# Modelling and learning controlled flexibility in software processes

Ricardo Martinho*

Department of Informatics Engineering, School of Technology and Management
Polytechnic Institute of Leiria, Portugal
E-mail: rmartin@estg.ipleiria.pt
*Corresponding author

João Varajão

Centro ALGORITMI
Department of Engineering
University of Trás-os-Montes e Alto Douro, Portugal
E-mail: jvarajao@utad.pt

Dulce Domingos

Department of Informatics, Faculty of Sciences
University of Lisboa, Portugal
E-mail:dulce@di.fc.ul.pt

**Abstract:** Software processes are dynamic entities that are often changed and evolved by skilful 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.

**Keywords:** software processes; learning; controlled flexibility; modelling; language.

**Reference:** to this paper should be made as follows: Martinho, R. and Varajão, J. and Domingos, D. (20xx) 'Modelling and learning controlled flexibility in software processes', *Int. J. Knowledge and Learning*, Vol. x, No. x, pp.xx–xx.

**Bibliographical notes:** Ricardo Martinho is an Assistant Professor at the Department of Informatics Engineering, School of Technology and Management, Polytechnic Institute of Leiria, Portugal. He has a BSc in

Electrotechnic and Computers Engineering from University of Coimbra, and a Masters degree in Computer Science from Technical University of Lisboa. He is currently a member and researcher of the LASIGE laboratory at the University of Lisboa, and a PhD student at the University of Trás-os-Montes e Alto Douro. His main research areas include process modelling languages, process-aware information systems, flexibility in software engineering models and metamodels.

João Varajão is currently a Professor at the Department of Engineering of the University of Trás-os-Montes e Alto Douro, Vila Real, Portugal, where he teaches undergraduate and postgraduate courses on information systems management and software engineering. He also supervises several MSc and PhD projects in the domain of information systems management, information systems outsourcing and e-business. He earned his PhD and MSc from the University of Minho. His scientific interests include information systems management, chief information systems profession, enterprise information systems and electronic business systems. He has over 120 publications, including books, book chapters, refereed publications, and communications at international conferences. He serves as associate editor and member of editorial board for international journals and has served in several scientific committees of international conferences.

Dulce Domingos is a Professor at Department of Informatics, Faculty of Sciences, University of Lisboa, Portugal. She graduated in Computer Science at Faculty of Sciences, University of Lisboa, has a Master in Electrotechnic and Computers Engineering, by Technical University of Lisboa and has a PhD in Computer Science, by Faculty of Sciences, University of Lisboa. She is a member of the LASIGE laboratory at University of Lisboa. Her current interests include adaptive workflow management systems, access control models, systems and applications.

## 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 (Cugola 1998). 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.

Consequently, process flexibility has been identified as one of the most important features that both PMLs and PSEEs should support (Reichert et al. 2009b).

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 (Bider 2005, Borch and Stefansen 2006).

To corroborate this, case studies on flexibility in software processes (see, e.g., Cass and Osterweil (2005)) 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* (Borch and Stefansen 2006). 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:

- *Identification* – consists in defining and systematising concepts that will be used to express this controlled flexibility in software processes;
- *Metamodelling* – maps those concepts onto an existing PML's metamodel. Here, we adopt the Software & Systems Process Engineering Metamodel (SPEM) (OMG 2008), and extend it by adding a flexibility sub-metamodel;
- *Modelling* – concerns to the PML representation and 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.

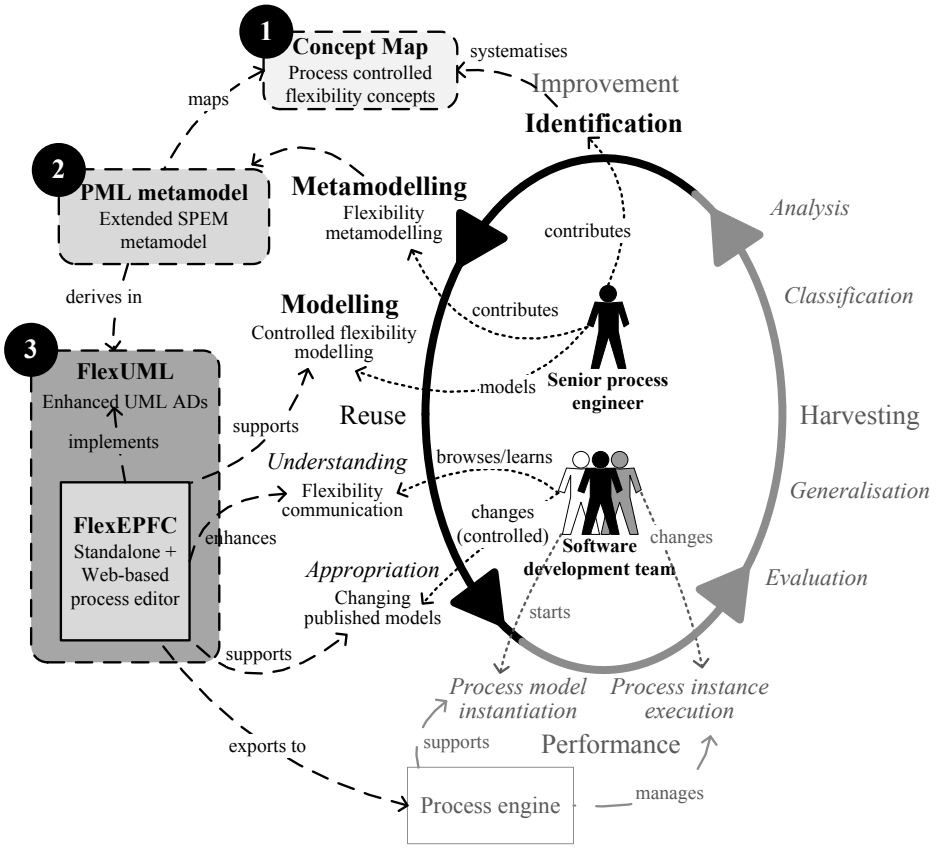
The rest of the paper is organised as follows: the next section provides an overview of FlexSPMF. Then, sections 3, 4 and 5 describe the details of each of the aforementioned contributions. Section 6 refers most prominent related work and section 7 concludes the paper.

