Published in IET Software
Received on 15th April 2010
Revised on 16th July 2010
doi: 10.1049/iet-sen.2010.0045

In Special Issue on Social Semantic Web Support for Software Development

ISSN 1751-8806

Using the semantic web to define a language for modelling controlled flexibility in software processes

R. Martinho[1,*] *J. Varajão*[2,†] *D. Domingos*[3,*]

[1]*School of Technology and Management, Polytechnic Institute of Leiria Campus 2, Morro do Lena, Alto do Vieiro, 2411-901 Leiria, Portugal*
[2]*Department of Engineering, University of Trás-os-Montes e Alto Douro, 5001-801 Vila Real, Portugal*
[3]*Department of Informatics, Faculty of Sciences, University of Lisboa, Edifício C6, Piso 3, Campo Grande, 1749-016 Lisboa, Portugal*
[*]*LaSIGE – Large-Scale Informatics Systems Laboratory, Department of Informatics, Faculty of Sciences, University of Lisboa, Edifício C6, Piso 3, Campo Grande, 1749-016 Lisboa, Portugal*
[†]*CAlg – Centro ALGORITMI Universidade do Minho, Campus Azurém, 4800 Guimarães, Portugal*
E-mail: rmartin@estg.ipleiria.pt

Abstract: Software processes and corresponding models are dynamic entities that must evolve to cope with changes occurred in the enacting process, the software development organisation, the market and the methodologies used to produce software. However, in the everyday practice, software team members do not want total flexibility. They rather prefer to learn about and follow previously defined controlled flexibility, that is, advices on which, where, how and by whom process models and related instances can change/adapt. Process engineers can express these advices within a process model with a domain-specific language (DSL), which complements the core process modelling language with additional controlled flexibility information. Then, software team members can browse and learn on this information in process models and instances, and be guided when performing changes. In this study, the authors propose the use of the semantic web and associated ontology-based technologies to develop and evolve their controlled flexibility DSL for software processes. They use an ontology-based format to define the controlled flexibility-related concepts, descriptions and axioms that specify the formal semantics of their DSL. In addition, the authors provide concrete mappings between these ontology concepts and a unified modelling language class-based DSL metamodel and describe how it supports changes made in the ontology.

1 Introduction

Software process modelling involves eliciting and capturing informal descriptions of a process used to develop software, and then converting them into a model [1]. A process model is expressed by using a suitable process modelling language (PML). Owing to the changeability essence of software, these process models are dynamic entities that process participants, such as process engineers and software development team members, often change and evolve. To enable these changes, flexibility has become one of the most important features within PML and supporting tools [2].

However, software team members like to be guided by process models when performing their work [3, 4]. This includes advice on a process's controlled flexibility, that is, information on which, where, how and by whom process models and related instances can be changed [5]. Therefore process engineers should be able to construct process models with this additional information, by using a domain-specific language (DSL), which complements the core PML.

Then, software team members can access this controlled flexibility information by browsing it in process models and

instances, and be guided when performing changes. Nevertheless, in the software process model lifecycle, these DSLs are constantly evolving because of the rapid changing nature of software processes and the intuitive knowledge work involved. Therefore these DSLs are always best developed in conjunction with the people who participate in or are affected by the processes they define [6]. It is important for both process engineers (DSL designers) and software team members (DSL users) to be familiar with the concepts behind the DSL language constructs.

In this paper, we propose the use of the semantic web (SW) and associated ontology-based technologies to develop and evolve our controlled flexibility DSL for software processes. The advantages of using this ontology-based DSL development include easier identification and negotiation of DSL's domain concepts, more precise specification and comprehension of the semantics of these concepts, guidance through the development of DSLs, and consistency checking of the developed DSL.

We use concept maps (Cmaps) along with the web ontology language (OWL) to define a controlled flexibility ontology, which includes the concepts, descriptions and axioms that define the formal semantics of our controlled flexibility-aware DSL. We also define concrete mappings between these ontology concepts and a unified modelling language (UML) class-based DSL metamodel.

In the context of our controlled flexibility for software processes (ConFlex4SP) framework (see e.g. [7]), this metamodel is to be adapted onto a core PML and implemented within a software process management tool. This way, process engineers and software team members are able to apply the DSL language constructs with controlled flexibility information to the design and use of software process models. Furthermore, and working together with process engineers, team members can help in evolving the DSL by suggesting new concepts or changing existing ones through ontology-based technologies and tools.

This paper is organised as follows: the next section refers most prominent works on the use of the SW jointly with DSL metamodelling and specification. Section 3 provides an overview of our approach by describing the ConFlex4SP framework. Section 4 addresses the ontology-based formats and technologies we use to define our DSL's conceptual base. In Section 5, we define the controlled flexibility ontology, and in Section 6, we explain the mappings between the ontology concepts and a meta-object facility (MOF)-based DSL metamodel. Finally, Section 7 concludes the paper and addresses future work.

2 Related work

The term 'ontology' originates from philosophy, denoting the study of existence. In computer science, the most common definition has been provided by Gruber [8]: 'An ontology is an explicit specification of a conceptualisation'. Here, ontologies are typically used as a formal and explicit way of specifying the concepts and relationships in a domain of discourse [9]. This is particularly important in the context of the SW. SW, commonly regarded as the next generation of the web, extends the current web by giving to its content a well-defined meaning, enabling computers and people to work in cooperation. It represents an emerging vision for a new kind of web with enhanced functionality, which will require semantic-based representation and processing of web information [10]. This paradigm has recently advanced across many domains for the assignment of metadata to Internet content, in order to define it with explicit, machine-readable meaning [11]. A good example of that is SEMO, which is a framework for customer social networks analysis based on semantics [12].

