Goals and Requirements for Supporting Controlled Flexibility in Software Processes

Ricardo Martinho*
Department of Informatics Engineering
School of Technology and Management
Polytechnic Institute of Leiria
Campus 2, Morro do Lena - Alto do Vieiro
2411-901 Leiria, Portugal
E-mail: rmartin@estg.ipleiria.pt
**Corresponding author*

Dulce Domingos
Department of Informatics
Faculty of Sciences
University of Lisboa
Edifício C6, Piso 3, Campo Grande
1749-016 Lisboa, Portugal
E-mail:dulce@di.fc.ul.pt

João Varajão
Centro ALGORITMI
Department of Engineering
University of Trás-os-Montes e Alto Douro
5001-801 Vila Real, Portugal
E-mail: jvarajao@utad.pt

ABSTRACT

Software processes are dynamic entities that are often changed and evolved by software development team members. Consequently, flexibility is one of the most important features within software processes and related tools. However, in the everyday practice, team members do not wish for total flexibility. They rather prefer to learn about and follow *controlled flexibility advice*, i.e., previously defined information on *which*, *where*, *how* and *by whom* they can change software process representations to match real-world situations. In this paper, we define a set of goals and requirements for a language and supporting software tool to control the flexibility within software processes. They follow a two-step approach, where 1) process engineers use the language constructs and supporting tool to define controlled flexibility-related information within software process models, and 2) software tem members browse and learn from this information, and perform changes accordingly.

Keywords: goal, requirement, software, process, model, controlled flexibility, language

INTRODUCTION

Software processes represent a specifically ordered and organised set of the elements and relationships which are involved in the development of software products. The main elements that compose such processes are *activities*, *agents*, *artifacts*, *roles* and production support *tools*.

Descriptions of these processes can be captured and converted into process models. Process engineers usually recur to a Process Modelling Language (PML) to define and represent these models. They facilitate human understanding and communication, and provide guidance for software development team members when executing the process.

Throughout the last three decades, several PMLs and supporting Process-centred Software Engineering Environments were developed to elicit process models and automate their support (see, e.g., Bandinelli et al. (1994), Wise (2006)). However, and as opposed to many stable business processes (such as production line-based ones), software processes are commonly held as dynamic entities, which often must be modified and evolved to cope with changes occurred, for instance, in the requirements of a certain software product, in the software organisation's structure or in the rapid changing software market (Cugola, 1998). Software process *flexibility* refers, precisely, to the ability to change parts of a process, without completely replacing it (Soffer, 2005).

However, more recent research advocate that, 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). In fact, latest case studies on flexibility in software processes (see, e.g., Cass and Osterweil (2005)) make evidence on the need of having software process engineers expressing and controlling the amount of changes that the remaining team members can or cannot make in the software process.

This *controlled flexibility* can be defined as *the ability to control the way changes are to be performed, taking into account*:

- *Which* process elements, as for example choosing which activities, roles or work products can or cannot be changed;
- At which abstraction level(s) of modelling (*where*) can or cannot those elements be subjected to changes. For example, the process engineer can require that changes made to the model representation of a *Test Solution* activity should be immediately reflected to all the software project plans (*instance* level) and real-world projects (*real-world* level) where this activity is referred;
- What are the dimensions of change involved (*how*), including, for example, which *operations* can or cannot be performed (such as *add*, *delete*, *move* or *skip*) and which *properties* will the change enclose (such as its *duration* and *extent*);
- *Who* can or cannot enforce those changes, including, for example, *single users* or *role-based* permissions.

In this paper we present a set of goals and requirements regarding a modelling language and proper software tool support to enable the representation of controlled flexibility information in software processes. This requires an in-depth understanding of software development social organisations, their work, and the ways cooperation and learning are enforced. Therefore, each derived goal and corresponding requirement(s) is supported by a set of needs and assumptions identified by important works from empirical software and knowledge engineering research areas.

This paper is organised as follows: the next section presents the research process we adopted for the development of a *controlled flexibility*-aware language and supporting software tool. It

comprises the goal and requirements' definition activities and related specifications. The third section contains these specifications, as well as thorough reviews and justifications for the viewpoints expressed by each goal and requirement. Then, we present most prominent related work, and finally we conclude the paper and present future work.

THE PROCESS OF ELICITING GOALS AND REQUIREMENTS

We adopted the research process in Figure 1 to conduct the main activities involved in the analysis and development of a *controlled flexibility*-aware language and proper software tool support. It is an iterative and incremental process, with seven main activities. These activities sometimes overlapped, and feedback flows occurred between them, to foresee adjustments and consistency checks between their resulting work products.

For instance, the definition of goals was interleaved with the specification of requirements, which in turn alternated forwards and backwards with language and software tool specification, implementation and evaluation activities. Therefore, the work products which resulted from the process were incrementally refined, until a satisfactory version was reached. The process illustrates two activities and two related work products in grey-shade. These refer to the goal and requirement's elicitation activities and related specifications, which delimit the scope of this paper.