## 2 Framework overview

FlexSPMF fits our *identification*, *metamodelling* and *modelling* contributions within the software process model lifecycle, as depicted in Figure 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 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

**Figure 1** FlexSPMF in the context of the software process model evolution lifecycle




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) (Novak and Cañas 2008). 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., Razmerita and Lytras (2008)). 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 Open Unified Process (OpenUP) (Eclipse Foundation 2008), eXtreme Programming (XP) (Beck 1999) and Scrum (Schwaber and Beedle 2001). 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 and tool support. It introduces UML stereotypes to enhance UML ADs as a flexibility-aware PML. It also comprises its implementation in the FlexEPFC: 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

We adopted Cmaps and CmapTools (Cañas et al. 2004) as a less formal but proven method for explaining and communicating domain knowledge (Hoffman and Woods 2000). CmapTools has been developed over the previous decade as an intuitive, human-centred computer interface for creating and managing concept maps. We use it to create and manage the relationships between the proposed concepts.

Our goal does not comply, for the time being, automated mechanisms as possible targets for sharing knowledge. We aim at enhancing the learning and sharing of process controlled flexibility-related concepts among process engineers and software team members. Nevertheless, CmapTools includes export and import features for ontologies represented in formal Web Ontology Language (OWL)-derived languages, which provides us the means for a latter and more formalised ontology approach.

To achieve our purpose, we begin by providing in the next section a focus question to which our Cmaps must be able to answer. Next, we present the corresponding diagrams and descriptions.

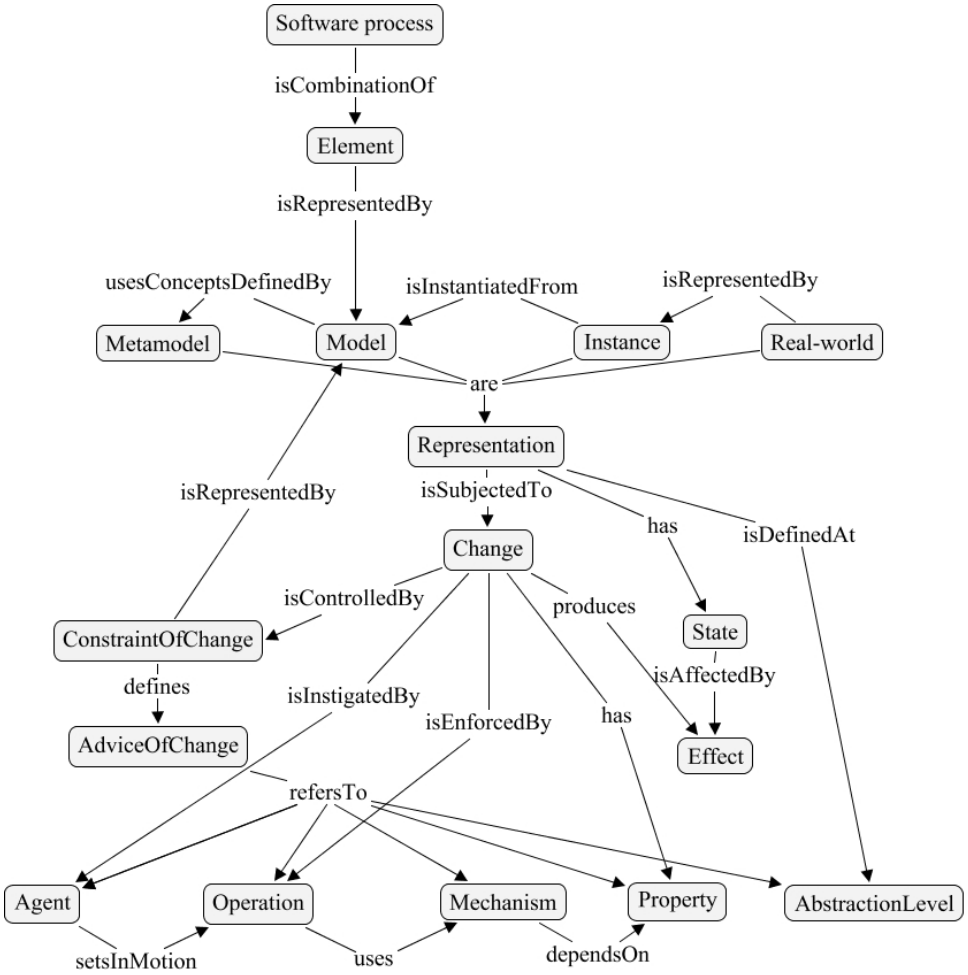
### *3.1 Focus question*

Recent studies advocate that, when a Cmap is constructed with a particular purpose and for a certain audience, its objective is better achieved by providing a \textit{focus question}, to which the Cmap will be designed to answer (Derbentseva et al. 2007, Novak and Cañas 2008). Moreover, and to prompt dynamic thinking and dynamic relationships between concepts, the question should be of type *"How does the concept X work?"*, rather than *"What is concept X?"*.

In the context of this work, we propose, in the next section, Cmaps to provide answers for the following focus question:

*"How can software processes be subjected to changes in a controlled way?"*

**Figure 2** Core Concept Map for sharing knowledge on process controlled flexibility




## *3.2 Diagrams*

In this section we present the diagrams for the concepts and relationships that provide answers to the focus question. We begin by illustrating in Figure 2 the main Cmap for this purpose. As we carry along describing those concepts, we also provide helpful descriptions on related contexts and examples (not represented in the diagrams, for not being within their focus).

The diagram on Figure 2 shows that a *software process* is a combination of *elements* represented by a *model*. These elements can be used to express correlated process perspectives, including (Curtis et al. 1992):

- *Functional* – what process elements are being performed (e.g. *phase*, *activity* and step elements);

- *Behavioural* – when process elements are performed (e.g. control flow nodes such as *fork*, *join*, *decision* and *merge* nodes, as also iterative/parallel region elements);
- *Organisational* – where and by whom (which *agents*) in the organisation process elements are performed (e.g. *role* elements, and resources like physical communication mechanisms, physical media and location used for storing entities); and
- *Informational* – informational entities produced or manipulated by a process (e.g. *data*, *artifact*, *work product* (intermediate and end) and *object* elements). It includes both structure of informational entities and the relationships among them.

