IEICE TRANS. INF. & SYST., VOL.E93–D, NO.8 AUGUST 2010

PAPER

Concept Maps for the Modelling of Controlled Flexibility in Software Processes

Ricardo MARTINHO[†a)], **Dulce DOMINGOS**[††b)], *Nonmembers*, *and* **João VARAJÃO**[†††c)], *Member*

SUMMARY Software processes and corresponding models are dynamic entities that are often changed and evolved by skillful knowledge workers such as the members of a software development team. Consequently, process flexibility has been identified as one of the most important features that should be supported by both Process Modelling Languages (PMLs) and software tools that manage the processes. However, in the everyday practice, most software team members do not want total flexibility. They rather prefer to have controlled flexibility, i.e., to learn and follow advices previously modelled by a process engineer on which and how they can change the elements that compose a software process. Since process models constitute a preferred vehicle for sharing and communicating knowledge on software processes, the process engineer needs a PML that can express this controlled flexibility, along with other process perspectives. To achieve this enhanced PML, we first need a sound core set of concepts and relationships that defines the knowledge domain associated with the modelling of controlled flexibility. In this paper we capture and represent this domain by using Concept Maps (Cmaps). These include diagrams and descriptions that elicit the relationships between the concepts involved. The proposed Cmaps can then be used as input to extend a PML with modelling constructs to express controlled flexibility within software processes. Process engineers can use these constructs to define, in a process model, advices on changes that can be made to the model itself or to related instances. Software team members can then consult this controlled flexibility information within the process models and perform changes accordingly.

key words: *software processes, modelling, flexibility, control*

1. Introduction

Software process modelling involves eliciting and capturing informal process descriptions and then converting them into process models. A process model is expressed by using a suitable Process Modelling Language (PML), and is best developed in conjunction with the people who participate in, or are affected by the process. Therefore, it is important for them to be familiar with the concepts expressed by the PML. Most common ones include activities, artifacts, work products, roles, control flow elements (e.g., sequencing, fork, join, decision and merge nodes) and object flow elements (e.g., inputs and outputs to activities). These process elements are specified in the PML's metamodel and can be used to define process models. Models are instantiated to realise software projects. Therefore, process instances follow a certain model, and are complemented with specific (runtime) project data such as activities' duration, cost and resource assignments. Process instances are then executed along with projects' real-world enactments.

Software systems that manage and execute software process models are often called Process-centred Software Engineering Environments (PSEE) [1].

These models and instances are commonly held as dynamic entities that often must be changed and evolved, in order to cope with alterations occurring in: the real-world software project (due to changing requirements or unforeseen project-specific circumstances); the software development organization; the market; and in the methodologies used to produce software [2]. Therefore, it should be possible to quickly design new process models, to enable on-the-fly adaptations of running instances, to defer decisions regarding the exact process modelling logic to runtime, and to evolve implemented process models over time. Consequently, process flexibility has been identified as one of the most important features to consider in PMLs and supporting PSEE [3]. In this context, it denotes the ability to change processes, without completely replacing them.

Totally rigid software processes, PML and supporting PSEE have long proven to be not effective for the dynamic nature of software [2]. On the other hand, allowing for total process flexibility (i.e., 100% changeable) threatens the original purpose of process models, which is to provide guidance for software project planning and execution. In fact, enabling software team members to perform (and describe) unlimited and unclassified changes to software processes hampers seriously the learning and reuse of this information in the future. Moreover, in the everyday business practice, most people do not want to have much flexibility, but would like to follow very simple rules to complete their tasks, making as little decisions as possible [4], [5].

To corroborate this, case studies on flexibility in software processes (e.g., [6]) make evidence on the need of having (senior) process participants expressing and controlling the changes that other process participants are advised to make in software process models, instances and, consequently, real-world software projects. This *controlled flexibility* can be defined as *the ability to express and control, by means of a PML and resulting process models, which, where*

Manuscript received June 22, 2009.
Manuscript revised April 1, 2010.