Since the elements of software development teams, such as programmers and analysts, constitute a critical group of knowledge workers in modern organisations [13] and they are generally professionals with a heterogeneous training, background and expertise [14], SW is becoming an important tool in the context of knowledge sharing and alignment between these professionals.

To realise the SW vision, many techniques and technologies have been proposed. The OWL and the emerging new version OWL2 are important ontology languages that the World wide web consortium has been proposing. They have an increasing expressiveness, making it well suited to formally define DSL metamodels [15]. The OWL language facilitates a greater machine understandability of web resources than the one supported by resource description framework schema, by providing additional constructors for building class and property descriptions (vocabulary) and new axioms (constraints), along with a formal semantics. The capability to describe classes in many different ways and to handle incomplete knowledge distinguishes OWL from class-based modelling languages like UML class diagrams [15].

We can find in the literature, several recent works in this new field. For instance, Happel and Seedorf [16] identify potential uses of ontologies in software engineering and provide a framework for categorising them. Walter *et al.* [15] present an ontology-based framework for DSLs that allows the definition of DSLs enriched by formal class descriptions, integrating existing metamodels, concrete syntax and a query language. Tairas *et al.* [17] apply ontologies in the early stages of domain analysis to identify domain concepts.

Model-driven software development (MDSD) advocates the use of domain specific modelling languages (DSLs) for describing software systems [18]. DSLs are commonly used to model and develop systems of application domains [15]. Each DSL focuses on different problem domains and, as far as possible, on automatic code generation [19].

In recent years, the importance of DSLs for describing software systems has increased and a convergence with MDSD can be witnessed. On one hand, when compared with large general purpose modelling languages, such as the UML, DSLs only offer a limited set of constructs [18], increasing the productivity and facilitating a precise definition of concerns within a particular domain [20]. On the other hand, modelling complex systems usually requires that several different DSLs and fragments of their models are used to develop them [15]. The result is the development of several domain-specific models, each describing one aspect of an application (e.g. data structures) [18].

DSLs are usually specified using metamodelling [20]. In multi-DSL development scenarios, this does not suffice to ensure the consistency and validity of all models constituting a complex system description, because the semantic relationships between constructs from different languages remain unspecified [18]. DSLs are high-level and should provide abstractions and notations for better understanding and easier modelling of applications of a special domain [15].

To address the insufficiencies of current approaches, a number of recent proposals argue in favour of ontologies as a means to specify language semantics and integrate multiple DSLs [21]. Parreiras *et al.* [22] propose agogo: a model-driven approach to automatically generate OWL APIs on demand. Bräuer and Lochmann [18], following [23], propose the use of ontologies to describe the semantic relationships and interdependencies between different DSLs, aiming to establish semantic links between different domain-specific models using an ontology knowledge base. They define a hierarchy of ontologies, namely

- upper ontologies – define concepts that are applicable in most if not all domains;
- core ontologies – define concepts shared by a number of similar domains;
- domain ontologies – define concepts specific to a particular area of interest;
- application ontologies – specialise the concepts from an upper, core or domain ontology with application-specific variants.

In this case, the main motivation for starting with an upper ontology is to provide a number of well-defined, theoretically principled and sufficiently axiomatised concepts as a base for concepts in more specialised ontologies. As argued in [23], the resulting core and domain ontologies usually have a higher quality than those developed bottom-up from a selection of sample languages [18].

Summing up, several works advocate the use of SW, ontologies and related languages as a preferred way of defining the concepts behind a DSL. However, most languages for defining ontologies (e.g. OWL and RDF) tend to be machine centred and of complex understandability for humans. None of the works referred foresees the challenge of having a more human-centred ontology format for defining the concepts that will be mapped onto language constructs in a DSL.

We advocate the use of this human-centred ontology format as a media for understanding, knowledge acquisition, sharing and negotiation (changes/updates) of DSL concepts prior to metamodelling, between both DSL designers (process engineers) and users (software team members). In our case, the concepts of this ontology format reflect a DSL for modelling controlled flexibility in software processes. The human-centred ontology format can easily be exported to a more machine-centred standard format (e.g. OWL2) and therefore take advantage on consistency checks and validations regarding the ontology.

Unlike several approaches referred above, which provide automatic mappings between ontologies and DSLs defined through MOF-based (UML class) metamodels, we adopt a more customised metamodelling strategy. It is based on a flexible metamodel structure, which is able to handle changes negotiated at the ontology level, in the context of our DSL. This strategy is part of the ConFlex4SP framework, which we describe in the next section.

3 ConFlex4SP framework

In this section, we provide an overview the ConFlex4SP framework. It illustrates our approach to control flexibility in software processes, which includes metamodelling a DSL that adds language constructs to a core PML. We called this DSL the controlled flexibility language (CFL), and with it process engineers are able to express controlled flexibility information within software process models. Then, software team members can browse, learn and change those process models and related instances, according to the previously modelled controlled flexibility information.

3.1 Overview

Fig. 1 fits this approach that we name the controlled flexibility for software processes (ConFlex4SP) framework. The black-lined part of the lifecycle denotes the areas of intervention conveyed by the framework.