Back to our diagram, a process element *model* depends on its *metamodel*, which establishes its structure of concepts, relationships and constraints. An overall process metamodel usually defines a PML, where all process modelling elements are specified. SPEM 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, and include as more or less specified arrangements of process element model representations such as activities, resources and resulting work products. Examples of software process models include the (textual and graphical/diagrammatic) process representations comprised by well known software methods such as the Unified Process (Jacobson et al. 1999).

Applying a process model for a specific software project is called process *instantiation*. An *instance* follows the model and needs to be provided with 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 process element *representations* defined at distinct but correlated *abstraction levels* of modelling. These representations are subjected to *changes*, which in turn have *effects* that can affect their *states*.

A *change* is characterised by *properties* and enforced by *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 process elements.

*Properties of change* are not dependent on a process element's type, but characterise multiple and general dimensions of a change. Concrete properties of change commonly referred in literature include (Regev et al. 2006, Schonenberg et al. 2007):

- *Extent* of change, denotes whether a change is only introduced to an already existing process model (*incremental* change), or abolishes the existing process model and creates a completely new one (*revolutionary* change). Often experts are required to do revolutionary changes to the whole or part of a process model;

- *Duration* of change, states that changes can be *temporary* or *permanent* changes. Temporary changes are valid for a limited period of time, and permanent changes are valid until the next permanent change;
- *Swiftness* of change, expresses whether changes are to be applied *immediate*ly to all process model instances (also the running ones), or *deferred* only to new instances of the changed process model;
- *Anticipation* of change, defines whether the change is *planned* or *ad-hoc*. Ad-hoc changes are often made to tolerate exceptional situations, and planned changes are often part of a process redesign.

Operations use *mechanisms* in order to be enforced. 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, and compliance rules associated with that operation. All these resources can be considered as mechanisms of change (Ross et al. 2008). These mechanisms can depend on the properties that are desired for a change to have. For example, if the former add operation was to be valid during two weeks (duration property of change), the mechanism(s) used to support it would have also to comprise this property.

Changes are instigated (put into action) by *agents* of change. In the software process context, the agent of change is responsible for setting into motion *operations* 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 process representations.

A change can be controlled by *constraints*, which are also represented by *models* that the process engineer configures. Constraints define *advices of change*, which refer to the abstraction level, operations, properties, mechanisms and/or agents that should be considered when changing a certain process element representation. Advice on a change can be expressed by a *value*- or *text*-based attribute (for instance, *60%* or *recommended*), or any other combination of values that best fit the process element representation to which the advice is associated.

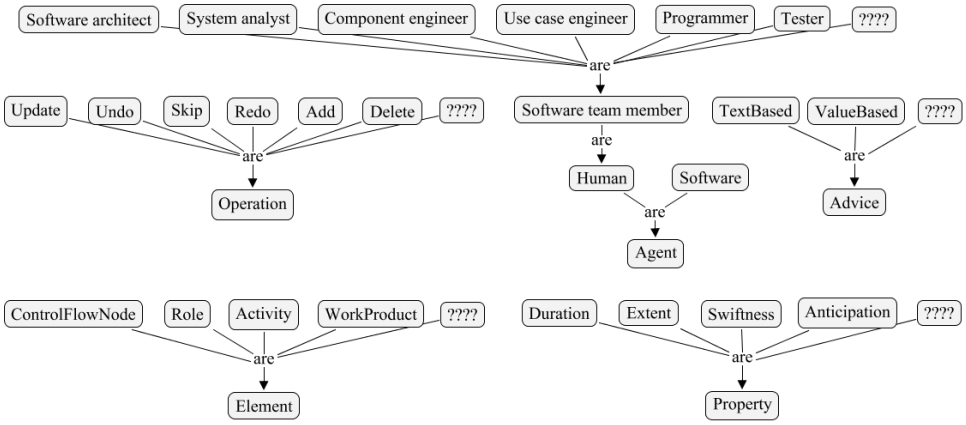
For example, a constraint of change may impose that, for a certain *Test solution* activity process element, *"skipping instance executions is denied"*.

The modelling of this constraint can be made by composing a tuple with three semicolon separated components of the form:

(abstraction level: String = 'instance'; operation: Operation= 'skip';
advice: TextBasedAdvice = 'denied').

Figure 3 represents some generalisation relationships pertaining concepts already presented in Figure 2 and referred above. Some concepts are filled with four question marks. This means that the general concept can have more or different specialised elements beyond the examples represented, since they can depend on factors such as the modelling language, application domain, tool support and/or organisational context.

**Figure 3** Generalisation relationships for some of the concepts illustrated in Figure 2



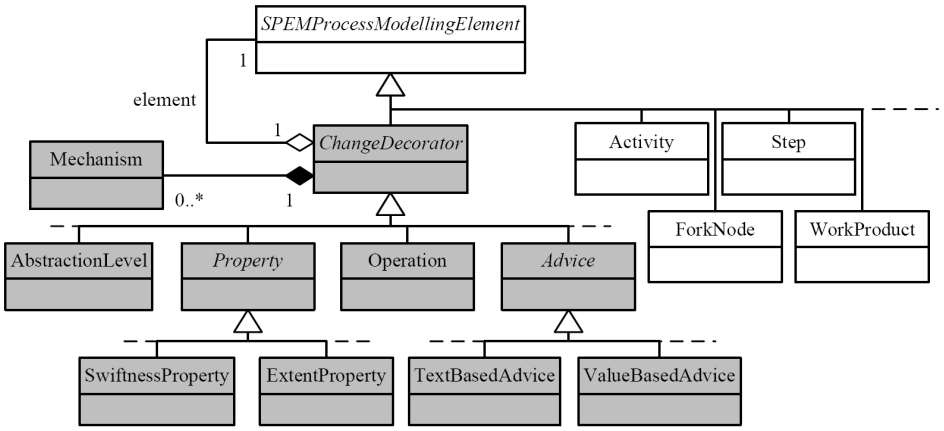

## 4 Metamodelling

We use a UML class diagram to express the metamodel structure that reflects the above concepts, and that can be used to extend the metamodel of an existing core PML. Here, we provide a concrete example by using SPEM and UML ADs as the metamodel and corresponding PML to be extended. Figure 4 pictures our derived FlexSPMF metamodel. The structure is based on the *decorator* design pattern from (Gamma et al. 1995). This is the leftmost and gray-shaded structure which inherits from a common abstract process modelling element which we called *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*.