[†]The author is with the Department of Informatics Engineering, School of Technology and Management, Polytechnic Institute of Leiria, Portugal.
[††]The author is with the Department of Informatics, Faculty of Sciences, University of Lisboa, Portugal.
[†††]The author is with the Department of Engineering, University of Trás-os-Montes e Alto Douro, Portugal.
a) E-mail: rmartin@estg.ipleiria.pt
b) E-mail: dulce@di.fc.ul.pt
c) E-mail: jvarajao@utad.pt

DOI: 10.1587/transinf.E93.D.2190

and how certain parts of a software process can change, while keeping other parts stable [5].

Therefore, the first step on deriving a controlled flexibility-aware PML is to provide a set of understandable concepts and relationships associated with this domain of knowledge. These should be easily learnt and shared among process engineers and software development team members. We provide, in this paper, a set of core concepts and relationships that pertain to the modelling of controlled flexibility in software processes. We achieve this purpose by using Concept Maps (Cmaps) [7] that capture these concepts and relationships through diagrammatic representations. We also provide detailed descriptions about these maps. The purpose is to form a sound knowledge base on controlled flexibility, to be shared between process engineers and software team members. The concepts and relationships presented in the proposed Cmaps are to be latter incorporated into a PML, in order to enable process engineers the modelling of controlled flexibility, i.e., advice on which, where and how changes can be made to software processes. Cmaps will also enable a better understanding for software team members to follow those advice and perform changes accordingly.

This paper is organised as follows: Sect. 2 introduces Cmaps and Sect. 3 presents the main diagrams for the proposed maps, along with descriptions of the key concepts and relationships. Section 4 analyses the most relevant related work on flexibility conceptual frameworks, taxonomies and corresponding applications. Finally, Sect. 5 concludes the paper and presents future work.

2. Ontologies and Concept Maps

The use of ontologies pursuit a standard way of specifying content-specific agreements for the sharing and reuse of knowledge [8]. 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 [9]. An *ontology* is an explicit specification of a conceptualisation [8].

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 of the knowledge it defines. If it is to be shared among humans, it can assume a less formal and human-centred representation, such as the diagrams and tools associated with Concept Maps (Cmaps) [7]. If the reasoner is automated (e.g., a software component), then a more formal and machine-centred language applies, such as the Resource Description Framework (RDF) and Web Ontology Language (OWL) [10] web standards associated with the semantic web.

According to Novak and Cañas [7], a *Concept Map* 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 between concepts represented by a connecting line linking two concepts. Lines have also a textual representation (referred to as *linking words* or *linking phrases*), which specify the relationship between the two concepts. *Concept* is defined as a perceived regularity in events or objects, or records of events or objects, designated by a label. This label is usually composed by one or more words, but can also assume other textual symbols, such as + or %. *Propositions* are statements regarding objects or events in the universe, either naturally occurring or constructed. They form a meaningful statement about the relationships that exist between two or more concepts.

Relationships may be classified under *static* and *dynamic*. Static relationships between concepts help to describe, define, and organise knowledge for a given domain. Classifications and hierarchies are usually captured in relationships that have a static nature and indicate belongingness, composition, and categorisation.

A dynamic relationship between two concepts reflects and emphasises the propagation of change in these concepts. It shows how change in quantity, quality, or state in one concept causes change in quantity, quality, or state of the other concept in a proposition.

These are the basic foundations of Cmaps. We apply them in the next section to capture the knowledge domain of modelling controlled flexibility within software processes.

3. Capturing the Knowledge Involved in Modelling Controlled Flexibility

In order to provide the context of use of Cmaps in this work, let us consider a process engineer that analyses the results of a certain set of real-world software projects, developed under a certain software process model. S/he is able to identify the deviations of project executions regarding that process model, and derive the need of considering a new *controlled flexibility*-related concept. S/he wants to incorporate the concept in the PML that s/he uses to design future process models, in order to decrease deviations in projects' executions.

However, the concepts comprised in PMLs and resulting process models should be shared, discussed and agreed upon by all process participants (process engineers and software team members included), since all of them will use process models as the preferred media for process communication, understanding and guidance [11]. Summing up, they need to share domain knowledge on how to control flexibility in software processes, before incorporating it in PMLs and derived process models.