Firstly, we have identified a set of concepts and relationships, and we have organised them onto the DSL metamodel. This is enforced by performing metamodelling, to which process engineers contribute. Our approach does not aim at including a throughout definition of a software process family metamodel and PML (such as the software

doi: 10.1049/iet-sen.2010.0045

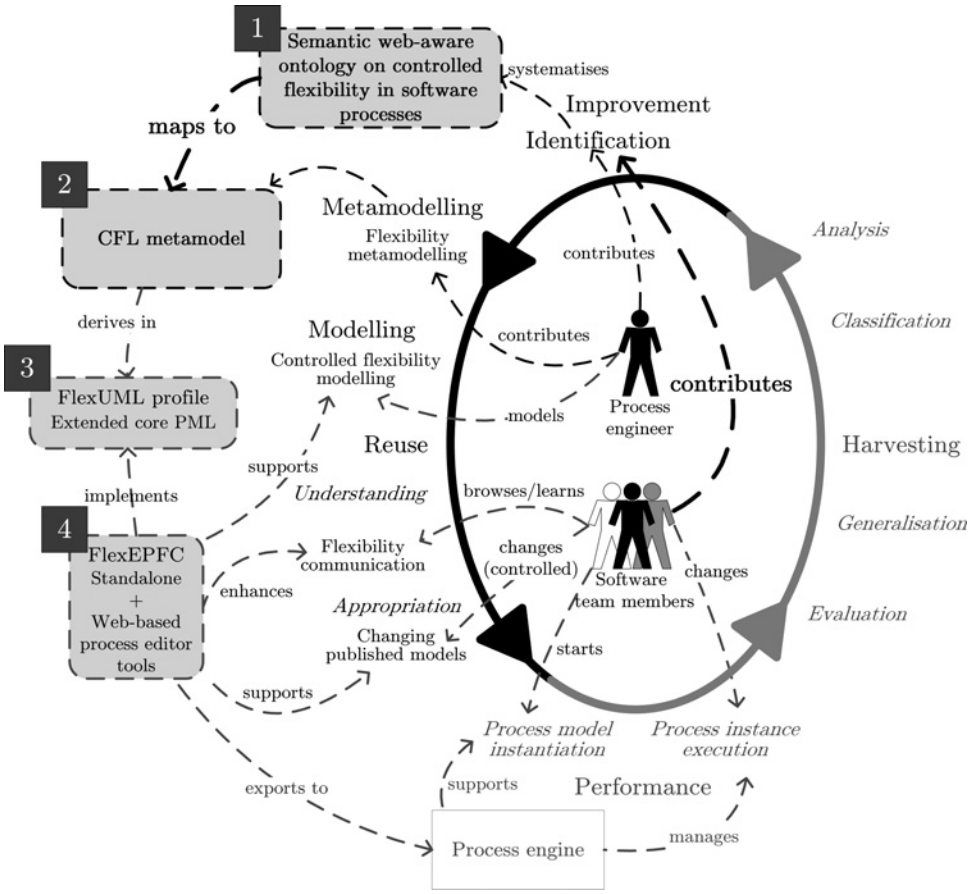

Figure 1 *Scope of the ConFlex4SP framework, regarding the software process model lifecycle*

process engineering metamodel (SPEM) in [24]). Instead, we perform metamodelling in order to achieve a common use for metamodels: a language to express additional semantics of existing information. Therefore this contribution focuses on metamodelling controlled flexibility-related additional semantics and extending an existing core PML with these semantics.

We design the metamodel structure of our DSL (the CFL language) with the object management group (OMG)'s UML standard (solution number 2 in Fig. 1). This metamodel also includes well-formedness rules, which we define using another OMG's standard, the object constraint language. Still part of the metamodelling activity is the use of UML profiles as the preferred mechanism to extend a chosen core PML. We define the FlexUML (Fig. 1's solution number 3) which includes UML stereotypes and tag definitions that a process engineer can use to express controlled flexibility in software process models. Earlier versions regarding the CFL language metamodel and the FlexUML profile can be found in [25, 26].

The modelling activity-related solutions in Fig. 1 refers to the implementation of the extended core PML within a customised software tool. In Fig. 1, this is referred as solution number 4, in which lies the FlexEPFC software tool. It includes code artifacts that implement the FlexUML profile (and the underlying CFL language constructs), as well as the necessary presentation logic and user interface widgets that a process engineer needs to apply FlexUML stereotypes to software process models. We have also enhanced the web-based electronic process guide, generated by FlexEPFC. It is called WebFlexEPFC, and includes a new interactive process editor, based on the graphical notations defined in the extended core PML. Software team members can now browse and learn about the controlled flexibility information defined in the published process models. They can also perform changes to these models accordingly. These modelling contributions have already been published in [7, 27].

3.2 Evolving ConFlex4SP

We have been refining and evolving this framework over the past three years, and its versions can be found in previously published work. Earlier in [27], we began by eliciting the distinction between the metamodelling and the modelling activities that are performed by process engineers and software team members, respectively.

In [7], we separated the construction of our DSL from the core PML, its extension mechanisms and the tools to be adopted for its implementation. We also introduced the use

of Cmaps as an easier and more human-centred representation for eliciting the controlled flexibility concepts to include in our DSL.

Still, we did not accommodate the possibility of having software team members to actively contribute in adding or updating these concepts (shown by a 'contributes' highlighted arrow in Fig. 1). This is a long time recommended practice when designing PMLs [6].

Therefore we need to improve the use of Cmaps and its ontology-based formats and functions to enhance software team members participation in refining and evolving our DSL. The next section emphasises on the use of Cmaps and related tools to collaborate on the definition of an ontology as the starting point to the metamodelling of our DSL.

Also, we need to clearly identify the mappings between the concepts and relationships expressed in Cmaps and our DSL metamodel structure (the highlighted 'maps' arrow in Fig. 1). We address these mappings in Section 6.