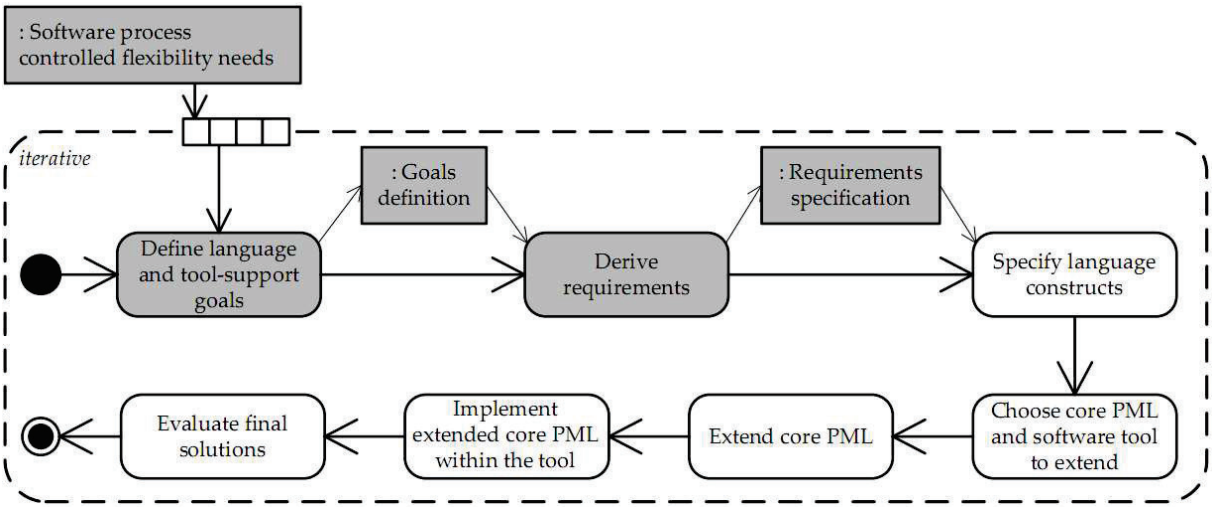

Figure 1. The research process adopted, including goals and requirements' definition activities.

In the area of requirements engineering, a *goal* represents an objective that the system under consideration should or must achieve. Goals can be defined on different levels of abstraction, ranging from high-level business goals to low-level technical concerns. Moreover, goals may be used to represent functional as well as non-functional aspects, such as *performance* or *usability*. They are well-suited to be applied in combination with scenarios or viewpoints in order to elicit and define requirements, and to drive a requirements engineering process (Alspaugh and Antón, 2008).

In our context, goals justify the development of a *controlled flexibility*-aware language and proper software tool support, and points out their intended scope. The strategy adopted to elicit them is a simpler version of the one referred in the prominent work of van Lamsweerde (2008). It includes the following three steps:

1. Define a hierarchy of *goal descriptions* and concerned *objects*, regarding the *controlled flexibility*-aware language and its supporting software tool;
2. Identify the *agents* (human agent, physical device, program) and the *actions* they are capable of performing on the objects involved in goal descriptions;
3. *Assign* actions to responsible agents.

With the goal specification as input, we proceed by defining the requirements, which provide more technical and system-oriented descriptions of what the language and software tool should provide to its users. For more complex problems, the main goal is divided into subgoals. More specific requirements are then associated to these subgoals.

GOAL DEFINITIONS AND RATIONALE

The next goals define our contributions for the contents of the language, what the language should do, and how a software tool should support it. The general goal that determines the high level scope is defined as follows:

> ***G****. Provide a language and proper tool support for process engineers to express controlled flexibility-related information within software processes, and for software team members to be guided by that information when performing changes to the processes.*

This goal rests on well supported assumptions, namely:

- Software processes are dynamic entities that often evolve to cope with changes in real world situations (Cugola, 1998, Fuggetta, 2000, Cass and Osterweil, 2005)
- For this kind of dynamic and changeable processes, it should be possible, through the use of software supporting tools, to quickly implement new processes, to enable *on-the-fly* adaptations of those already running, to defer decisions regarding the exact process logic to runtime, and to evolve implemented processes over time (Schonenberg et al., 2008, Reichert et al., 2009)

The next subgoals assume software process representations (models and instances) and also PMLs as the preferred choices for supporting our main goal:

> ***G1****. Specify a Process Modelling Language (PML) to express controlled-flexibility information;*
>
> ***G2****. Adopt software process representations and supporting tools, as the media for learning and sharing knowledge on controlled flexibility information between process engineers and software team members.*

These two goals follow long time acknowledged facts about software process models and the activity of *process modelling*, starting with the one that states *"to model is to understand"* (McGowan and Bohner, 1993). Indeed, process models facilitate human understanding, support process management and change, and provide foundations for process guidance and execution automation (Curtis et al., 1992, Fuggetta, 2000, Sommerville, 2006). Process models are built using PMLs, which can enhance several process perspectives, such as the functional, behavioural, organisational and the informational ones (Curtis et al., 1992). Similarly, *controlled flexibility* can easily become one of these process perspectives, and take similar advantages within software process model and instance representations.

Based on current research and practice of modelling languages and supporting software tools, the following subgoals unroll goal ***G1*** by presenting concrete choices made to develop our *controlled flexibility*-aware PML:

> ***G1.1***. *Provide a language that complements existing (*non change-*controlled) process representations with information on the way changes to those representations can or cannot be performed;*
>
> ***G1.2***. *Support user adaptability on the definition of language concepts and derived constructs.*

Regarding ***G1.1***, the way that it is defined assumes a language design choice, made between the following options:

1. Develop a whole new PML; or
2. Adopt an existing and sufficiently prominent PML and, upon availability, extend it with *controlled flexibility*-aware language constructs.

We dropped the first option for considering it unnecessary. Much like other non-functional aspects of software process representations (such as *usability*, *security* or *validity*), the modelling of controlled flexibility only makes sense if applied to *existing* process elements (such as *activities*). Therefore, *controlled flexibility* can be assumed as an additional aspect of the core PML that process engineers and software team members use on a daily basis to create and customise software process representations.