We adopted Cmaps and CmapTools [7] as a less formal but proven method for explaining and communicating domain knowledge [12]. CmapTools has been developed over the previous decade as an intuitive, human-centred computer interface for creating and managing concept maps. We use it to create and manage the relationships between concepts that address the need of controlling changes within software processes. Our goal does not comply, for the time being, automated software mechanisms as possible process participants for the sharing of this domain knowledge. Nevertheless, CmapTools includes export and import features for

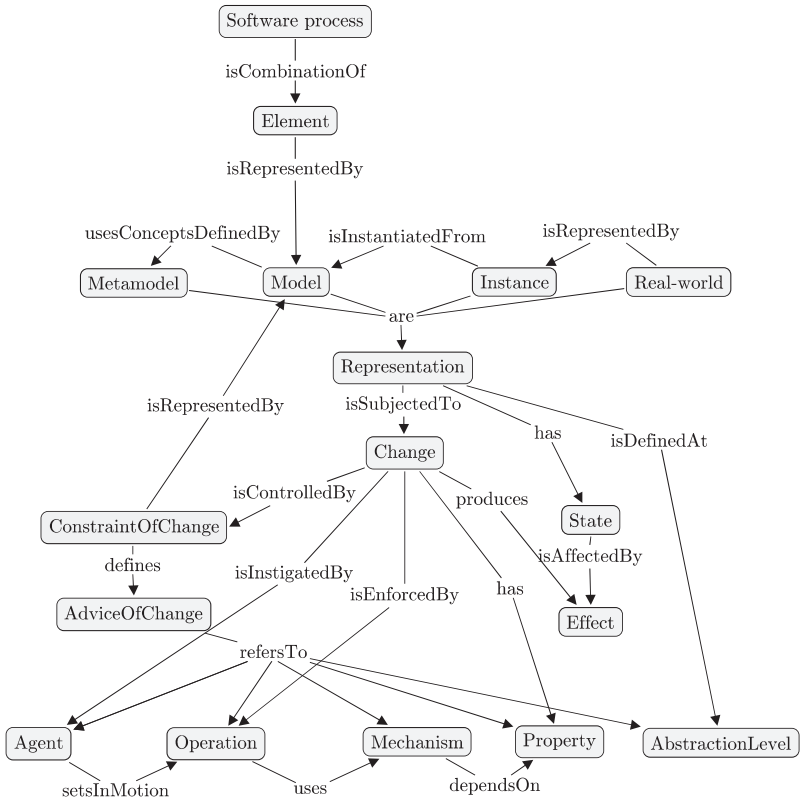

Fig. 1 Core concept map for sharing knowledge on process controlled flexibility.

ontologies represented OWL-derived languages, which provides us the means for a latter and more formalised ontology approach.

Besides creating and editing Cmaps, process engineers and software development team members can use CmapTools' *publishing* and *sharing* features which allow for the compositions of new concepts and relationships through the reuse of existing ones stored in online repositories.

To start the construction of our Cmaps, we begin by providing in the next section a *focus question*.

3.1 Focus Question

Recent studies advocate that, when a Cmap is constructed with a particular purpose and for a certain audience, its objective is better achieved by providing a *focus question*, to which the Cmap will be designed to answer [7]. Moreover, and to prompt dynamic thinking and dynamic relationships between concepts, the question should be of type *"How does the concept X work?"*, rather than *"What is concept X?"*.

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

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

3.2 Diagrams

In this section we present the diagrams for the concepts and relationships that provide answers to the focus question. We begin by illustrating in Fig. 1 the main Cmap for this purpose. As we carry along describing its containing concepts, we also provide helpful descriptions and examples on related contexts. The bases for these concepts and relationships rely on several contributions from most prominent theoretical, empirical and practical research works on business and software process management. These are referred throughout this section and complemented with a *controlled-flexibility* context. We also propose new concepts for this context.