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 referred tuple of the form *(abstraction level; operation; advice)* maps to a combination of three decorators and values, namely: *AbstractionLevel* (*instance*); *Operation* (*skip*) and *TextBasedAdvice* (*denied*), which would be applied to the model representation element *Test solution*.

**Figure 4** FlexSPMF metamodel structure as an extension to the SPEM metamodel




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

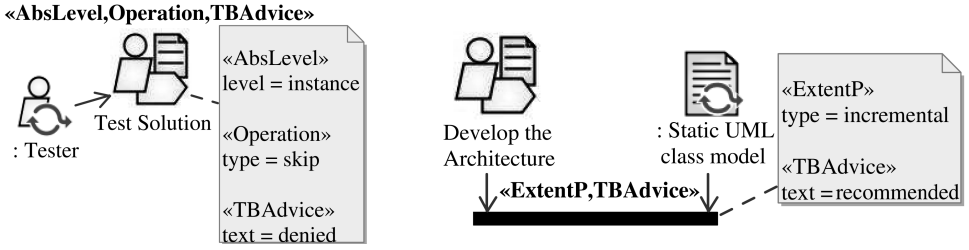
The next section describes the way a process engineer can use the extended SPEM ADs and FlexEPFC to model controlled flexibility in software processes.

## 5 Modelling

For the *modelling* contribution we developed FlexUML: a UML profile for ADs (see Martinho et al. (2007) for further details). Each concrete specialisation of the *ChangeDecorator* class of Figure 4's metamodel corresponds to a FlexUML stereotype with its own tagged values. Figure 5 presents two applications of FlexUML, noted by the names of the stereotypes between «guillemets» above process elements' graphical notations, and callouts with their respective tagged values (according to OMG (2005)).

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 *deny skipping* the activity's *instance* executions. The latter has the *«ExtentP,TBAdvice»* stereotypes that advise as *recommended* an *incremental* change.

**Figure 5** FlexUML stereotype applications made onto activity (left) and join node (right) process elements



After establishing a flexibility-aware PML metamodel and providing a UML profile to express controlled flexibility in software process models, we developed FlexEPFC - an Electronic Process Guide (EPG) generator tool, based on IBM's open source EPFC. The original EPFC is a Standard Widget Toolkit ($SWT^2$)-based standalone application where process engineers compose *configurations* for software processes by using an *authoring* Eclipse perspective. After composing a configuration, the process engineer can publish it onto an auto-generated web application, where process participants can use it as a typical EPG tool for browsing and learning software processes.

Our FlexEPFC extends and customises the original EPFC project to include:

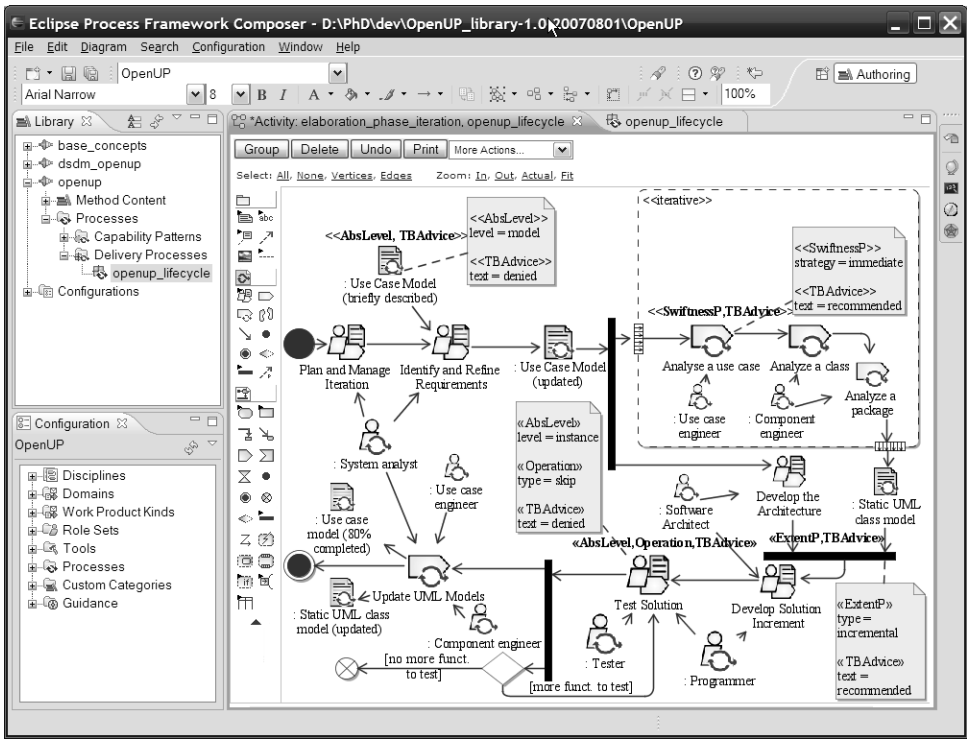
- The FlexUML metamodel structure presented in Figure 4, and corresponding specific User Interface (UI) elements to configure constraints of change for a certain process modelling element;
- New process modelling features to enable process participants to perform changes in published configurations, using the auto-generated web application.

The EPFC tool is built from around fifty distinct Eclipse Plugin projects. The one that refers to the software process engineering metamodel is the Unified Method Architecture (UMA) plugin project. Our approach to FlexEPFC began by extending this project with our metamodel structure. We added the *ChangeDecorator* a*nd Mechanism* elements and corresponding implementation classes and interfaces onto the UMA project, and implemented Figure 4's metamodel associations accordingly.

Next, we adjusted EPFC's UI to allow a process engineer to configure constraints of change for process modelling elements. This was done by extending another EPFC's plugin project: the Authoring User Interface project. It deals with the user interface logic and presentation for the authoring perspective of EPFC, where the process engineer composes and configures software process models and related content. Figures 6 and 7 illustrate two different view customisations of this authoring perspective, regarding an OpenUP-based software process model configuration.