Goal ***G1.2*** refers the support for user adaptability on the way controlled flexibility is defined, regarding the language concepts and related constructs. In spite of many efforts (see, e.g., Weber et al. (2008), Schonenberg et al., (2008)), there is not a sufficiently widely adopted taxonomy or conceptual framework on process flexibility. Therefore, the PML metamodel architecture should provide easy incorporation of new concepts, or the ability of changing existing ones, in order to allow software organisations to define their own perspectives on controlled flexibility.

We unroll goal ***G2*** into the next subgoals, to specify how process model and/or instance representations should enable learning, knowledge transferring and enacting of controlled flexibility:

> ***G2.1***. *Adopt an appropriate modelling approach to enhance controlled flexibility modelling within process model and/or instance representations;*
>
> ***G2.2***. *Provide additional and updated controlled-flexibility information on published process models, in order to allow software team members to take informed decisions when changing these models and/or related instances;*
>
> ***G2.3***. *Allow software team members to change published process model and/or instance representations, based on the information provided about controlled flexibility.*

These three subgoals reflect the changeability nature of software and supporting process models, which justify the use of modelling approaches that promote evolving, incomplete and semi-formal process representations. These modelling approaches are also needed to control and reflect the changes made in real-world situations (***G2.2***). This opposes to rigid and pure formal

approaches that do not allow for process deviation, inconsistency tolerance, exception handling and late modelling aspects that often occur when developing software (Fuggetta, 2000).

Also, the unusually strong modelling skills of software team members are related to the nature of software development (Cass and Osterweil, 2005). Here, models are frequently used, often several models for each piece of software, as also programming languages, which have similarities to the languages used to develop software process models (Osterweil, 1987).

Additionally, those who perform the work should be involved in modelling it. In our context, we consider *process engineers* as being responsible for modelling *which*, *where*, *how* and *by whom* software process element representations can or cannot be created, changed/evolved or deleted by team members, within a process model and/or associated instances. This means that a process engineer can, for example, delegate to more skilful team members the modelling of an underspecified part of a process model (***G2.3***).

For this, it is important that team members, using a software tool, gain access to published and updated process model and/or instance representations with *controlled flexibility* information (***G2.2***). For example, the use of software tools that promote interactive and distributed models plays a main role in quickly updating process representations upon changes (***G2.3***). They also provide software team members not only enhanced (concurrent) modelling capabilities but also immediate knowledge transference about changes occurred (Turetken and Demirors, 2008).

These goals constitute the input to the *Derive requirements* activity in the research process of Figure 1, and all resulting requirements must relate to one or more goal(s). The next two sections define the language and proper software tool requirements.

REQUIREMENTS FOR A CONTROLLED FLEXIBILITY-AWARE LANGUAGE

Each of the goals and subgoals derives into one or more associated requirements. These emphasise language and software tool related features. We begin with those which particularly refer to the *controlled flexibility*-aware language. We also include, in front of each requirement definition, an abbreviated form of the associated (sub)goal(s). We start by the following two general requirements:

> ***LR1***. *Develop language constructs to model the way changes to process element representations are advised to be made (****G1***. Define PML to express controlled flexibility information*);*
>
> ***LR2***. *Support possible adaptation/evolution needs regarding the definition of the language constructs and related concepts (****G1.2***. Support user adaptability*).*

The first requirement establishes the scope of the language. It includes constructs to define controlled changes and associating them within process representations. To clarify these features, we divide it in the following sub-requirements:

> ***LR1.1***. *Develop language constructs that a process engineer can use to specify constraints of change (****G1.1***. Provide a language that complements existing (non change-controlled) process representations*);*
>
> ***LR1.2***. *Develop language representations that are able to inform agents of change (software team members) about the constraints of change associated with process element representations (****G2.2***. Provide additional and updated controlled-flexibility information on published process models*).*

Requirement ***LR1.1*** provides further insight of the language, by referring the language constructs to be used by a process engineer. It is based on the concept of *constraint of change*, included in a conceptual framework proposed in Martinho et al. (2009a). A *constraint of change* defines that changes to process representations can be controlled by imposing them one or more constraints. These constraints contain tuple-based expressions with a variable number of parts. As a whole, expressions contain *advice* on the *abstraction levels*, *properties*, *operations* and *agents of change*, which describe the kind of change that can or cannot be made to the associated process element. For example, if a process engineer wants to express that *skipping instances of a* Test Solution *activity is denied*, s/he can define a constraint of change associated to this activity model representation, with the following tuple expression:

{tbAdvice=denied, level=instance, operation=skip},

where *denied* describes the advice of change to follow, regarding the *instance* abstraction level and the *skip* operation.

The basis for this kind of *constraint-based* modelling is associated with the approaches and language paradigms usually adopted for modelling software processes. The resulting models tend to be more *descriptive* and less *prescriptive*, i.e., they tell software team members *how software is actually (or has been) developed*, rather than *how software should be developed. Descriptive* PMLs can enforce flexibility through the use of constraint-based modelling (see, e.g., the *ConDec* PML in Pesic et al. (2007)).

However, there is a distinction between developing language constructs to be used by a process engineer (***LR1.1***), and representing them to be understood by software team members, who are in fact going to perform changes to process representations (***LR1.2***).