Figure 1's diagram shows that a *software process* is a combination of *elements* represented by a *model*. These elements can be used to express correlated process perspectives, including [13]:

- *Functional* - what process elements are being performed (e.g. *phase*, *activity* and *step* elements);
- *Behavioural* - when process elements are performed (e.g. control flow nodes such as *fork*, *join*, *decision* and *merge* nodes, as also iterative/parallel region elements);
- *Organisational* - where and by whom (which agents) in the organisation process elements are performed (e.g. *role* elements, and resources like physical communication mechanisms, physical media and location used for

storing entities); and
- *Informational* - informational entities produced or manipulated by a process (e.g. *data*, *artifact*, *work product* (intermediate and end) and *object* elements). It includes both structure of informational entities and the relationships among them.

Back to our diagram, a process element *model* depends on its *metamodel*, which establishes its structure of concepts, relationships and constraints. An overall process metamodel usually defines a Process Modelling Language (PML), where all process modelling elements are specified [14]. The Software & Systems Process Engineering Metamodel (SPEM) [15] is an example of a software process metamodel which defines Unified Modelling Language (UML) Activity Diagrams (AD) as the core PML. In turn, SPEM and UML rely on the Meta Object Facility (MOF) as their own metamodel, provided by the Object Management Group (OMG) in [16].

Process models are then created as instances of the metamodel, and include more or less specified arrangements of process element model representations, such as activities, resources and resulting work products. Examples of software process models include the (textual and graphical/diagrammatic) process representations comprised by well known software methods such as the Unified Process (UP) [17].

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 [14].

We introduce *metamodel*, *model*, *instance* and *real-world* as process element *representations*, defined at distinct but correlated *abstraction levels* of modelling. These representations are subjected to *changes*, which in turn have *effects* that can affect their *states* [18]. For instance, Casati et al. [19] present a set of strategies to migrate process instances that were changed during execution.

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

Properties of change are not dependent on a process element's type, but characterise multiple and general dimensions of a change. Concrete properties of change commonly referred in literature include [20], [21]:

- *Extent* of change, denotes whether a change is only introduced to an already existing process model (*incremental* change), or abolishes it and creates a completely new one (*revolutionary* change). Often experts are required to do revolutionary changes to the whole or part of a process model;
- *Duration* of change, states that changes can be *temporary* or *permanent* changes. Temporary changes are valid for a limited period of time, and permanent changes are valid until the next permanent change;
- *Swiftness* of change, expresses whether changes are to be applied *immediate*ly to all process model instances (also the running ones), or *deferred* only to new instances of the changed process model;
- *Anticipation* of change, defines whether the change is *planned* or *ad-hoc*. Ad-hoc changes are often made to tolerate exceptional situations, and planned changes are often part of a process redesign.

Operations use *mechanisms* in order to be enforced. For example, executing an *add* operation on a process model implies the use of a software tool that, besides supporting process editing features, also provides verification of conformance, consistency, and compliance rules associated with that operation. All these resources can be considered as mechanisms of change [18]. These mechanisms can depend on the properties that are desired for a change to have. For example, if the former *add* operation was to be valid during two weeks (*duration* property of change), the mechanism(s) used to support it would have also to comprise this property.

Changes are instigated (put into action) by *agents* of change. In the software process context, the agent of change is responsible for setting into motion *operations* that will result in an *effect* of change endured by one or more process element representations.

Agents of change may be software components that automatically change process element representations under some criteria, or humans such as software process engineers, project managers, analysts, designers, programmers and testers, that need to change/adjust software process representations.

A change can be controlled by *constraints*, which are also represented by *models* that the process engineer configures. Constraints define *advice of change*, which refer to the abstraction level, operations, properties, mechanisms and/or agents that should be considered when changing a certain process element representation. Advice on a change can be 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 *changes made at the process model are recommended to be reflected immediately to all corresponding running instances*. The modelling of this constraint can be made by composing a tuple with semicolon separated values

