability to set a mechanism of change into motion. In the software process context, the agent of change is responsible for triggering mechanisms of change that will result in an effect of change endured by one or more process element representations. Agents of change may be software components that automatically change process element representations under some criteria, or humans such as software process engineers, project managers, analysts, designers, programmers and testers, that need to change/adjust software processes.

A change can be controlled by *constraints*, which are also represented by *models*. These contain *advice* on operations, properties, mechanisms and/or agents that should be considered when changing a certain process element representation. Advice on a change can be a *value*- or *text*-based attribute (e.g., *60%* or *recommended*) , or any other combination of values that best fit the process element representation to which the advice is associated.

For example, a constraint of change may impose that *a* `Test solution` *activity instance representation cannot be skipped*. The modelling of this constraint can be made using a three component tuple of the form *(abstraction level, operation, advice)*, with the values *(instance, skip, denied)*.

4 Metamodelling

We use a UML class metamodel structure that reflects the above concepts, and that can be used to extend the existing metamodel of a PML. Here, we provide a concrete example by using SPEM and UML ADs as the metamodel and corresponding PML to be extended. Figure 3 pictures our derived FlexSPMF metamodel.

The structure is based on the decorator design pattern from Gamma et al. [13]. This is the leftmost and gray-shaded structure which inherits from a common abstract `SPEMProcessModellingElement`. The decorator pattern allows the attachment of additional responsibilities to an object dynamically. In our context, when designing a process model, a process engineer can pick any process element (such as an `Activity`) and decorate it with one or more change decorators, represented by `ChangeDecorator`'s specialised classes. These derive from the aforementioned concepts of change operation, property, mechanism, advice and abstraction level.

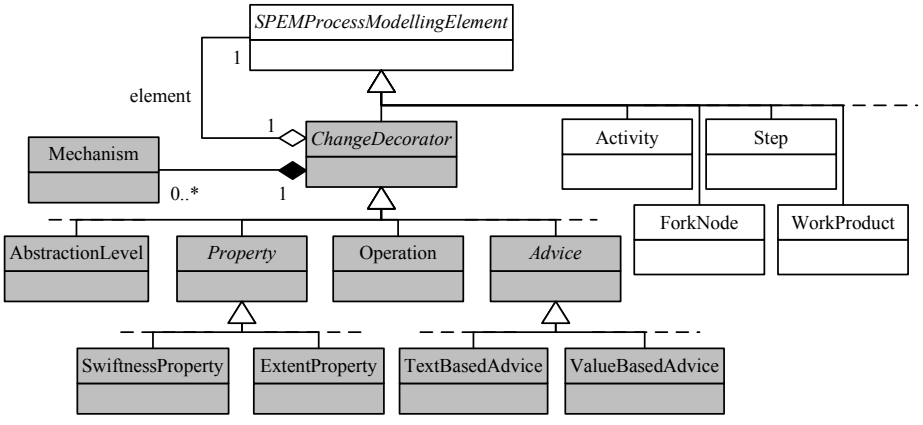

Fig. 3. FlexSPMF metamodel structure as an extension to the SPEM metamodel

When applied to a process element, a decorator represents one of the components that form a *constraint of change*. Taking from our former example, the tuple *(abstraction level, operation, advice)* maps to a combination of three decorators, namely: `AbstractionLevel` (*instance*), `Operation` (*skip*) and `TextBasedAdvice` (*denied*), which are applied to instances of the `Test solution` activity.

Mechanisms can be associated with any decorator, providing implementation details according to a process element's type.

The next section describes the way a process engineer can use the extended SPEM ADs and FlexEPFC to model controlled flexibility in software processes.

5 Modelling

For the *modelling* contribution we developed FlexUML: a UML profile for ADs (see [14] for further details). Each `ChangeDecorator` concrete specialisation class of Fig. 3's metamodel corresponds to a FlexUML stereotype with its own tagged values. Figure 4 presents two applications of FlexUML, noted by the names of the stereotypes between «guillemets» above the process elements' graphical notations and callouts with their respective tagged values (according to [15]).

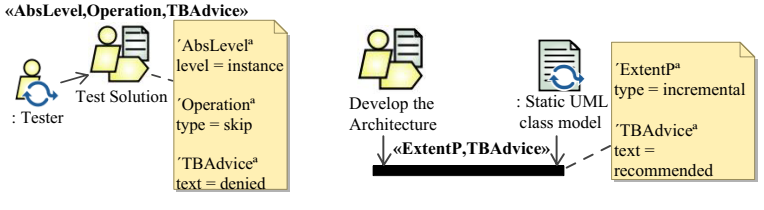

Fig. 4. `Activity` (left) and `Join node` (right) FlexUML stereotype applications

The profile applications are made to an activity (`Test solution`) and to a join node. The former includes the «`AbsLevel,Operation,TBAdvice`» stereotypes, which values advise to not skip the activity's instances. The latter has the «`ExtentP,TBAdvice`» stereotypes that advise as *recommended* an incremental change.