Figure 6 shows an AD for the *Activity: elaboration_phase_iteration, openup_lifecycle* process element. The view also comprises an AD editor, which we improved from the original EPFC regarding the pallete of process modelling elements, layout and drawing interface. We also customised this editor to enable the modelling of controlled flexibility. A process engineer can do this by applying FlexUML stereotypes onto process elements, which will be represented as the ones illustrated in Figure 5.

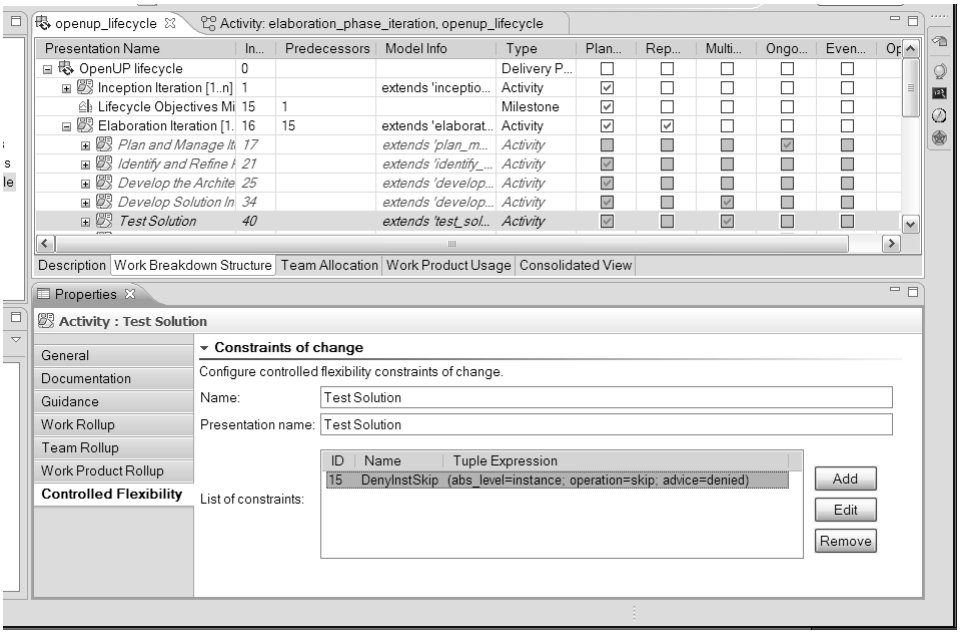
**Figure 6** FlexEPFC authoring perspective, improved with an enhanced AD editor




Alternatively, the process can model controlled flexibility by associating textually-defined constraints of change to a process element, without using a diagram editor. Figure 7 shows a *WorkBreakDownStructure* view for the same *openup_lifecycle* software process configuration, where we can see some of the general properties of all its composing elements.

Notice the selected *Test Solution* activity process element, and the corresponding properties below, where several property pages can be seen, including *General*, *Documentation*, *Guidance*, *Work RollUp*, *Team RollUp* and *Work Product RollUp*. We added the *Controlled Flexibility* property page, which is the one activated in Figure 7. Within this page, a process engineer can manage (*add*, *edit* or *remove*) a list of constraints of change associated with the process element. These constraints can be textually defined by providing a name and a tuple expression. Figure 7 shows a constraint with the tuple expression *(abs_level=instance; operation=skip; advice=denied)*, defined for the *Test Solution* process element.

**Figure 7** FlexEPFC authoring perspective, improved with an additional *Controlled Flexibility* property page

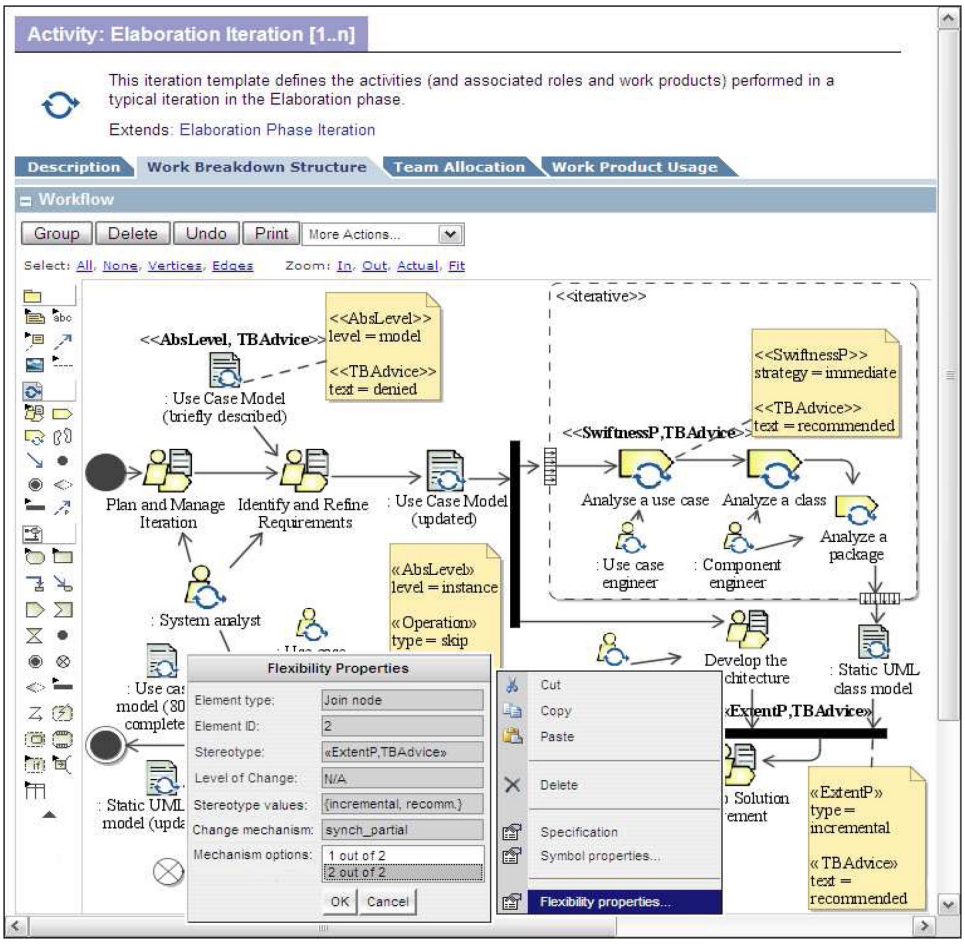



Once a process engineer finishes the modelling of constraints of change (either through stereotype applications or textually-defined tuple expressions), s/he can use FlexEPFC to generate a web-based published version of the process model. The original website auto-generation plugin project from EPFC does not allow for any changes to be made onto web published process models. To fulfil our requirement of also enabling the software team to change these models, we customised the auto-generation engine to include, within the web published process models, a process editor similar to the one made for FlexEPFC. This way, besides browsing, the team can also learn about the advised controlled flexibility, and actually perform changes accordingly.