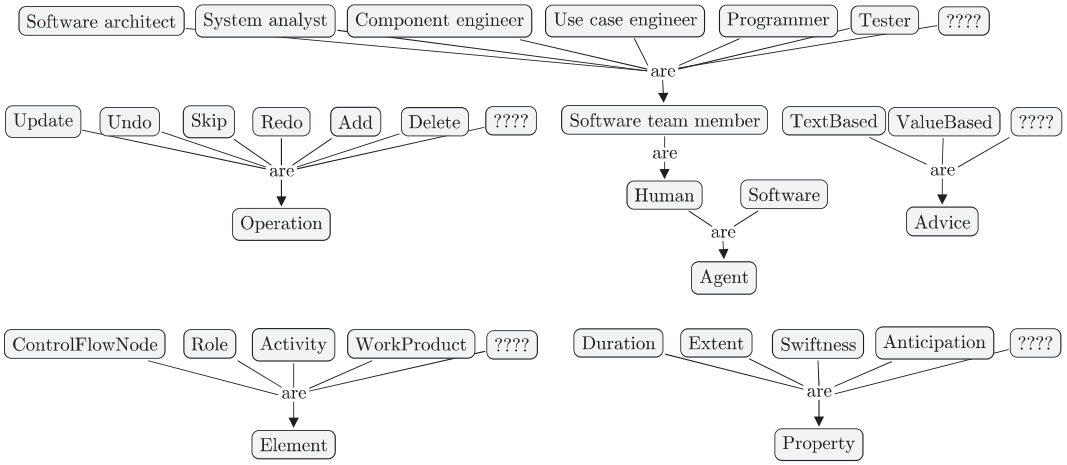

Fig. 2 Generalisation relationships for some of the concepts illustrated in Fig. 1.

of the form *(abstraction level representation: <String>; change property name: <String>, change property value: <ChangePropertyValue>; advice type: <AdviceType>, advice value: <AdviceValue>)*, with the values *(abstraction level representation: `"model"`; change property name: `"swiftness"`, change property value: `immediate`; advice type: `TextBasedAdvice`, advice value: `"recommended"`)*.

Figure 2 represents some generalisation relationships pertaining concepts already presented in Fig. 1 and referred above. Some concepts are filled with four question marks. This means that the general concept can have more or different specialised elements beyond the examples presented, since they can depend on factors such as the modelling language, application domain, tool support and/or organisational context.

4. Related Work

One of the most prominent foundational research works on software process flexibility is the one presented by Madhavji [22] and his Prism model of changes. He has identified several items strongly related with the software development environment, which are subjected to continuous change. The basic items of change are people, policies, laws, resources, processes and results. For various predictable and unpredictable reasons, such items may need to be changed on an ongoing basis. Therefore, the Prism model of change included the following features:

- Separation between changes made to the described items and changes made to the environmental facilities encapsulating those items;
- A *dependency structure* that describes the various items and their interdependencies. It is used to identify the items affected by a given change;
- A *change structure* that classifies, records and analyses change-related data, and makes qualitative judgement of the consequences of a change;
- Identification of the many distinct properties of a change;

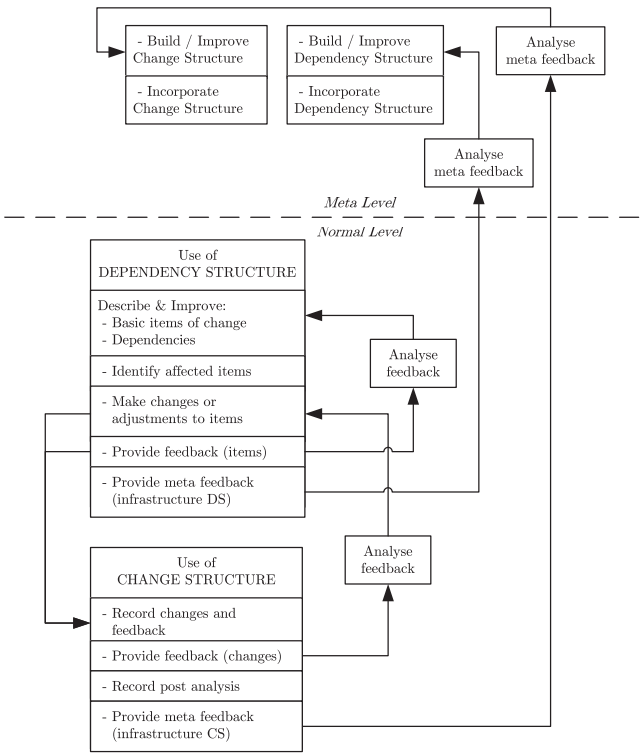

Fig. 3 The PRISM Model of changes (adapted from [22]).