4 Ontology formats and technologies

In this section, we justify the ontology-based formats and technologies we choose to define our DSL's conceptual base. The use of ontologies pursuit a standard way of specifying content-specific agreements for the sharing and reuse of knowledge [28]. A body of formally represented knowledge is based on a conceptualisation: the objects, concepts and other entities that are assumed to exist in some area of interest and the relationships that hold among them [29].

An ontology is an explicit specification of a conceptualisation [28]. This specification can assume more or less formal representations, according to the purpose of the ontology, and to whom/what will be the sharing target ofthe knowledge it defines. If it is to be shared among humans, it can assume a less formal and human-centred representation. If the reasoner is automated (e.g. a software component), then a more formal and machine-centred language applies, such as RDF and OWL [30] web standards associated with the SW.

Regarding our context, the ideal solution should include both these kinds of representation formats. This means that, on one hand, we need a less formal format and supporting tools to enable process engineers and software team members to:

- easily change concepts and relationships – by searching and editing concepts' metadata, classes, properties and individuals;

- perform collaborative work – by adding notifications and comments, versioning ontologies and registering changes (who and what), and providing an ontology and related concepts quality indicator, according to participants' opinions (e.g. pool voting).

On the other hand, a more formal OWL-based ontology format and tools can provide:

- consistent integration and validation between our DSL language constructs and the extended core PML (UML, in this case);

- abstractions and notations for better understanding and easier modelling of software process models with controlled flexibility information.

To address all these issues, we use Cmaps and CmapTools [31] as a less formal but proven method and tool for explaining and communicating the knowledge of a certain domain [32]. CmapTools have been developed over the previous decade as an intuitive, human-centred computer interface for creating and managing Cmaps. We use it to create and manage the relationships between our proposed concepts. It also includes export and import features for ontologies represented in formal OWL-derived languages [OWL (an OWL format of our ConFlex4SP ontology can be found at http://www.knoodl.com/ui/groups/software_process_controlled_flexibility/vocab/ConFlex4SP(2.0)) and RDF].

According to Novak and Cañas [31], a Cmap is a graphical tool for organising and representing knowledge. It includes concepts that are graphically represented by a box or circle drawing shape, and relationships represented by a connecting line linking two concepts. Lines have also a text associated, referred to as linking words or linking phrases, which specify the relationship between the two concepts.

These are the basics on Cmaps that we apply in the next section to capture the knowledge domain of modelling controlled flexibility within software processes.

5 Cmaps for building our DSL ontology

In our approach, we use Cmaps and CmapTools to enhance the learning and sharing of process-controlled flexibility-related concepts among process engineers and software team members. To achieve our purpose, we begin by providing, in the next section, the focus question which our Cmaps must be able to answer. Next, we present Cmaps diagrams and define their concepts and relationships.

5.1 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 focus question, to which the Cmap will be designed to answer [31].

doi: 10.1049/iet-sen.2010.0045

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?' [33].

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?'

5.2 Diagrams

In a Cmap, concepts are drawn and associated in a diagram. The structure of this diagram is usually hierarchical for Cmaps that abound on static relationships. However, for dynamic ones, there are other structures such as cyclic Cmaps [34] that are more adequate. In our diagrams, we use a mix between hierarchies to represent belongingness, composition and categorisation, and semantically layered concepts to represent dynamic relationships. The diagrams of Figs. 2 and 3 represent our running example on the use of an SW approach to define an ontology for a DSL.

Fig. 2 illustrates the main Cmap that captures our knowledge domain, whereas Fig. 3 provides a Cmap that show generalisation, composition or belongingness relationships pertaining some concepts of Fig. 2.

In Fig. 3 some concepts are filled with four question marks. This means that the general top (class) concept can have more or different specialised (individual) concepts beyond the samples provided, since they can depend on factors such as the modelling language, application domain, tool support and/or organisational context.

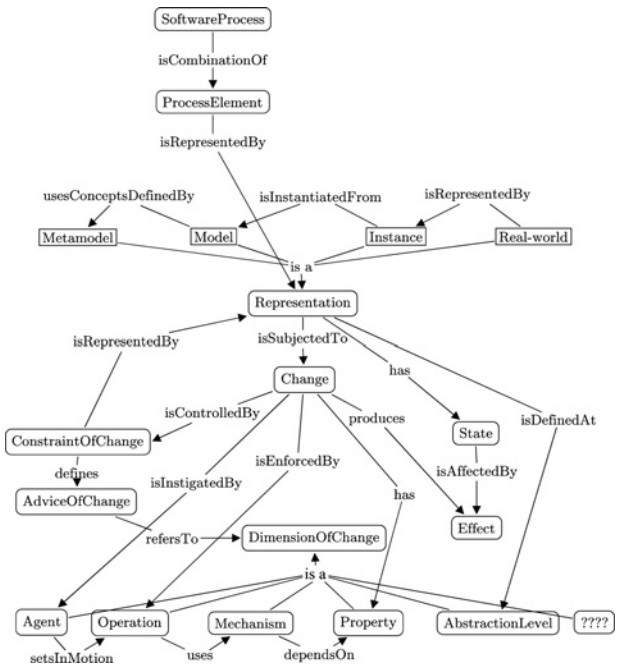

Figure 2 *Core Cmap for capturing knowledge involved in modelling controlled flexibility in software processes*

In the next section, we define the concepts and relationships presented in these figures.

5.3 Concepts and relationships

In the following, we define the concepts and relationships presented in both Figs. 2 and 3. The diagram of Fig. 2 shows that a SoftwareProcess is a combination of ProcessElements represented by a model. These elements can be used to express correlated process perspectives, such as the functional, behavioural, organisational and informational ones defined in [1]. A process element's 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 that defines UML activity diagrams as the core PML. Process models are then created as instances of the metamodel and included 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 comprising well known software methods such as the unified process [35].