Still, requirement ***LR1.1*** can be decomposed into:

> ***LR1.1.1****. Provide a language for modelling constraints of change that, upon availability, uses the same modelling features than those used to model any other process perspective with the adopted PML (****G2.1***. Adopt an appropriate modelling approach to enhance controlled flexibility modelling*);*
>
> ***LR1.1.2****. Support variable degrees of detail in the definition of a constraint of change (****G1.1***. Provide a language that complements existing process representations*,* ***G1.2***. Support user adaptability*);*
>
> ***LR1.1.3****. Support different levels of complexity regarding the definition of constraints of change for a certain process element (****G1.1***. Provide a language that complements existing process representations, ***G1.2***. Support user adaptability*).*

The first of these sub-requirements explicitly refers to the desirable compliance between our *controlled flexibility*-aware language and the existing core PML (to be extended), which is already used to model other process perspectives. For instance, let us suppose that a process engineer wants to model the *access control* perspective of an *Activity* type process element. This is accomplished by using several core PML functional and non-functional features such as information hiding, multiple views, textual and graphical modelling and decomposition mechanisms. Requirement ***LR1.1.1*** proposes that, when modelling *constraints of change* within

a process element model, a process engineer should have similar modelling features at disposal. This would also reduce the learning curve about the use of our *controlled flexibility*-aware part of the language by process engineers. Hence, this requirement is dependable on the adopted core PML and its corresponding modelling features.

LR1.1.2 describes the need of having different degrees of detail, when defining constraints of change. This is also related with the user adaptability goal (***G1.2***) since organisations have different needs, regarding the detail with which they want to control flexibility. As mentioned above, a constraint of change can be expressed by a tuple expression comprising variable number of tuple parts, ranging from a single (and mandatory) *advice* part, to a fully filled tuple with all possible tuple parts. For example, if the aforementioned *Test solution* activity had a constraint of change with a single *{tbAdvice=denied}* part expression, it would mean that a process engineer advises software team members *not to change* the activity in any way. However, when associated with a constraint defined by the tuple expression above (*{tbAdvice=denied, level=instance, operation=skip*} the same activity has a more precise information on controlled flexibility.

Requirement ***LR1.1.3*** requires the language to support multiple *constraints of change* for a certain process element. The language should support a process element which has a large, and possibly conflicting set of constraints, and another process element with a smaller set that requires less effort to define.

The requirement concerning the development of process representations for *constraints of change* (***LR1.2***) has a different but correlated purpose to requirement ***LR1.1***. We clearly aim to separate the two main roles of *process engineer* and *software team member*. The former should configure the *constraints of change* when defining process models, while the latter should be able to change the models. For this, team members must be aware of the *constraints of change* associated to a certain process element, by visualising its textual and/or graphical representations. Requirement ***LR1.2*** can be more detailed into the following sub-requirement:

> ***LR1.2.1.*** *Support textual and/or graphical representations for constraints of change compatible with the overall process representation standards used by the adopted core PML (****G1.1***. Provide a language that complements existing process representations*).*

Similarly to ***LR1.1.1***, this sub-requirement aims at a better integration of our language with the adopted core PML. For instance, if this core PML is *Little JIL* (Cass et al., 2000), then representing *constraints of change* will probably follow the same *step badge*-type representation which characterises the PML, i.e., constraints of change can be, for example, represented within an additional step badge called *Controlled Flexibility*, just like other step badges such as the *Exception Handler* one.

The following sub-requirements expand requirement ***LR2***, explicitly to define desirable extensibility and modularity properties shared between the adopted core PML and our language:

> ***LR2.1****. The adopted core PML should provide an extensible metamodel, in order to incorporate the language's new concepts on controlled flexibility (****G1.1***. Provide a language that complements existing process representations*);*
>
> ***LR2.2****. The controlled flexibility language metamodel to be coupled with the core PML's metamodel should enhance easy adaptation/evolution regarding its own concepts and relationships (****G1.2***.* Support user adaptability*);*

***LR2.3**. The resulting extended PML should maintain the possibility of defining software process models without controlled flexibility definitions (**G1.1**.* Provide a language that complements existing process representations*).*

Goal ***G1.1*** clearly assumes the position of adopting a core PML to be extended with our *controlled flexibility*-aware language constructs. Therefore, we must have access to the core PML's metamodel, in order to extend it ***LR2.1***. Additionally, the metamodel part of our language must also provide adaptability concerning the concepts and relationships considered in the language's domain. Since these can vary and suffer changes/improvements (due to distinct organisation cultures and application scenarios), it is desirable to have also a flexible metamodel structure (***LR2.2***), which allows for the definition of process models with or without controlled flexibility (***LR2.3***).

These metamodel structure-related language requirements finalise our requirement specification for our *controlled flexibility*-aware language. We proceed, in the next section, with the definition of the requirements for a supporting process-aware software tool.

SOFTWARE TOOL REQUIREMENTS

The features presented by PMLs gain relevance through their software tool support. For example, process modelling complexity management by hiding information and having multiple views/perspectives on process models are some (non-functional) language requirements that can be greatly enhanced with proper software tool support. In fact, the software process modelling experience of a process engineer nowadays is built up on a compromise between the expressive power of the PML constructs, and the efficiency of their software tool support, making it difficult to dissociate them (Gruhn, 2002).

Based on the language and software tool goals presented above, it is necessary to guarantee that the tool will accomplish the following three main requirements:

***TR1**. Provide process engineers with tool support to manage the constraints of change associated with a certain software process element representation (**G2.1**.* Adopt an appropriate modelling approach to enhance controlled flexibility modelling within process representations*);*

***TR2**. Provide process engineers with tool support to publish process models with information on their constraints of change (**G2.2**.* Provide additional and updated controlled flexibility information on published process models*);*

***TR3**. Provide software team members with tool support to enable the changing of software process element representations according with the associated constraints of change (**G2.3**.* Allow software team members to change published process representations*).*

These three requirements define the main functional features to be supported by the adopted software tool. The first requirement can be further decomposed into:

***TR1.1**. Process engineers should be able to create, read, update and delete constraints of change associated with a certain process element representation (G2.1.* Adopt appropriate modelling approach*);*

TR1.2 *Provide tool support for the syntax and semantic analysis on constraints of change (G2.1.* Adopt appropriate modelling approach*).*

Sub-requirement ***TR1.1*** refers to the basic operations regarding constraints of change and their components. Process engineers should be able to use the software tool to create new constraints, and to read, update and delete existing ones, regarding a certain process element.