- A built-in mechanism for providing feedback on changes made.

Figure 3 illustrates the Prism model of change with a two-level approach. Both dependency and change structures are to be use at the normal level. It is at the meta level that these structures can be added, refined and improved through meta feedback mechanisms that come from their use in the normal level.

In spite of being an important contribution to characterise change, it does not comply a way for process engineers to model some control of changes, based on feedback

information from software process runs.

The business process research area has been also contributing to clarify process flexibility. According to a recent taxonomy proposed in [20], process flexibility can be classified in three orthogonal dimensions:

1. *Abstraction level of change*, that distinguishes *where* changes are to be made, i.e., if at the *type* or *instance* levels (or both);
2. *Subject of change*, representing *which* modelling elements are to be changed, and, consequently, which related perspective(s) (such as the ones in proposed in [13], and mentioned above in Sect. 3.2);
3. *Properties of change*, denoting *how* can a modelling element be changed. Examples include properties mentioned above in Sect. 3.2, such as *extent*, *duration*, *swiftness* and *anticipation* of change.

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

More recently, Schonenberg et al. proposed in [21] another taxonomy on process flexibility. Four distinct approaches are identified, each having its own application area. They are *flexibility by*:

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

Each flexibility type operates in different ways. Figure 4 provides an illustration of the distinction between these types in isolation, in terms of the time at which the specific flexibility options need to be configured - at design time, as part of the process model or at runtime via facilities in the process execution environment.

Although our purpose does not include an explicit classification of types of flexibility, this work by Schonenberg et al. contributed to refine the scope of several change concepts and relationships of our Cmaps, namely the *mechanism*, *property*, *operation* and *abstraction level* concepts.

Constraints of change in process models are analysed in [23]. The authors propose modelling tool artifacts to support correctness, compliance and consistency constraint

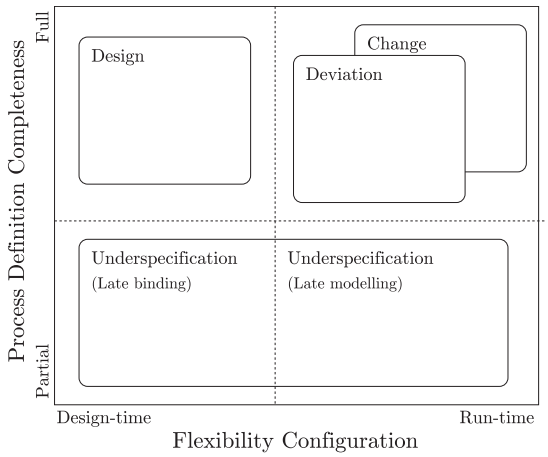

Fig. 4 Flexibility type spectrum (adapted from [21]).

modelling and checking. They have developed a simple diagrammatic language named Business Process Compliance Language (BPCL) which a process engineer can use to specify constraints to which process representations have to comply.

This work has contributed to develop our *constraint of change* concept. It also provides us several ideas for future work, namely regarding tool support for the (syntax and semantic) analysis on the modelling and checking of controlled changes.

Although all these works are valuable for refining the process flexibility scope, they do not share a common set of inherent concepts. In fact, we can observe that there are different descriptions, perspectives and classifications for the concepts involved (see, e.g., the concept of *abstraction level of change*, which fall into different classifications in [21], [20] and [22]).

The concepts associated with controlled flexibility in software processes follow similar goals than the ones regarding the *process rationale modelling* approach. Here, process models are changed with the help of information captured on the rationale behind those changes. Specifically considering software processes, the authors in [24] propose a set of concepts to be used in the rationale behind the changes made to process models. They advocate that these concepts provide a means for better understanding process evolution.