Once a process engineer finishes applying the flexibility stereotypes, s/he can use FlexEPFC to generate a web-based published version of the process model. The original auto-generation from EPFC does not allow any changes to this published version. To fulfil our requirement of also enabling the team to change the models, we embedded the process editor also in the web-based application. This way, besides browsing, the team can also learn about the advised flexibility, and perform changes accordingly.

Figure 5 is a snapshot of this customised web-based process editor which we called WebFlexEPFC (see [16] for more details). The snapshot shows the editing/changing of an OpenUP `Elaboration phase` join node, but according to

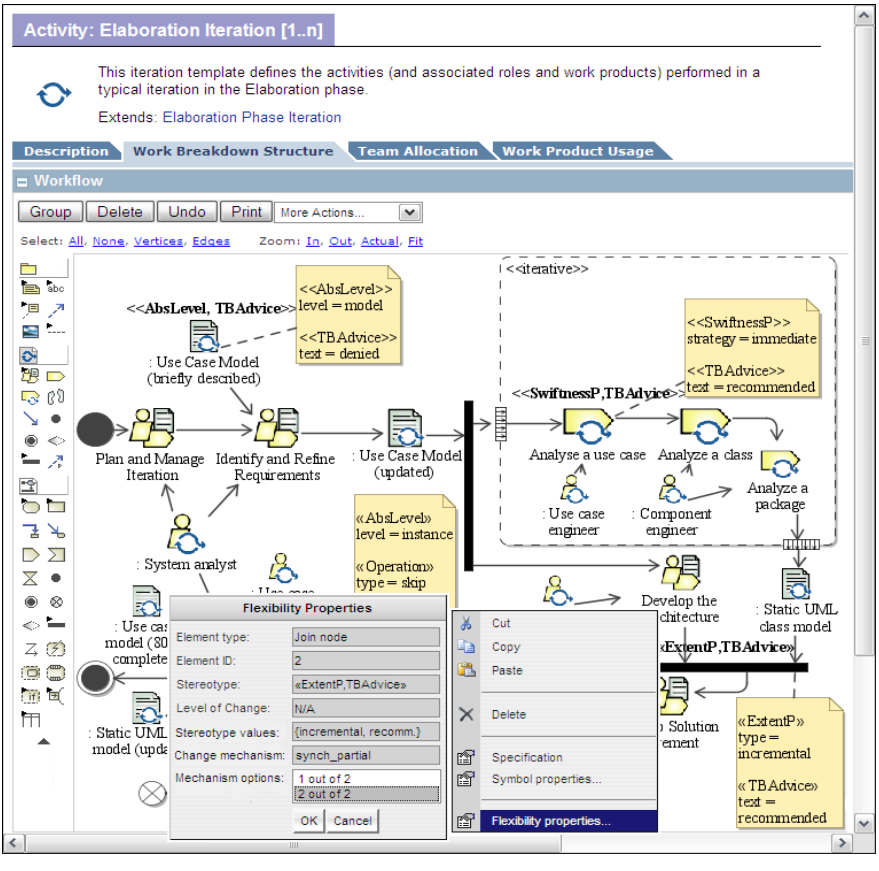

Fig. 5. WebFlexEPFC: changing a join node of an OpenUP Elaboration phase model

the previously modelled constraint of change (noted by the «ExtentP,TBAdvice» stereotype applications).

6 Related Work

Recent process-aware flexibility taxonomies include the one by Regev et al. in [11]. We use some of the concepts referred here, including *abstraction level*, *subject* and *property of change*. These and other change concepts are also present in the taxonomy proposed by Schonenberg et al. in [12], under a four type-based flexibility classification: *design*, *deviation*, *underspecification* and *change*. *Constraints of change* in process models are analysed by Wörzberger et al. in [17]. They propose modelling tool artifacts to support correctness, compliance and consistency constraint modelling and checking.

Although there is no shared classification of concepts among these works, they have the same intent of FlexSPMF *identification* contribution. However, our concept map focuses mainly on the definition of a core set of concepts, delegating on the decorator-based metamodel structure the possibility of combining them interchangeably, to form adjustable categories on controlled flexibility.

In [5], Cass and Osterweil advocate that, in spite of software design requiring a lot of creativity and insight, some process rigidity seems necessary. They conclude that combining flexibility rules with process guidance helps designers to produce better software designs, and to produce them faster.

A framework approach on flexibility-aware PAIS is also proposed by Reichert et al. in [2]. The ADEPT2 flexible PAIS is able to adapt process instances to changes occurred in real-world processes, on a quest to support most of the workflow patterns, including the exception ones.