The second sub-requirement (***TR1.2***) addresses a fairly common requirement for PSEEs and associated PMLs, which is syntax and semantic analysis on the resulting process models. This includes deciding on the tool behaviour regarding check invocation on the definitions of constraints of change (for instance, automatic or manual invocation), and tool behaviour on the violation of correctness, compliance and consistency rules. For instance, a tool may disallow the violation of correctness on constraints of change by denying exiting from a User Interface (UI) window where constraints are edited, when a certain constraint is wrongly defined. However, it can only display warnings, when compliance and/or consistency rules are violated.

Requirement ***TR2*** refers to the possibility of enabling process engineers to publish, through features of the software tool, process models with information on controlled flexibility. These are to be consulted and updated by software team members, and used as their preferred vehicle of communication to guide the everyday software development practice. For this, published software process models must be interactive, i.e., allow changes and immediate updates by software team members. Therefore, the software tool support is needed to:

TR2.1*. Generate a distributed application to support interactive software process models with information on constraints of change (****G.2.2***. Provide additional and updated controlled-flexibility information on published process models*);*

TR2.2*. Store the content of the published versions of process representations, regarding their controlled flexibility perspective (****G2.2***. Provide additional and updated controlled-flexibility information on published process models*);*

TR2.3*. Provide propagation for updates made between published and stored process representations (****G2.2***. Provide additional and updated controlled-flexibility information on published process models*).*

These requirements rest on the distributed nature that software processes have, i.e., their management, modelling, enacting, monitoring and improvement activities are normally decentralised, carried out by more than a single team, and at more than a single site (Gruhn, 2002). Regarding the *modelling* activities, it is common for a software process engineer to delegate to more skilful team members the modelling of an underspecified part of a software process. Therefore, it is crucial that team members have access to published and updated process model and/or instance representations. For example, recent advances in client side web interaction technologies (such as Asynchronous JavaScript and XML (AJAX)) have enabled the use of interactive web clients and, specifically, interactive process editors. These can provide software team members not only enhanced distributed and concurrent process modelling and storage (***TR2.2***), but also immediate updating of changes made in process models (***TR2.3***) (Turetken and Demirors, 2008).

Regarding the possibility of software team members using the software tool to change published process models (***TR3***), we propose the following sub-requirements:

***TR3.1**. Provide tool support that enables visualising information on software process models about the constraints of change associated with the composing process elements (G2.2. Provide additional and updated controlled-flexibility information on published process models);*

***TR3.2**. Provide tool support to enable the changing and the guiding of software team members in the action of changing a process element representation according with the predefined constraints of change (G2.3. Allow software team members to change published process representations);*

***TR3.3**. Update the published process models upon any change committed by a software team member (G2.2. Provide additional and updated controlled-flexibility information on published process models, G2.3. Allow software team members to change published process representations).*

These sub-requirements address simple but essential features that the software tool must provide to software team members, when changing published process representations. They must be able to visualise the constraints of change associated to process element representations through a software tool that provides a representation of the published process model (***TR3.1***). Next, they must be able to make changes according to those constraints (***TR3.2***). Here, we are assuming that the software tool provides, at start, full changeability to all process representations, and that it additionally provides some guidance features when a software team member changes the process representation, according with the predefined constraints of change.

Finally, when a software team member commits a change onto a published process model, the software tool must update it in order to provide software team members with its latest version (***TR3.3***).

Together with the previously defined language requirements, these software tool requirements provide the necessary input for the specification, implementation and testing activities of our research process illustrated in Figure 1. The next section proceeds with an analysis of related work, particularly for PMLs and related software tools which support flexibility in their processes.

RELATED WORK

The goals and requirements elicited in the previous sections are related with some of the process *flexibility* features presented by most prominent and current PMLs and supporting software tools. We selected ten of these products, including academia and industry research initiatives, which support process flexibility in some manner. Since our scope includes controlling flexibility through all levels of abstraction (*metamodel*, *model*, *instance* and *real-world* process representations) we divide the selected products in two groups:

1. *General process languages and tools* - can be used for general purpose Business Process Management (BPM) (thus, also for software process management). Academia PMLs and related tools analysed include ADEPT2 (Reichert et al., 2009), YAWL (van der Aalst and ter Hofstede, 2005) and *Declare* (Pesic et al., 2007). We also included, as commercial products, TIBCO's *iProcess Suite* (TIBCO Software Inc., 2009);

2. *Software process-specific languages and tools* - are used solely for managing software processes. We separate these in two additional categories:
    a. *PSEEs* - which focus on supporting enactment and monitoring of software process representations, and often provide integration abilities with akin EPGs. Here, we chose SLANG/SPADE (Bandinelli et al., 1994), *Little JIL* (Cass et al., 2000, Wise, 2006) and PROMENADE (Balust and Franch, 2002), as reference academia works. We also included the commercial product IBM Rational *Team Concert* (IBM Rational Software, 2009);
    b. *EPG-generators* - which focus essentially on guiding process engineers and software team members through the modelling and use of visual and interactive representations of software process models. We included academia SPEARMINT (Becker-Kornstaedt et al., 1999) and Eclipse's EPFC open-source project (Haumer, 2006).