However, this rationale approach only concerns to changes made to process *models*. It does not consider the possibility of a rationale behind change made to *metamodel*, *instance* or *real-world* representations of software processes. Moreover, while this approach uses past information on changes made to process models and the rationale behind them, our (proactive) *controlled flexibility* approach is concerned on providing advice about the changes that software team members can or cannot realise in a near future, within process representations.

We could not find any related work on concept maps for process flexibility, nor any kind of specific (sub) taxon-

omy of concepts related with the control of that flexibility. The taxonomies and approaches presented here as related work are a mix between flexibility concept descriptions and flexibility classifications, where relationships between the concepts described/classified are not always clear.

Summing up, by using Cmaps we complemented some of the concepts on process flexibility already presented by these related works. We also clarified the relationships between those concepts, by using Cmaps diagrams and presenting several application examples. Besides this, we proposed totally new concepts and relationships, namely: *Representation* and the fact that any type of process element representation can be subjected to a *Change*; *AdviceOfChange* and its reference to *Agent*, *Operation*, *Mechanism*, *Property* and *AbstractionLevel*; and the fact that *Change* is controlled by *ConstraintOfChange*, which in turn includes an *AdviceOfChange*.

5. Conclusions and Future Work

We provide in this paper concept maps for the modelling of controlled flexibility in software processes. The main purpose of these maps is to establish a sound knowledge base to enhance process engineers and software team members' understanding of process flexibility and how to control it.

Process engineers and software development team members can use CmapTools and related online repositories to share and discuss about *controlled flexibility*-related concepts. The intent is to latter translate those concepts onto semantic controlled flexibility-aware language constructs of a PML. These are to be later used to express controlled flexibility in software process representations. If the adopted core PML's metamodel is flexible enough to be easily integrated with these new concepts, process engineers and software team members can use CmapTools and Cmaps as a primary source to continuously refine the PML, regarding *controlled flexibility*-aware language constructs.

For this matter, and to provide means for future evaluation on our approach, we have used the presented maps as a conceptual basis to extend UML Activity Diagrams (AD) as a *controlled flexibility*-aware PML. We have been working on this through the use of a UML profile of our own called FlexUML (see [25] for further details).

We have also implemented the extended UML AD on a PSEE tool called FlexEPFC (see [26] for further details), which is based on the IBM's Eclipse Process Framework Composer (EPFC[†]) tool. This enables process engineers to define and publish software process models with additional (textual/graphical) controlled flexibility information through the use of the FlexUML profile stereotypes. Other software team members can then visualise and learn about this information, and change process model and/or instance representations accordingly.

Future work includes using structured case studies' analysis, according with the guidelines provided by Kitchenham et al. [27]. Currently, our proposed solutions (Cmaps, extended UML AD and FlexEPFC) have only been evaluated by a limited audience. We would like to extend our evaluation with feedback from the real-world software organisations and people, in order to get more objective data.

