SPMDSL Language Model
Onto a DSL for Agile Use Case Driven Software Project's Management

Gilberto G. Gomes Ribeiro[1], Ângela M. Amorim Barros[1] and António M. Rosado da Cruz[1,2]
[1]*Polytechnic Institute of Viana do Castelo, Av. do Atlântico, s/n, Viana do Castelo, Portugal*
[2]*Algoritmi Research Centre, University of Minho, Portugal*

Keywords: DSL, Domain Specific Language, Ontology, Agile Use Case-driven Software Project Management.

Abstract: Project management involves applying knowledge, skills, tools and techniques to project activities to meet the project requirements. Each project's unique nature implies tailoring that knowledge, skills, tools and techniques to adapt the management activities to cope with project constraints. Management and technical activities meet at some points, namely on activities that have technical and management relevance. This paper proposes SPMDSL and presents its language model and the domain analysis made during its development. SPMDSL aims to be a DSL defining a set of representational primitives with which to model projects in the domain of agile software project management. These primitives are represented as classes and their interrelationships. The proposed DSL focuses on agile use case driven software development project management, and so it also integrates concepts from software modeling. The goal is to enable representing past projects' information to facilitate retrieving information for lessons learned analysis.

1 INTRODUCTION

Software and other domain's projects involve one or more sequences of technical activities and project management activities. These sequences of activities intersect at some points, on activities with technical and management relevance. Examples of such intersecting activities can be found in the process of requirements analysis and specification or in the quality control process, among other processes.

According to the PMBOK Guide (PMI, 2013), project management (PM) is the application of knowledge, skills, tools and techniques to project activities to meet the project requirements. Managing a project typically includes (PMI, 2013):

- Identifying requirements;
- Managing the needs, concerns, and expectations of the stakeholders;
- Managing and carrying out communications with the stakeholders;
- Balancing the competing project constraints, namely: Scope, Quality, Schedule, Budget, Resources, and Risks.

The project manager needs to focus its efforts on the project constraints that affect more a specific project. Each project has different characteristics and conditions that determine the constraints that need more attention. Project managers make use of software tools for helping them manage projects according to several dimensions (e.g.: time, cost, scope), which are bound by project constraints. A software tool for assisting in managing projects must allow the project manager to realize the full project workflow, enabling a better resource management, and helping manage the different project constraints.

Software projects also involve technical activities, which typically follow a predefined process. Development processes are, nowadays, typically iterative in nature, where requirements, plans, tasks, and their inner priorities, are continuously assessed and refined. Agile Methods are the best examples of this kind of processes.

Modern software development processes are also typically use case or user story driven. Herein, the terms use case and user story are used interchangeably, because both describe system's features from the users' point of view, and for our purposes that supersedes any minor difference between the two concepts. From the project's point of view, use cases define a unit of functionality that is selectable for development in each iteration. A use case is then as important for the product as it is for the process that develops it.

Ribeiro, G., Barros, Â. and Cruz, A.
SPMDSL Language Model - Onto a DSL for Agile Use case driven Software Projects' Management.
DOI: 10.5220/0006326804030410
In *Proceedings of the 19th International Conference on Enterprise Information Systems (ICEIS 2017) - Volume 2*, pages 403-410
ISBN: 978-989-758-248-6

This paper presents SPMDSL core language model. The core language model is in fact an Ontology for that knowledge domain. SPMDSL intends to be a domain specific language (DSL) for the management of agile use case driven software projects, based on the good practices embodied in the PMBOK Guide (PMI, 2013) and Agile Methods, without overlooking concepts and good practices in software engineering, namely from the Unified Modeling Language (UML). The presented DSL focuses on the main concepts of both PM and software engineering domains. The DSL engineering process is tailored from the one proposed by Strembeck and Zdun (2009).

The rest of the paper is structured as follows: next section overviews DSL artifacts and the process of building a DSL. Section 3 briefly presents PMBOK (PMI, 2013), the key Agile methodologies, the main PM tools and existing proposals to PM ontologies. The goal is to pave the road for eliciting the main concepts and their interrelationships that forms the SPMDSL core language model presented in section 4, together with the appropriate constraints. Section 5 concludes the paper and presents some ideas for future work.

2 DEVELOPING A DSL

2.1 DSL Engineering

A DSL is a "tailor-made (computer) language for a specific problem domain" (Strembeck and Zdun, 2009). A DSL comprises the development of:

- A Language model (Abstract Syntax), defining the concepts and relationships (the core language model), as well as a set of rules enforcing well-formedness through static semantics (model constraints). This may be defined by an Ontology or Metamodel.
- A Concrete Syntax, establishing a textual or graphical concrete notation for the language, which defines the allowed language constructs and phrases (Moody, 2009).
- A semantics for the allowed language phrases: Giving meaning to those phrases, either formally, through denotational semantics, or more informally, through illustrative examples of language usage and informal explanations of those examples, among other ways.