From the analyses performed, we concluded that the support for (controlled) flexibility is varied, and strongly dependant on the objectives and audiences of the selected products. For example, the workflow-related ADEPT2 and YAWL projects focus on the enactment of flexibility by supporting deviations, exceptions and inconsistencies between process models and related instances, as also the migration of (executing) process instances to match changed process models. The *Declare* project goes further by providing a declarative, constraint-based modelling approach, which allows for additional flexibility-related features. For instance, process engineers can predefine alternative control flow paths within a process model, and defer till runtime the choice for one of these paths. Although in a different way, this anticipation planning can be considered as a kind of constraints of change, which inform process engineers about the conditions necessary to be met for a certain path to be followed within a process.

For the PSEEs chosen, approaches to flexibility are centred on the *metamodelling*, *modelling* and *improvement* activities surrounding process models. All works analysed 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 descriptive (goal/artifact oriented) modelling in *Little JIL*.

EPG generators 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 process engineer and software development team member's tasks. Process engineers manage process model representations, while team members learn from and are guided by the generated (web) EPG. 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 (OMG, 2008) as a widely accepted metamodel for well-known software development processes such as the Rational Unified Process (RUP), XP and *Scrum*. The UML-based language used 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.

As for the commercial products analysed, *iProcess Suite* is a general purpose workflow-based product which also supports late modelling and late binding within the modelling of (business) processes. However, changes made in process instances are only supported through *anticipation*, i.e., flows of execution can be chosen in runtime through pre-planned points of choice modelled by the process engineer. Similarly, *Team Concert* also foresees flexibility at the instance level of modelling by including pre-planned extension points in a process instance representation (called a *project area*).

From what we could perceive from these analysed PMLs and supporting software tools, we found none that focused explicitly how to express *controlled flexibility*, and how to support it within a process-aware tool. No language or tool foresees the need for process engineers to specify *advices* on *which*, *where*, *how* and *by whom* changes can be made in process model and instance representations. Nevertheless, the modelling of pre-planned extension points supported by some of these products is a possible approach to the modelling of controlled flexibility. However, the formal nature of the modelling languages that allow these points forces the process engineer to anticipate, at least, *where* and *how* process instance executions can deviate from the control flow predefined in the process model.

CONCLUSIONS AND FUTURE WORK

The goals and associated requirements derived throughout this paper delimit the scope of a language and software tool support which can be used to control the flexibility in software processes. We began by adopting a strategy for the goal definition activities, which include goal descriptions that refer to the *objects*, *actions* and *agents* involved. Objects identified in these goals include: *(published) process representations and supporting medium*; *language constructs* and *controlled flexibility information*. Actions on these objects included *modelling*, *publishing*, *updating*, *adapting* and *changing*. These are performed by the following agents: *process engineers* (modelling, publishing); *software team members* (adapting and changing), and the *software tool* to develop, which does the *publishing* part of generating the medium support for process representations, the *updating* part related with distributed software process modelling, and the *adapting* part of correlating distinct process element representations into integrated process views.

Next, the associated requirements were divided into *language* and *software* tool requirements. Briefly, they define an approach for controlling flexibility in software processes which comprises the following two steps (Martinho et al., 2008):

Step 1. *The process engineer defines controlled flexibility information for process elements* – by associating a (variable) number of constraints of change to process elements' model representations;

Step 2. *Software team members change process elements accordingly – b*y accessing the published version of process models, and performing informed changing actions

to the published process model itself, or to other process representation of any abstraction level (*metamodel*, *instance* or *real-world* representations).

We have already implemented prototype solutions regarding the requirements specified in this paper. They include the *Controlled Flexibility Language* (CFL) specification and the *FlexEPFC* supporting software tool. The former is defined by an object-oriented metamodel structure which comprises *constraint of change*, *expression* and *expression part* elements. It also includes well-formedness rules to address correctness violations such as having constraint expressions with no parts, or having conflicting constraints associated to the same process element. To provide *proof-of-concept* applicability for these language constructs, we adopted UML as the core PML, and extended it with a UML profile called *FlexUML* (Martinho et al., 2007). Then, we implemented this profile within the open-source EPFC software tool. Besides extending EPFC's UML-based core PML with *FlexUML*, we customised it to provide the necessary UI components to manage the application of *FlexUML*'s stereotypes and attributes, within the process elements supported by EPFC.

Figure 2 presents a snapshot of the customised *FlexEPFC*. We can observe an activity diagram editor loaded with the Elaboration phase of an UP-based software process model. The Test Solution activity has a constraint of change associated named *denySkip*. It represents a normal UML *Constraint* element, to which the process engineer applied the «COfChange» stereotype from *FlexUML*. The attributes of this stereotype can be configured below in the *Stereotypes* section pane. A «COfChange» is mainly composed by an *expression*, to which can be applied one or more expression stereotypes, also from *FlexUML.* Figure 2 shows the stereotypes needed to compose the expression *{tbAdvice=denied, level=instance, operation=skip}*, already mentioned above. More details on these developments can be found in Martinho et al. (2009b).