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 individuals of the process element representation class, defined at distinct but correlated AbstractionLevels of modelling. These representations are subjected to changes, which in turn have effects that can affect their states.

A change is characterised by property elements 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 process element in a process running instance. Actually, change operations are the actions that will change the state of process element representations.

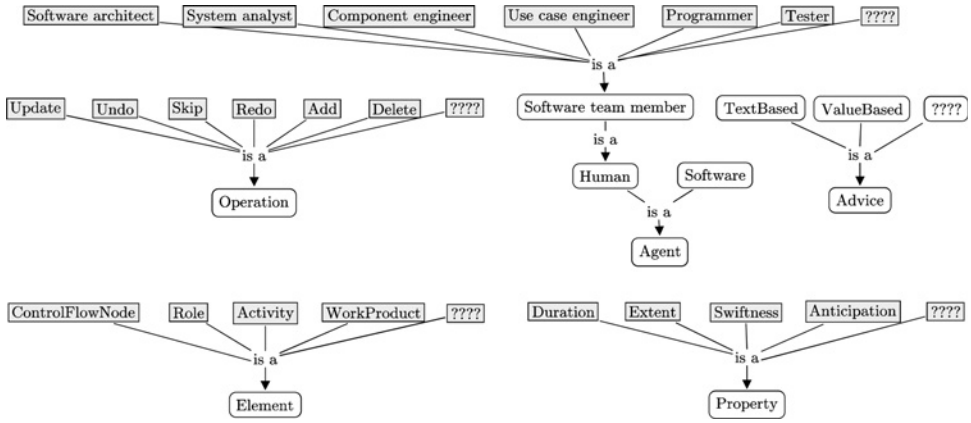

Figure 3 *Generalisation relationships of concepts illustrated in Fig. 2*

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 extent, duration, swiftness and anticipation of change [36, 37].

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 [38]. 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 a set of ConstraintOfChange elements, which are also have representations. These constraints define AdviceOfChange, which refer to one or more DimensionsOfChange, such as abstraction levels, 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

$$\begin{aligned}&(\text{tbAdvice:TextBasedAdvice} = \text{denied},\\ &\quad \text{absLevel:AbstractionLevel} = \text{instance};\\ &\quad \text{operation:Operation} = \text{skip})\end{aligned}$$

The concepts above are described and supported by the aforementioned prominent research works, which in turn have foundations mostly on empirical knowledge on software engineering. A more detailed version of these concepts and related discussions can be found in [39]. Nevertheless, the Cmaps and/or other possibly derived ontology formats are shared in a centralised repository, which process engineers and software team members can access. Therefore they can learn, share, redefine, discuss and/or change them, according to their needs. The next section presents the mappings of the concepts above onto a MOF-based metamodel for our DSL.

6 Mapping an ontology to a DSL metamodel

Considering the diagrams and concept descriptions of the previous section, process engineers should be able to define controlled flexibility through the use of constraints of change. These can have distinct degrees of detail, and the set of constraints of a process element can perceive different levels of complexity.

In addition, process engineers should be able to define constraints of change for every type of process element, such as activities, control flow nodes (join, fork, decision and merge nodes), roles, role assignments, artefacts, artefact flows (input and output parameters from activities) and resource allocations. Constraints of change may be applied to any type of process element using the same tuple components.

doi: 10.1049/iet-sen.2010.0045

However, the final extended PML metamodel should keep intact the possibility for process engineers to continue designing software process models with no controlled flexibility, that is, models with none of our DSL-based constraints of change associated.

To address these requirements, we propose, in the next subsection, the CFL metamodel (our DSL).

6.1 Metamodel structures

Fig. 4 illustrates the CFL metamodel as a UML class diagram-based metamodel that defines the logic involved with constraints of change and controlled flexibility-enabled process elements.

It comprises three related substructures, denoted by the following differently grey-shaded classes and their corresponding associations:

1. ConFlexElement and COfChange;

2. CFLExpression, CFLPart, CFLNameLiteral, CFLTypeLiteral and CFLValue;

3. CFLExpression and the corresponding hierarchy of expressions, including the shared association between ExpressionDecorator and CFLExpression.

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

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

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

The expression informs a software team member that it is recommended to change the model representation of the constrained ConFlexElement. We can observe that the expression follows a tuple-like format, and results from the composition of a variable number of CFLPart elements (two for the example above). Each CFLPart has a name literal (CFLNameLiteral), a type literal (CFLTypeLiteral) and a value (CFLValue).Considering the example above, the first CFLPart instance is defined by a tbAdvice name literal, a TBAdvice type literal and recommended value. A CFLExpression must have, at least, one CFLPart, and