DSL development involves an iterative lifecycle with four activities (Strembeck and Zdun, 2009):

1. Defining the DSL's core language model and model constraints;
2. Defining DSL language elements' behavior;
3. Defining the DSL's concrete syntax(es);
4. Integrating DSL artefacts with the platform.

Each of these main activities is in itself a subprocess that may be tailored to better meet the influencing factors identified by Strembeck and Zdun (2009). In this paper, we focus on defining the DSL's core language model and model constraints, including reporting the domain analysis activities.

2.2 Tailored Subprocess for Defining the DSL's Language Model

This subsection presents the tailored subprocess "Defining the DSL's core language model and constraints", following (Strembeck and Zdun, 2009). The tailored subprocess is depicted in Figure 1, and comprises the following activities:

1. Domain Analysis: Covers the analysis of the problem domain, in this case the standards and practices in PMBOK and main Agile methods for finding domain concepts, and the identification of corresponding domain abstractions, that take the form of language elements added to the DSL being created;
2. Analysis of existing platforms and tools: Identifying elements on existing PM tools and defining or deriving language elements for the DSL, if they aren't already defined.
3. Integrate Domain Abstractions/Language Elements to ensure that the defined DSL is not redundant (there is no overload of language elements) and does not have neither deficit nor excess of language elements (Moody, 2009).
4. Define Language Model Constraints: Identify rules and constraints in the domain, and reflect them as Model Constraints.
5. Check Language Model: The language model, comprising the core language model and the constraints, must be checked for completeness and correctness from a domain-oriented perspective (Strembeck and Zdun, 2009).

In this work, domain knowledge is gathered both inductively (bottom-up), by identifying elements in existing PM tools and existing ontologies, and deductively (top-down), by identifying domain abstractions from the real world, embodied in the PMBOK and the Agile Methodologies (Voelter, 2013). Then, the domain abstractions are associated to existing or new language elements, and elements from existing platforms and ontologies are used to derive new language elements. These elements are related to each other, also per the gathered knowledge, and together form the DSL being built.

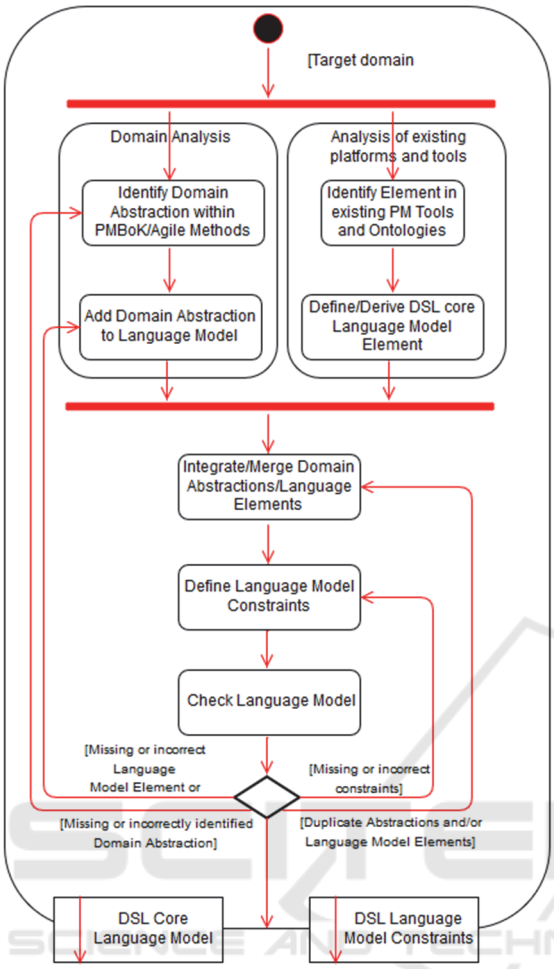

Figure 1: The subprocess for defining the DSL language model, tailored from (Strembeck and Zdun, 2009).

3 ANALYSIS OF DOMAIN AND OF EXISTING PM TOOLS AND ONTOLOGIES

The process of constructing a DSL must gather concepts and concept interrelationships of the knowledge domain it addresses. The PMBOK Guide (PMI, 2013) enfolds knowledge and good practices in PM and forms a body of knowledge for the PM domain. The next subsection introduces PMBOK.