Figure 8 is a snapshot of this customised web-based process model perspective, which we called WebFlexEPFC. It is an EPG-like web application which also allows software team members to change process element representations, according to the previously configured constraints of change in FlexEPFC. Particularly, Figure 7 shows the editing/changing of a join node from an OpenUP web published process model, but according to the previously modelled constraint of change (noted by the *«ExtentP,TBAdvice»* stereotype applications).

Summing up, the *modelling* of controlled flexibility was exemplified within EPFC by following a *two-step approach* (see Martinho et al. (2008) for more details): 1) process engineers configure software process models and corresponding constraints of change in FlexEPFC's authoring perspective; and 2) software team members access a web published version of the configured process models, and are allowed to change them according to the previously configured constraints of change.

**Figure 8** WebFlexEPFC: changing a join node of an OpenUP phase process element




## 6 Related work

Regarding our *identification* contribution, one of the most prominent foundational research works on software process flexibility is the one presented by Madhavji (1992) and his Prism model of changes. The author has identified several items strongly related with the software development environment, which are subjected to continuous change. The basic items of change are *people*, *policies*, *laws*, *resources*, *processes* and *results*. For various predictable and unpredictable reasons, such items may need to be changed on an ongoing basis. Therefore, the Prism model of change included the following features:

- Separation between changes made to the described items and changes made to the environmental facilities encapsulating those items;

- A *dependency structure* that describes the various items and their interdependencies. It is used to identify the items affected by a given change;
- A *change structure* that classifies, records and analyses change-related data, and makes qualitative judgement of the consequences of a change;
- Identification of the many distinct properties of a change;
- A built-in mechanism for providing feedback on changes made.

In spite of being an extensive conceptual and architectural framework on changes, Prism does not focus on describing how those changes can be controlled before actually being performed.

According to a recent taxonomy proposed in (Regev et al.2006), process flexibility can be classified in three orthogonal dimensions:

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

2 *Subject of change*, representing *which* process modelling elements are to be changed, and, consequently, which related perspective(s) (such as the ones in (Curtis et al.1992) mentioned above);

3 *Properties of change*, denoting *how* a process modelling element can be changed. Examples include properties mentioned above, such as *extent*, *duration*, *swiftness* and *anticipation* of change.

This taxonomy was very useful to establish some of the concepts we use in our Cmaps, namely the concepts of *abstraction level of change* and *property of change*. These can be used to compose the tuple of a *constraint of change* to control the changes made to a process element representation. Process element representations correspond (although not explicitly), in this taxonomy, to the *subjects of change* concept.

In a sequence of also recent contributions, Schonenberg et al. (2007) propose another taxonomy on process flexibility. Four distinct approaches are identified, each having its own application area. They are *flexibility by*:

- *design* – for handling anticipated changes in the operating environment, where supporting strategies can be defined at design-time;
- *deviation* – for handling occasional unforeseen behaviour, where differences with the expected behaviour are minimal;
- *underspecification* – for handling anticipated changes in the operating environment, where strategies cannot be defined at design-time, because the final strategy is not known in advance or is not generally applicable;
- *change* – either for handling occasional unforeseen behaviour, where differences require process adaptations, or for handling permanent unforeseen behaviour.

The same authors also survey contemporary approaches in (Schonenberg et al.2008), and evaluate their taxonomy against a selection of PAIS, namely ADEPT1 (Reichert et al.2003), YAWL (van der Aalst and ter Hofstede2005), FLOWer (version 3.0) (van der Aalst and Weske2005) and Declare (version 1.0) (Pesic et al.2007).

These works by Schonenberg et. al contributed to refine the scope of several change concepts and relationships of our Cmaps, namely *mechanism*, *property*, *operation* and *abstraction level*.

We could not find any related work on concept maps for process flexibility, or any kind of specific (sub)taxonomy of concepts related with the control of that flexibility. Nevertheless, the taxonomies presented here as related work are a mix between flexibility concept descriptions and flexibility classifications, where relationships between the concepts described/classified are not always clear. Also, these works emphasise on flexibility within process models and instances, disregarding the possibility of controlling flexibility within metamodel and real-world representations as well.

Regarding flexibility *metamodelling* and *modelling* related work, we analysed a set of prominent modelling languages and supporting tools. General workflow-based projects we analysed include ADEPT2 (Reichert et al.2009a), YAWL (van der Aalst and ter Hofstede2005) and *Declare* (Pesic et al.2007). These focus on the *enactment* of flexibility by supporting deviations, exceptions and inconsistencies, as also the migration of (executing) process instances.

Particularly, *Declare* is a constraint-based workflow system that provides for multiple declarative languages (e.g., *ConDec* (Pesic and van der Aalst 2006)). In fact, *Declare* is extensible and, without any programming, it can be configured to support additional constraint-based languages. Instead of explicitly defining the ordering of activities in models, *Declare* process models rely on constraints to implicitly determine the possible ordering of activities (any order that does not violate constraints is allowed).

YAWL is a PML based on the workflow patterns (van der Aalst et al.2003). The YAWL environment supports Worklets and Exlets, which allow for dynamic process flexibility, i.e., the possibility to defer the control of the flow of a process till runtime (worklets) and to define additional processes for handing exceptions occurred in the execution of a process.

ADEPT2 allows for changes to be made both at the *instance* and *model* abstraction levels of modelling. (Reichert et al. 2009a) call these changes *ad-hoc flexibility* and *process schema evolution*, respectively. At the instance level the ADEPT2 framework allows for ad-hoc deviations from the pre-modelled process schema (e.g., to insert, delete, or move activities). Regarding process model evolution, important goals were to provide the full spectrum of change operations for updating a process model, and to be able to migrate process instances (including those that were individually modified) to a new model version.