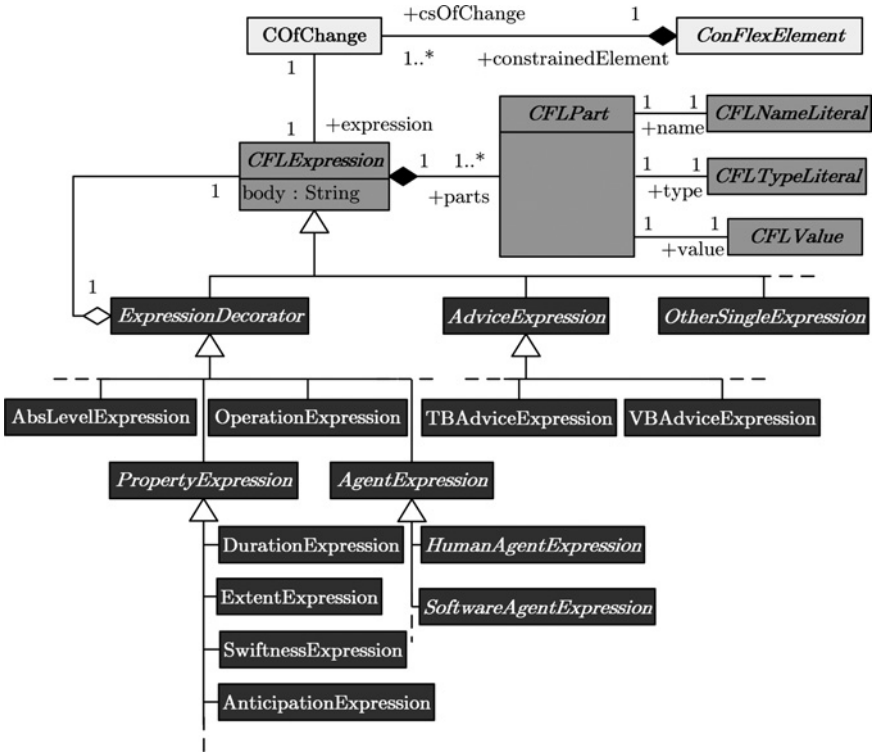

Figure 4 *Structure of the CFL metamodel*

aCFLPart belongs only to one CFLExpression at a time. The CFLTypeLiteral of a part is optional (association with multiplicity of 0..1 to 1), and therefore the expression above can also be written in its simplest form

{tbAdvice = recommended, absLevel = model}

The third substructure of the CFL constructs is composed by CFLExpression and the corresponding hierarchy of expressions, along with the shared association between ExpressionDecorator and CFLExpression. The structure applies the decorator structural software pattern of [40]. As we have already stated, a constraint of change must be able to assume different degrees of detail in its definition. It can vary from having a single AdviceExpression to a combination of CFLExpression elements, composing a constraint of change with more detail.

Our expression example is itself a combination of a TBAdviceExpression with an AbsLevelExpression. One alternative to this decorator-based structure is defining a metamodel comprising all possible combinations of expressions, which can be achieved through an extensive static subclassing approach. In this case, we would have to consider a hierarchy of CFLExpression elements to address each possible combination. For instance, we would have a metamodel with elements such as AdviceAbstractionLevelExpression, AdviceOperation Expression, AdvicePropertyExpression, AdviceAgen tExpression, AdviceAbstractionLevelOperation Expression, AdviceAbstractionLevelPropertyExpression etc., until we reach the number of possible combinations between these expressions. This approach would give rise to a metamodel with many classes, some of them hard to implement, because of the amount of responsibilities that larger expressions would have to address.

Also, at some point, it is hard to keep track of all implemented combinations of expressions. In addition, for each new type of expression that a software organisation felt the need to have in their constraints, the programming effort would have to include additional implementations for all possible combinations between the new expression and the existing ones. Finally, by implementing combinations in advance, the tool that will support the modelling of these expressions cannot provide the process engineer the possibility of freely attach and detach expressions at runtime. Every combination must be statically defined prior to its use.

The decorator pattern from [40] avoids these disadvantages of static subclassing. It enables a process engineer to combine several CFLExpressions of different types into a global one. This means that a mandatory AdviceExpression (leaf expression) can be decorated with other (non-leaf) expressions at runtime by the process engineer. The result is still a single CFLExpression object, which begins by being a single AdviceExpression, to which is later added responsibilities through the CFLExpression decorators, such as the AbsLevelExpression one. Regarding the expression example referred above, its concrete implementation comprises a leaf TBAdviceExpression and a AbsLevelExpression decorator.

This sums up the process of analysing and deriving a MOF-based DSL metamodel having an SW-based ontology and supporting tools as the starting point. The next subsection exemplifies how adding or changing a concept in the Cmaps-based ontology is reflected in the CFL metamodel.

6.2 Applied example

In this section, we provide an example of adding a new concept to the Cmaps-based ontology, and how this is reflected in our CFL metamodel. As referred in Section 5, both Cmaps illustrated in Figs. 2 and 3 contain concepts filled with four question marks. An example is the one in Fig. 2, associated with the DimensionOfChange concept. It means that these dimensions are not limited to the examples provided in the figure, through the relationship 'is a'.

Our example reports to a software practitioner that, after using our CFL, proposed a new dimension of change that reflected the maximum size of the software team (number of members) for which changes to software process representations are supposed to be advised. For example, a process engineer would like to enforce the following constraint of change: 'skipping the Write documentation activity instance is allowed for teams up to 4 members'. For this, s/he added a new concept to Fig. 2's Cmap called TeamSize and the corresponding relationship of type 'is a' towards the DimensionOfChange.

Mapping this new ontology concept onto the CFL metamodel implies adding one more CFLExpression type, in order to be able to have a constraint expression like

{tbAdvice = allowed, absLevel=instance,
operation = skip, maxTeamSize = 4}

Owing to our flexible and decorator-based metamodel structure, this can be achieved quite straightforwardly. The new TeamSize concept can be reflected in a new TeamSizeExpression element, which in turn is a specialisation of the abstract ExpressionDecorator one.

This means that mandatory AdviceExpressions can now be decorated with an additional expression that indicates the maximum team size to which a change should be advised. The decorator structure also assures that the new expression type can be combined with other distinct expressions seamlessly.