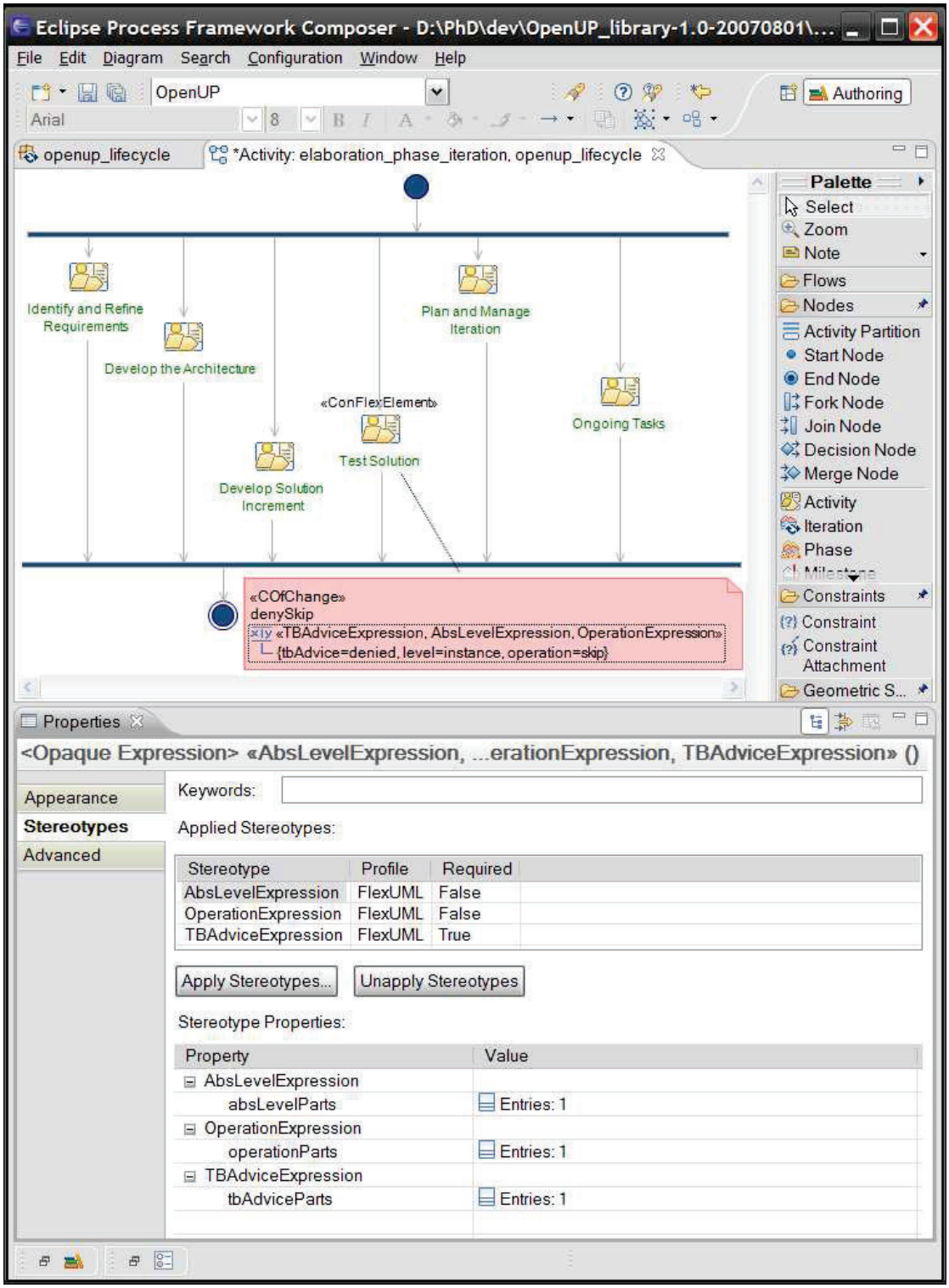

Figure 2. FlexEPFC *software tool and* FlexUML *profile applications within a* Test Solution *activity.*

Future work includes improving tool support and documentation, and performing additional iterations between the requirements presented above and the related implementations. This is to be accomplished by realising structured case studies' analysis on the use of our customised *FlexEPFC* tool in real-world software organisations and people. Only by enlarging our test audience regarding *FlexEPFC* and the underlying CFL language constructs can we get the necessary data to evaluate and evolve our current implementations.

REFERENCES

Alspaugh, T. A. and Antón, A. I. (2008). Scenario support for effective requirements. *Information and Software Technology*, 50(3):198–220.

Balust, J. M. R. and Franch, X. (2002). A precedence-based approach for proactive control in software process modelling. In Tortora, G. and Chang, S.-K., editors, *Proceedings of the 14th International Conference on Software Engineering and Knowledge Engineering (SEKE'2002)*, volume 27, 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*, volume 1, chapter 9, pages 223–247. Research Studies Press Ltd., Taunton, UK.

Becker-Kornstaedt, U., Hamann, D., Kempkens, R., Rösch, P., Verlage, M., Webby, R., and Zettel, J. (1999). Support for the Process Engineer: The Spearmint Approach to Software Process Definition and Process Guidance. In Jarke, M. and Oberweis, A., editors, *Proceedings of the 11th International Conference on Advanced Information Systems Engineering (CAISE'99)*, volume 1626 of *Lecture Notes In Computer Science*, pages 119–133, London, UK. Springer-Verlag.

Bider, I. (2005). Masking Flexibility Behind Rigidity: Notes on How Much Flexibility People are Willing to Cope With. In Pastor, O. and e Cunha, J. F., editors, *Proceedings of the 17th International Conference on Advanced Information Systems Engineering (CAiSE'05)*, volume 3520 of *Lecture Notes in Computer Science*, pages 7–8, London, UK. Springer-Verlag.

Borch, S. E. and Stefansen, C. (2006). On Controlled Flexibility. In Dubois, E. and Pohl, K., editors, *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)*, volume 4001 of *Lecture Notes in Computer Science*, pages 121–126, London, UK. Springer-Verlag.