We also analysed specific software process modelling languages and tools, which included the following PSEEs: SLANG/SPADE (Bandinelli et al. 1994), Little JIL (Cass et al. 2000, Wise 2006) and PROMENADE (Balust and Franch 2002). All these have well specified metamodels, which can be changed automatically (such as in SPADE) or manually, to support process model improvement. They also support flexibility in the *modelling* of a software process by, for example, allowing its underspecification, along with late modelling and late binding mechanisms to defer its completion till execution. They do this through dynamic precedences between tasks in PROMENADE, or proactive control by allowing prescriptive (goal/artifact oriented) modelling in Little JIL.

Finally, we studied the EPFC and SPEARMINT EPG generators, which point their efforts exclusively to the *metamodelling* and *modelling* of software processes, although exporting executable process representations is supported by EPFC. Both works include a clear separation of concerns regarding the process engineer and the software development team tasks. Process engineers manage process model representations, while team members learn from and are guided by the generated (web) EPGs. Nevertheless, SPEARMINT strengths rely on a (customisable) model consistency checking engine, which allows process engineers to consider inconsistent and partially defined process models to be later completed upon the start of a software project.

EPFC implements SPEM as a widely accepted and used Model Driven Engineering (MDE)-based metamodel for software development processes. The UML ADs profile used as a PML to visually model these processes provides start up modelling advantages to process engineers and software team members. UML is a standard *de facto* in which they are naturally skilled by current practice. EPFC's flexibility support is entirely dedicated to the management of relationships that can be established between different process models. Contributions, extensions and adaptations/replacements can be made within process modelling elements, enhancing modularity and best practice reuse within a pool of distinct process models.

From what we could perceive from the analysed PMLs and supporting software tools, we found none that focused explicitly on the modelling of controlled flexibility in process models. In addition, no language or tool foresees the need for process engineers to specify *advices* on *where*, *which* and *how* changes can be made in process model and instance representations by software team members. Albeit, the modelling of preplanned extension points supported by some of these works is a possible approach to the challenges of modelling controlled flexibility.

## 7 Conclusions

The main goal of our FlexSPMF framework is to provide process engineers with means to control the changes that team members can make in software process models. It includes three main contributions: 1) concept maps to identify the knowledge domain of controlled flexibility; 2) a flexibility-aware metamodel that extends a core PML to enable the modelling of constraints of change onto process models; and 3) tool support for the modelling of those constraints, based on the SPEM metamodel, UML profile for ADs and EPFC tool customisations.

Concept maps help us to establish a sound knowledge base to enhance process engineers and software team members' understandability on process flexibility and how to control it. We provide no taxonomic classifications for the concepts. The intent is to develop a PML metamodel structure flexible enough to support variations of these concepts and corresponding relationships, in order to reflect software organisations' distinct needs of controlling flexibility within their software processes.

Our metamodel structure can be used as an extension to a PML, in order to provide a way to express controlled flexibility in software process models. This is achieved by associating constraints of change to process modelling elements, i.e., by *"decorating"* process elements with interchangeable *advices* on *operations*, *properties*, *mechanisms* and *agents of change*. It is also a non-intrusive metamodel extension, since it does not

modify the structure of the core PML metamodel to be extended, allowing for process engineers to model software processes with or without constraints of change.

We implemented this metamodel as an extension of SPEM: a software process metamodel which comprises UML ADs as its core PML. IBM's EPFC is an EPG generator which already implements SPEM. Therefore, we customised it to include the modelling elements of our metamodel structure. Modelling controlled flexibility with EPFC can be achieved by specifying constraints of change for the elements that compose a software process model.

Process engineers can use the customised FlexEPFC tool and its improved UML AD editor to make FlexUML profile applications, i.e., to apply onto process elements UML stereotypes that reflect how they wish to constrain changes. Alternatively, constraints of change can also be defined textually in a work breakdown view of a process model, by accessing a controlled flexibility property page of a process element and managing its constraints of change and corresponding tuple expressions.

From several prominent works analysed, none allowed for process engineers to control flexibility in software processes by modelling it onto their composing elements. Software process models, modelling languages and related tool support are commonly used among software engineering practitioners, as they increasing quality in delivered software products (Sommerville 2006). Representing this controlled flexibility through modelling elements (constraints of change) in software process models enhances communication between process engineers and software team members. Particularly, it allows process engineers to advise team members on *which*, *where* and *how* these models can be changed. With WebFlexEPFC, they can browse, learn and actually change the models, guided by the controlled flexibility information expressed through either activity diagram stereotype applications or textual constraint tuple expressions.

## References

Balust, J. M. R. and Franch, X. (2002). A precedence-based approach for proactive control in software process modelling. In *SEKE '02: Proceedings of the 14th international conference on Software engineering and knowledge engineering*, pages 457–464, New York, NY, USA. ACM Press.

Bandinelli, S., Fuggetta, A., Ghezzi, C., and Lavazza, L. (1994). SPADE: an environment for software process analysis, design, and enactment. In Finkelstein, A., Kramer, J., and Nuseibeh, B., editors, *Software process modelling and technology*. Research Studies Press Advanced Software Development, Taunton, UK, UK.

Beck, K. (1999). *Extreme Programming Explained: Embrace Change*. Addison-Wesley Professional, 1st edition.

Bider, I. (2005). Masking Flexibility Behind Rigidity: Notes on How Much Flexibility People are Willing to Cope With. In *Proceedings of the 17th International Conference on Advanced Information Systems Engineering (CAiSE'05)*, pages 7–8.

Borch, S. E. and Stefansen, C. (2006). On Controlled Flexibility. In *Proceedings of the 7th Workshop on Business Process Modeling, Development and Support (BPMDS'06) co-located with the 18th Conference on Advanced Information Systems Engineering (CAiSE'06)*, pages 121–126.

Cañas, A. J., Hill, G., Carff, R., Suri, N., Lott, J., Gómez, G., Eskridge, T. C., Arroyo, M., and Carvajal, R. (2004). Cmaptools: A knowledge modeling and sharing environment. In *Proceedings of the 1st International Conference on Concept Mapping*.