7 Conclusions and future work

In this paper, we illustrate how ontologies and the SW-related technologies can be used to form a solid conceptual framework

doi: 10.1049/iet-sen.2010.0045

for a DSL. Our DSL's purpose is to define language constructs to help process engineers and software team members to control flexibility in software process representations.

By using Cmaps and CmapTools, we were able to address three correlated challenges: (i) providing a less formal and easier ontology format in order to motivate process engineers and software team members in the participation of an ontology definition; (ii) using import and export feature of CmapTools to integrate or derive formal OWL-based ontology formats. These can be easily kept in a repository, shared and queried by software reasoners, and used to validate the DSL concepts against their integration with an existing core PML; and (iii) mapping (new or changed) concepts from an ontology onto a DSL metamodel.

We are working on the automation of mappings between these formal ontology formats and our design pattern-oriented DSL metamodel. We will start by automating mappings between ContraintOfChange and the decorator-based expressions illustrated in the metamodel. This will allow process engineers and software team members to almost instantly manage (add, change, delete) the controlled flexibility-aware language constructs regarding constraints of change expressions and containing components. This way, we provide a systematised way of supporting our DSL's evolution through the use of a human-centred and user-friendly ontology-based approach such as Cmaps and CmapTools and mapping its concepts directly to our DSL metamodel structure.

For instance, upon the need for expressing a new dimension of change through our DSL, process engineers and software team members can define and agree upon it using Cmaps. Then, an OWL-based exported format can be used to consider that dimension of changeas an additional constraint expression, with new component elements also derived from Cmaps.

8 References

[1] CURTIS B., KELLNER M.I., OVER J.: 'Process modeling', *Commun. ACM*, 1992, **35**, pp. 75–90

[2] REICHERT M., RINDERLE-MA S., DADAM P.: 'Flexibility in process-aware information systems', *LNCS Trans. Petri Nets Other Models Concurrency* (ToPNoC), (Special Issue on Concurrency in Process-aware Information Systems), 2009, **2**, pp. 115–135

[3] CASS A.G., OSTERWEIL L.J.: 'Process support to help novices design software faster and better'. Proc. 20th IEEE/ACM Int. Conf. on Automated Software Engineering (ASE'05), Long Beach, CA, USA, November 2005, pp. 295–299

[4] BIDER I.: 'Masking flexibility behind rigidity: notes on how much flexibility people are willing to cope with'. Proc. 17th Int. Conf. on Advanced Information Systems Engineering (CAiSE'05), Porto, Portugal, June 2005, pp. 7–18

[5] BORCH S.E., STEFANSEN C.: 'On controlled flexibility'. Proc. Seventh Workshop on Business Process Modeling, Development and Support (BPMDS'06) co-located with the 18th Conf. on Advanced Information Systems Engineering (CAiSE'06), Luxembourg, June 2006, pp. 121–126

[6] CONRADI R., JACCHERI M.L.: 'Process modelling languages' in DERNIAME J.-C., KABA B.A., WASTELL D.G. (EDS.): 'Software process: principles, methodology and technology' (Springer Berlin/Heidelberg, 1999), pp. 27–52

[7] MARTINHO R., VARAJÃO J., DOMINGOS D.: 'Modelling and learning controlled flexibility in software processes', *Int. J. Knowl. Learn.*, 2009, **5**, pp. 423–442

[8] GRUBER T.R.: 'A translation approach to portable ontology specifications', *Knowl. Acquis. J.*, 1993, **5**, (2), pp. 199–220

[9] ZHANG Y., WITTE R., RILLING J., HAARSLEV V.: 'Ontological approach for the semantic recovery of traceability links between software artefacts', *IET Softw.*, 2008, **2**, (3), pp. 185–203

[10] WANG H.H., DONG J.S., SUN J., SUN J.: 'Reasoning support for semantic web ontology family languages using alloy', *Multiagent Grid Syst.*, 2006, **2**, (4), pp. 455–471

[11] GARCÍA-CRESPO A., COLOMO-PALACIOS R., GÓMEZ-BERBÍS J.M., GARCÍA-SÁNCHEZ F.: 'SOLAR: social link advanced recommendation system', *Future Gener. Comput. Syst.*, 2010, **26**, (3), p. 374 – 380

[12] GARCÍA-CRESPO A., COLOMO-PALACIOS R., GÓMEZ-BERBÍS J.M., RUIZ-MEZCUA B.: 'SEMO: a framework for customer social networks analysis based on semantics', *J. Inf. Technol.*, 2010, **25**, (2), pp. 178–188

[13] TRIGO A., VARAJÃO J., SOTO-ACOSTA P., BARROSO J., MOLINA-CASTILLO J., GONZALVEZ-GALLEGO N.: 'IT professionals: an Iberian snapshot', *Int. J. Hum. Capit. Inf. Technol. Prof. (IJHCITP)*, 2010, **1**, (1), pp. 61–75

[14] GARCÍA-CRESPO A., COLOMO-PALACIOS R., GÓMEZ-BERBÍS J.M., MENCKE M.: 'BMR: benchmarking metrics recommender for personnel issues in software development projects', *Int. J. Comput. Intell. Syst.*, 2009, **2**, (3), pp. 256–266

[15] WALTER T., PARREIRAS F.S., STAAB S.: 'OntoDSL: An ontology-based framework for domain-specific languages'. Proc. 12th ACM/IEEE Int. Conf. on Model Driven Engineering Languages and Systems (MODELS'09), Denver, Colorado, USA, October 2009, pp. 408–4227

[16] HAPPEL H.J., SEEDORF S.: 'Applications of ontologies in software engineering'. Proc. Int. Workshop on Semantic