[†]http://www.eclipse.org/epf

References

[1] V. Gruhn, "Process-centered software engineering environments, a brief history and future challenges," Annals of Softw. Eng., vol.14, no.1-4, pp.363–382, 2002.
[2] G. Cugola, "Tolerating deviations in process support systems via flexible enactment of process models," IEEE Trans. Softw. Eng., vol.24, no.11, pp.982–1001, 1998.
[3] M. Reichert, S. Rinderle-Ma, and P. Dadam, "Flexibility in process-aware information systems," LNCS Trans. Petri Nets and Other Models of Concurrency (ToPNoC), vol.2, pp.115–135, 2009.
[4] I. Bider, "Masking flexibility behind rigidity: Notes on how much flexibility people are willing to cope with," Proc. 17th Intl. Conference on Advanced Information Systems Engineering (CAiSE), pp.7–8, 2005.
[5] S.E. Borch and C. Stefansen, "On controlled flexibility," Proc. 7th Workshop on Business Process Modeling, Development and Support (BPMDS), pp.121–126, 2006.
[6] A.G. Cass and L.J. Osterweil, "Process support to help novices design software faster and better," Proc. 20th IEEE/ACM Intl. Conf. on Automated Software Engineering (ASE), pp.295–299, 2005.
[7] J.D. Novak and A.J. Cañas, "The theory underlying concept maps and how to construct and use them," Tech. Rep., IHMC CmapTools, 2006-01 Rev 2008-01, Florida Institute for Human and Machine Cognition, 2008.
[8] T.R. Gruber, "Toward principles for the design of ontologies used for knowledge sharing," Int. J. Human-Computer Studies, vol.43, no.5-6, pp.907–928, 1995.
[9] M.R. Genesereth and N.J. Nilsson, Logical foundations of artificial intelligence, Morgan Kaufmann Publishers, San Francisco, CA, USA, 1987.
[10] W3C, "OWL web ontology language guide — W3C recommendation 10 February 2004," Tech. Rep., W3C (MIT, ERCIM, Keio), 2004.
[11] I. Sommerville, Software Engineering, 8th ed., Addison-Wesley, 2006.
[12] R.R. Hoffman and D.D. Woods, "Studying cognitive systems in context: Preface to the special section," J. Human Factors and Ergonomics Society, vol.42, pp.1–7, 2000.
[13] B. Curtis, M.I. Kellner, and J. Over, "Process modeling," Commun. ACM, vol.35, no.9, pp.75–90, 1992.
[14] M. Heller, A. Schleicher, and B. Westfechtel, "A management system for evolving development processes," Proc. 7th Intl. Conference on Integrated Design and Process Technology (IDPT), 2003.
[15] OMG, "Software process engineering metamodel specification, v2.0," Tech. Rep., Object Management Group, 2007.
[16] OMG, "Meta object facility v2.0," Tech. Rep., Object Management Group, 2006.
[17] I. Jacobson, G. Booch, and J. Rumbaugh, The Unified Software Development Process, Addison-Wesley Longman Publishing, Boston, MA, USA, 1999.
[18] A.M. Ross, D.H. Rhodes, and D.E. Hastings, "Defining changeability: Reconciling flexibility, adaptability, scalability, modifiability, and robustness for maintaining system lifecycle value," J. Systems Engineering, vol.11, no.3, pp.246–262, 2008.
[19] F. Casati, S. Ceri, B. Pernici, and G. Pozzi, "Workflow evolution," Data & Knowledge Engineering, vol.24, pp.211–238, 1998.
[20] G. Regev, P. Soffer, and R. Schmidt, "Taxonomy of flexibility in business processes.," Input to the 7th Workshop on Business Pro-

cess Modeling, Development and Support (BPMDS), Website, June 2006. http://lamswww.epfl.ch/conference/bpmds06/taxbpflex
[21] H. Schonenberg, R. Mans, N. Russell, N. Mulyar, and W.M.P. van der Aalst, "Towards a taxonomy of process flexibility (extended version)," BPM Center Report BPM-07-11, BPMcenter.org, 2007.
[22] N.H. Madhavji, "Environment evolution: The prism model of changes," IEEE Trans. Softw. Eng., vol.18, no.5, pp.380–392, 1992.
[23] R. Wörzberger, T. Kurpick, and T. Heer, "On correctness, compliance, and consistency of process models," Proc. 17th IEEE Intl. Workshops on Enabling Technologies: Infrastructures for Collaborative Enterprises (WETICE), 2008.
[24] A. Ocampo and J. Münch, "Rationale modeling for software process evolution," Software Process: Improvement and Practice, vol.14, no.2, pp.85–105, 2009.
[25] R. Martinho, D. Domingos, and J. Varajão, "FlexUML: A UML profile for flexible process modelling," Proc. 19th Intl. Conf. of Software Engineering and Knowledge Engineering (SEKE), pp.215–220, 2007.
[26] R. Martinho, J. Varajão, and D. Domingos, "A two-step approach for modelling flexibility in software processes," Proc. 23rd IEEE/ACM Intl. Conf. on Automated Software Engineering (ASE), 2008.
[27] B. Kitchenham, L. Pickard, and S.L. Pfleeger, "Case studies for method and tool evaluation," IEEE Softw., vol.12, no.4, pp.52–62, 1995.

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

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

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