None of these works approach the challenge of modelling controlled flexibility, i.e., expressing in process models how much flexibility is allowed, in order to restrain changes on declarative (starting as 100% flexible) software processes.

7 Conclusions

The main objective of FlexSPMF is to provide process engineers with means to control the flexibility that team members can enjoy when guided by a software process. Our contributions include a controlled flexibility concept map, metamodel and modelling language based on SPEM UML ADs.

The metamodel provides a way to associate constraints of change by decorating process elements with interchangeable advices on operations and properties of change. It is also a non-intrusive metamodel extension, since it does not modify the structure of the original one.

Being UML a standard *de facto* nowadays, the use of a PML other than UML ADs would increase the software development team's learning curve about the semantics of controlled flexibility expressed within software process models. Software teams already use UML ADs and stereotypes on a daily basis to develop software. They also recognise them as the PML used for modelling well known SPEM-based processes like RUP, XP, OpenUP and Scrum.

References

1. Cugola, G.: Tolerating Deviations in Process Support Systems via Flexible Enactment of Process Models. IEEE Transactions on Software Engineering 24(11), 982–1001 (1998)
2. Reichert, M., Rinderle-Ma, S., Dadam, P.: Flexibility in process-aware information systems. LNCS Transactions on Petri Nets and Other Models of Concurrency (ToPNoC) 2, 115–135 (2009)
3. Bider, I.: Masking Flexibility Behind Rigidity: Notes on How Much Flexibility People are Willing to Cope With. In: Pastor, Ó., Falcão e Cunha, J. (eds.) CAiSE 2005. LNCS, vol. 3520, pp. 7–8. Springer, Heidelberg (2005)
4. Borch, S.E., Stefansen, C.: On Controlled Flexibility. In: Proc. of the 7th Workshop on Business Process Modeling, Development and Support (BPMDS), pp. 121–126 (2006)
5. Cass, A.G., Osterweil, L.J.: Process Support to Help Novices Design Software Faster and Better. In: Proc. of the 20th IEEE/ACM Intl. Conference on Automated Software Engineering (ASE), pp. 295–299 (2005)
6. OMG: Software Process Engineering Metamodel Specification, v2.0. Technical report, Object Management Group (2007)
7. Novak, J.D., Cañas, A.J.: The theory underlying concept maps and how to construct and use them. Technical report, IHMC CmapTools, 2006-01 Rev 2008-01, Florida Institute for Human and Machine Cognition (2008)
8. Razmerita, L., Lytras, M.D.: Ontology-based user modelling personalization: Analyzing the requirements of a semantic learning portal. In: Lytras, M.D., Carroll, J.M., Damiani, E., Tennyson, R.D. (eds.) WSKS 2008. LNCS (LNAI), vol. 5288, pp. 354–363. Springer, Heidelberg (2008)
9. Martinho, R., Domingos, D., Varajão, J.: On a concept map for the modelling of controlled flexibility in software processes. Technical report TR-2009-12. Dep. de Informática, Faculdade de Ciências da Universidade de Lisboa (May 2009)
10. Jacobson, I., Booch, G., Rumbaugh, J.: The Unified Software Development Process. Addison-Wesley Longman Publishing Co., Inc, Boston (1999)
11. Regev, G., Soffer, P., Schmidt, R.: Taxonomy of Flexibility in Business Processes. In: Input to the 7th Workshop on Business Process Modeling, Development and Support (BPMDS 2006) (June 2006), `http://lamswww.epfl.ch/conference/bpmds06/taxbpflex`
12. Schonenberg, H., Mans, R., Russell, N., Mulyar, N., van der Aalst, W.M.P.: Towards a taxonomy of process flexibility. In: CAiSE 2008, pp. 81–84 (2008)
13. Gamma, E., Helm, R., Johnson, R., Vlissides, J.: Design Patterns: Elements of Reusable Object-Oriented Software. Addison-Wesley, Reading (1995)
14. Martinho, R., Domingos, D., Varajão, J.: FlexUML: A UML Profile for Flexible Process Modelling. In: Proc. of the 19th Intl. Conference of Software Engineering and Knowledge Engineering (SEKE), pp. 215–220 (2007)
15. OMG: Unified Modeling Language: Superstructure, version 2.0. Technical report, Object Management Group (2005)
16. Martinho, R., Varajão, J., Domingos, D.: A two-step approach for modelling flexibility in software processes. In: Proc. of the 23rd IEEE/ACM Intl. Conference on Automated Software Engineering, ASE (2008)
17. Wörzberger, R., Kurpick, T., Heer, T.: On correctness, compliance, and consistency of process models. In: Proc. of the 17th IEEE Intl. Workshops on Enabling Technologies: Infrastructures for Collaborative Enterprises, WETICE 2008 (2008)