Web Enabled Software Engineering (SWESE'06), Athens, GA, USA, November 2006, pp. 1–14

[17] TAIRAS R., MERNIK M., GRAY J.: 'Using ontologies in the domain analysis of domain-specific languages'. Proc. First Int. Workshop on Transforming and Weaving Ontologies in Model Driven Engineering, Toulouse, France, September 2008, pp. 332–342

[18] BRÄUER M., LOCHMANN H.: 'Towards semantic integration of multiple domain-specific languages using ontological foundations'. Proc. Fourth Int. Workshop on Software Language Engineering (ATEM'2007), Nashville, Tennessee, USA, October 2007, pp. 28–33

[19] KELLY S., TOLVANEN J.P.: 'Domain-specific modeling: enabling full code generation' (John Wiley & Sons, 2008)

[20] STAHL T., VÖLTER M., EFFTINGE S., HAASE A.: 'Modellgetriebene softwareentwicklung. Techniken, engineering, management' (dpunkt.verlag, Heidelberg, 2007, 2nd edn.) (in Germany)

[21] BAUER B., ROSER S.: 'Semantic-enabled software engineering and development'. INFORMATIK 2006 – Informatik fr Menschen, 2006, Lecture Notes in Informatics (*LNI*, **94**), pp. 293–296

[22] PARREIRAS F.S., SAATHOFF C., WALTER T., FRANZ T., STAAB S.: 'APIs à gogo: automatic generation of ontology APIs'. Proc. Third IEEE Int. Conf. on Semantic Computing (ICSC 2009), Santa Clara, California, USA, September 2009, pp. 342–348

[23] OBERLE D.: 'Semantic management of middleware. Vol. 1 of semantic web and beyond' (Springer, 2006)

[24] OMG: 'Software & systems process engineering meta-model. Object Management Group', 2008 v2.0

[25] MARTINHO R., DOMINGOS D., VARAJÃO J.: 'A flexible perspective for software processes – supporting flexibility in the software process engineering metamodel'. Proc. Ninth Int. Conf. on Enterprise Information Systems (ICEIS'07), Madeira, Portugal, May 2007, pp. 559–562

[26] MARTINHO R., DOMINGOS D., VARAJÃO J.: 'FlexUML: a uml profile for flexible process modelling'. Proc. 19th Int. Conf. on Software Engineering and Knowledge Engineering (SEKE'2007), Boston, Massachusetts, USA, July 2007, pp. 215–220

[27] MARTINHO R., VARAJÃO J., DOMINGOS D.: 'A two-step approach for modelling flexibility in software processes'. Proc. 23rd IEEE/ACM Int. Conf. on Automated Software Engineering (ASE'2008), L'Aquila, Italy, September 2008, pp. 427–430

[28] GRUBER T.R.: 'Toward principles for the design of ontologies used for knowledge sharing', *Int. J. Hum. Comput. Stud.*, 1995, **43**, pp. 907–928

[29] GENESERETH M.R., NILSSON N.J.: 'Logical foundations of artificial intelligence' (Morgan Kaufmann Publishers Inc., 1987)

[30] W3C: 'OWL web ontology language guide – W3C recommendation'. (W3C, MIT, ERCIM, Keio, 2004)

[31] NOVAK J.D., CAÑAS A.: 'The theory underlying concept maps and how to construct and use them'. IHMC CmapTools, 2006-01 Rev 2008-01, Florida Institute for Human and Machine Cognition, 2008

[32] HOFFMAN R.R., WOODS D.D.: 'Studying cognitive systems in context: preface to the special section', *J. Hum. Factors Ergon. Soci.*, 2000, **42**, pp. 1–7

[33] DERBENTSEVA N., SAFAYENI F., CAÑAS A.J.: 'Concept maps: experiments on dynamic thinking', *J. Res. Sci. Teach.*, 2007, **44**, pp. 448–465

[34] SAFAYENI F., DERBENTSEVA N., CAÑAS A.: 'A theoretical note on concepts and the need for cyclic concept maps', *J. Res. Sci. Teach.*, 2005, **42**, pp. 741–766

[35] JACOBSON I., BOOCH G., RUMBAUGH J.: 'The unified software development process' (Addison-Wesley Longman Publishing Co., Inc., 1999)

[36] REGEV G., SOFFER P., SCHMIDT R.: 'Taxonomy of flexibility in business processes'. Input Seventh Workshop on Business Process Modeling, Development and Support (BPMDS'06) Co-located with the 18th Conf. on Advanced Information Systems Engineering (CAiSE'06), 2006, http://lamswww.epfl.ch/conference/bpmds06/taxbpflex

[37] SCHONENBERG H., MANS R., RUSSELL N., MULYAR N., VAN DER AALST W.M.P.: 'Towards a taxonomy of process flexibility (extended version)'. BPMcenter.org, 2007

[38] ROSS A.M., RHODES D.H., HASTINGS D.E.: 'Defining changeability: reconciling flexibility, adaptability, scalability, modifiability, and robustness for maintaining system lifecycle value', *Syst. Eng.*, 2008, **11**, pp. 246–262

[39] MARTINHO R., DOMINGOS D., VARAJÃO J.: 'Concept maps for the modelling of controlled flexibility in software processes', *IEICE Trans. Inf. Syst.*, 2010, **E93-D**, (8), pp. 2190–2197

[40] GAMMA E., HELM R., JOHNSON R., VLISSIDES J.: 'Design patterns: elements of reusable object-oriented software' (Addison-Wesley, 1995)

doi: 10.1049/iet-sen.2010.0045