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 Ghezzi, C., Jazayeri, M., and Wolf, A. L., editors, *Proceedings of the 22nd International Conference on Software Engineering (ICSE'00)*, pages 754–757, Los Alamitos, CA, USA. IEEE Computer Society.

Cass, A. G. and Osterweil, L. J. (2005). Process Support to Help Novices Design Software Faster and Better. In Redmiles, D. F., Ellman, T., and Zisman, A., editors, *Proceedings of the 20th IEEE/ACM International Conference on Automated Software Engineering (ASE '05)*, pages 295–299, New York, NY, USA. ACM Press.

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.

Fuggetta, A. (2000). Software process: a roadmap. In Finkelstein, A., editor, *Proceedings of the Conference on The Future of Software Engineering, co-located with the 22nd International Conference on Software Engineering (ICSE'00)*, volume 14, pages 25–34, Red Bank, NJ, USA. J. C. Baltzer AG, Science Publishers.

Gruhn, V. (2002). Process-centered software engineering environments, a brief history and future challenges. *Ann. Softw. Eng.*, 14(1-4):363–382.

Haumer, P. (2006). Increasing Development Knowledge with EPFC. *Eclipse Review*, 1(2):26–33.

IBM Rational Software (2009). Rational Team Concert 2.0 Release, from https://jazz.net/projects/rational-team-concert/.

Martinho, R., Domingos, D., and João Varajão (2009a). On a concept map for the modelling of controlled flexibility in software processes. Technical report TR-2009-12, Departamento de Informática, Faculdade de Ciências da Universidade de Lisboa, Campo Grande, 1749–016 Lisboa, Portugal.

Martinho, R., Domingos, D., and Varajão, J. (2007). FlexUML: A UML Profile for Flexible Process Modelling. In Institute, K. S., editor, *Proceedings of the 19th International Conference of Software Engineering and Knowledge Engineering (SEKE'2007)*, volume 32, pages 215–220, Skokie, IL, USA. Knowledge Systems Institute Graduate School.

Martinho, R., João Varajão, and Domingos, D. (in press) (2009b). Modelling and learning controlled flexibility in software processes. *International Journal on Knowledge and Learning*, 5.

Martinho, R., Varajão, J., and Domingos, D. (2008). A Two-Step Approach for Modelling Flexibility in Software Processes. In Marchetti, E., editor, *Proceedings of the 23rd IEEE/ACM International Conference on Automated Software Engineering (ASE'2008)*, pages 427–430, Piscataway, NJ, USA. IEEE eXpress Publishing.

McGowan, C. L. and Bohner, S. A. (1993). Model based process assessments. In Basili, V. R., DeMillo, R. A., and Katayama, T., editors, *Proceedings of the 15th international conference on*

Software Engineering (ICSE'93), pages 202–211, Los Alamitos, CA, USA. IEEE Computer Society Press.

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

Osterweil, L. J. (1987). Software Processes are Software Too. In Riddle, W. E., editor, *Proceedings of the 9th International Conference on Software Engineering (ICSE '87)*, pages 2–13, New York, NY, USA. ACM Press.

Pesic, M., Schonenberg, M. H., Sidorova, N., and van der Aalst, W. M. P. (2007). Constraint-based workflow models: Change made easy. In Meersman, R. and Tari, Z., editors, *Proceedings of the OTM Conference on Cooperative information Systems (CoopIS 2007)*, volume 4803 of *Lecture Notes in Computer Science*, pages 77–94, New York, NY, USA. Springer-Verlag.

Reichert, M., Dadam, P., Rinderle-Ma, S., Jurisch, M., Kreher, U., and Göeser, K. (2009). *Architectural Principles and Components of Adaptive Process Management Technology*, volume P-151 of *Lecture Notes in Informatics*, pages 81–97. Koellen-Verlag, Bonn, Germany.

Schonenberg, H., Mans, R., Russell, N., Mulyar, N., and van der Aalst, W. M. P. (2008). Process flexibility: A survey of contemporary approaches. In Bellahsene, Z. and Léonard, M., editors, *Proceedings of Advances in Enterprise Engineering I, 4th International Workshop CIAO! and 4th International Workshop EOMAS, held at CAiSE 2008*, volume 5074 of *Lecture Notes in Computer Science*, pages 16–30, Berlin, Germany. Springer-Verlag.

Soffer, P. (2005). On the Notion of Flexibility in Business Processes. In Pastor, O. and e Cunha, J. F., editors, *Proceedings of the 17th International Conference on Advanced Information Systems Engineering (CAiSE'05)*, volume 3520 of *Lecture Notes in Computer Science*, pages 35–42, New York, NY, USA. Springer-Verlag.

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

TIBCO Software Inc. (2009). Introduction to TIBCO iProcess Suite, from http://www.tibco.com/multimedia/wp-tibco-iprocess-suite_tcm8-786.pdf.

Turetken, O. and Demirors, O. (2008). Process modeling by process owners: A decentralized approach. *Software Process Inprovement and Practice*, 13(1):75–87.

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 Lamsweerde, A. (2008). Requirements engineering: From craft to discipline. In Harrold, M. J. and Murphy, G. C., editors, *Proceedings of the 16th ACM SIGSOFT International Symposium on Foundations of Software Engineering (SIGSOFT'08/FSE-16)*, pages 238–249, New York, NY, USA. ACM Press.

Weber, B., Reichert, M., and Rinderle-Ma, S. (2008). Change patterns and change support features - enhancing flexibility in process-aware information systems. *Data & Knowledge Engineering*, 66(3):438–466.

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