Cass, A. G., Lerner, B. S., Stanley M. Sutton, Jr., McCall, E. K., Wise, A., and Osterweil, L. J. (2000). Little-JIL/Juliette: A Process Definition Language and Interpreter. In *Proceedings of the 22nd International Conference on Software Engineering (ICSE'00)*, pages 754–757.

Cass, A. G. and Osterweil, L. J. (2005). Process Support to Help Novices Design Software Faster and Better. In *Proceedings of the 20th IEEE/ACM International Conference on Automated Software Engineering (ASE '05)*, pages 295–299.

Cugola, G. (1998). Tolerating Deviations in Process Support Systems via Flexible Enactment of Process Models. *IEEE Transactions on Software Engineering*, 24(11):982–1001.

Curtis, B., Kellner, M. I., and Over, J. (1992). Process Modeling. *Communications of the ACM*, 35(9):75–90.

Derbentseva, N., Safayeni, F., and Cañas, A. J. (2007). Concept maps: Experiments on dynamic thinking. *Journal of Research in Science Teaching*, 44:448–465.

Eclipse Foundation (2008). OpenUP. Wiki. Accessed on April 6th, 2009.

Gamma, E., Helm, R., Johnson, R., and Vlissides, J. (1995). *Design Patterns: Elements of Reusable Object-Oriented Software*. Addison-Wesley.

Hoffman, R. R. and Woods, D. D. (2000). Studying cognitive systems in context: Preface to the special section. *Journal of the Human Factors and Ergonomics Society*, 42:1–7.

Jacobson, I., Booch, G., and Rumbaugh, J. (1999). *The Unified Software Development Process*. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA.

Madhavji, N. H. (1992). Environment evolution: The prism model of changes. *IEEE Trans. Softw. Eng.*, 18(5):380–392.

Martinho, R., Domingos, D., and Varajão, J. (2007). FlexUML: A UML Profile for Flexible Process Modelling. In *Proceedings of the 19th International Conference of Software Engineering and Knowledge Engineering (SEKE'2007)*, pages 215–220.

Martinho, R., João Varajão, and Domingos, D. (2008). A two-step approach for modelling flexibility in software processes. In *Proceedings of the 23rd IEEE/ACM International Conference on Automated Software Engineering (ASE'2008)*.

Novak, J. D. and Cañas, A. J. (2008). 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. available at: http://cmap.ihmc.us/Publications/ResearchPapers/TheoryUnderlyingConceptMaps.pdf.

OMG (2005). Unified Modeling Language: Superstructure, version 2.0. Technical report, Object Management Group.

OMG (2008). Software & Systems Process Engineering Meta-Model. Specification v2.0, Object Management Group.

Pesic, M., Schonenberg, M. H., Sidorova, N., and van der Aalst, W. M. P. (2007). Constraint-based workflow models: Change made easy. In *Proceedings of the OTM Conference on Cooperative information Systems (CoopIS 2007)*, pages 77–94.

Pesic, M. and van der Aalst, W. M. P. (2006). A declarative approach for flexible business processes management. In *Proceedings of the Business Process Management Workshops, BPM 2006 International Workshops, BPD, BPI, ENEI, GPWW, DPM, semantics4ws*, pages 169–180.

Razmerita, L. and Lytras, M. D. (2008). Ontology-based user modelling personalization: Analyzing the requirements of a semantic learning portal. In *Proceedings of the 1st World Summit on The Knowledge Society (WSKS'08)*, pages 354–363.

Regev, G., Soffer, P., and Schmidt, R. (2006). Taxonomy of Flexibility in Business Processes. Input to the 7th Workshop on Business Process Modeling, Development and Support (BPMDS'06) co-located with the 18th Conference on Advanced Information Systems Engineering (CAiSE'06), Website. Available at: http://lamswww.epfl.ch/conference/bpmds06/taxbpflex.

Reichert, M., Dadam, P., Rinderle-Ma, S., Jurisch, M., Kreher, U., and Göeser, K. (2009a). Architecural principles and components of adaptive process management technology. In *PRIMIUM - Process Innovation for Enterprise Software. Lecture Notes in Informatics (LNI)*. Koellen-Verlag.

Reichert, M., Rinderle, S., and Dadam, P. (2003). Adept workflow management system. In *Proceedings of the 1st International Conference on Business Process Management (BPM'03)*, pages 370–379.

Reichert, M., Rinderle-Ma, S., and Dadam, P. (2009b). Flexibility in process-aware information systems. *LNCS Transactions on Petri Nets and Other Models of Concurrency (ToPNoC), Special Issue on Concurrency in Process-aware Information Systems*, 2:115–135.

Ross, A. M., Rhodes, D. H., and Hastings, D. E. (2008). Defining changeability: Reconciling flexibility, adaptability, scalability, modifiability, and robustness for maintaining system lifecycle value. *Systems Engineering*, 11(3):246–262.

Schonenberg, H., Mans, R., Russell, N., Mulyar, N., and van der Aalst, W. M. P. (2007). Towards a taxonomy of process flexibility (extended version). BPM Center Report BPM-07-11, BPMcenter.org.

Schonenberg, H., Mans, R., Russell, N., Mulyar, N., and van der Aalst, W. M. P. (2008). Process flexibility: A survey of contemporary approaches. In *Proceedings of Advances in Enterprise Engineering I, 4th International Workshop CIAO! and 4th International Workshop EOMAS, held at CAiSE 2008*, pages 16–30.

Schwaber, K. and Beedle, M. (2001). *Agile Software Development with Scrum*. Prentice Hall PTR.

Sommerville, I. (2006). *Software Engineering, Eighth Edition*. Addison-Wesley.

van der Aalst, W. M. P., Hofstede, A. H. M. T., Kiepuszewski, B., and Barros, A. P. (2003). Workflow patterns. *Distributed Parallel Databases*, 14(1):5–51.

van der Aalst, W. M. P. and ter Hofstede, A. H. M. (2005). Yawl: yet another workflow language. *Information Systems*, 30(4):245–275.

van der Aalst, W. M. P. and Weske, M. (2005). Case handling: a new paradigm for business process support. *Data and Knowledge Engineering*, 53(2):129–162.

Wise, A. (2006). Little-jil 1.5 language report. Technical Report UM-CS-2006-51, Department of Computer Science, University of Massachusetts, Amherst, MA 01003.

## Notes

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

2 http://www.eclipse.org/swt