Additionally, Agile Methods are used today in almost every software project. Agile methods share many concepts, useful for describing and discussing software projects. As such, the domain analysis for UC driven software projects must also address Agile Methods, which are overviewed in subsection 3.2.

The study of existing applications in the PM domain is also essential, so that the concepts already in place in existing PM tools may be taken into consideration. Subsection 3.3 outlines five PM tools. Subsection 3.4 presents existing approaches to domain ontologies related to project management.

3.1 The PMBOK Guide

The PMBOK guide (PMI, 2013) includes knowledge and good practices about project management, which have application in "most projects most of the time", and their value and usefulness is object of consensus. The knowledge described is not directly and uniformly applicable to all projects, being the responsibility of each organization/project manager to select and adapt what is applicable in each particular project. The PMBOK guide does not prescribe a process model, but rather a set of process groups that may be selectively adapted to any process model, including agile processes.

The PMBOK Guide also promotes a common vocabulary within the PM domain, having published the PMI Lexicon of Project Management Terms (PMI, 2015), which provides a set of vocabulary that can be consistently used by project managers and other stakeholders. That vocabulary is not, however, an ontology as it does not focus on PM main concepts nor addresses their relationships. It mainly focuses on concepts and terms used in the PMBOK Guide when describing tools and techniques within each process or process activity.

3.2 Agile Methodologies

In 2001, a group of developers established a set of 12 principles and called it the "Agile Manifesto" (http://agilemanifesto.org), which focused more on people than on processes. Those principles are the basis to many variations of agile methods. Agile methods involve a process definition and a set of tools and techniques. Process and tools envisage to embrace the Agile principles, whichever the specific Agile method they belong to. Practice has leveraged some Agile methods in detriment of other less used ones. Scrum and XP (eXtreme Programming) are commonly accepted as two of the most currently used agile methods. XP can be characterized by short development cycles, incremental planning, continuous feedback, reliance on communication, and evolutionary design (Beck, 2004). Scrum focuses on the team work rather than a set of specific software development rules. What Scrum and XP have in common is a prioritized list of

features, short iteration cycles for developing selections of those features, frequent deliveries of running software and an approach to development that promotes developing simple yet needed features, instead of nice to have but unrequired ones (Beck, 2004; Schwaber and Sutherland, 2016).

Besides Scrum and XP, other Agile methods made their way at different levels of adoptability. Examples are Feature Driven Development (FDD) and Extreme Modeling (XM). Unlike other methodologies, FDD does not cover the entire development process but rather focuses on the design and building phases (Palmer and Felsing, 2002). FDD also has frequent deliverables, along with accurate monitoring of the progress report. Its five sequential steps include developing an overall system model, which models requirements of the whole system being developed, and progresses together with the system (De Luca, n.d.). A domain classes model is built, identifying attributes, relationships and methods. FDD also develops a prioritized features' list to support the requirements. Each feature identifies actions over objects of the domain model, and new or changing requirements are modeled back into the overall model. New or changed features arise from the updated model.

"Model first, code later" is the main idea behind Extreme Modeling (XM). XM claims that modeling can be something done on the spur of the moment whereas coding might not be as immediate or practical, making the feedback not as easy to get. By modeling first, the software developer can analyze the problem prior to coding the solution. XM and XP share the same values and principles. The main practices of XM are model to communicate and model to understand (Ambler, 2001).

3.3 Project Management Tools

This subsection presents five PM tools, which are compared in terms of features and concepts elicited from the PMBOK, but also from typical software development process models, such as the Unified Process and Scrum. The studied tools are Microsoft Project (https://products.office.com/en/Project), MSP, the most used PM tool, and the open source tools Open Project (openproject.org), OP, LibrePlan (libreplan.com), LP, Agilefant (agilefant.com), AF, and Redmine (redmine.org), RM.

In software project management, requirements are usually modeled through use cases or user stories, and these typically drive the software development activities, by being divided into work items, and these further divided into work tasks. This way, use cases are associated to project work tasks, as the latter are defined by the need to implement and satisfy the former.

MSP, LP, AF, OP, and RM allow an effective project management, enabling some good practices underlined by PMBOK. Task (work package) management is supported, along with their hierarchical structuring, enabling the establishment of a Work Breakdown Structure (WBS). The first two do not support the concepts of Requirement or Use Case/User Story, which commonly leads to confusion between requirements and project tasks.

All the surveyed tools allow defining task dependencies (e.g. follows, precedes). AF, OP, LP and RM also allow the creation of links between tasks and external documents (e.g. text documents, diagrams, images). Tasks may be associated to team members.

MSP, AF, OP and LP allow creating budgets associated to the defined project tasks, helping to keep track of the effort and cost the project activities require. Time, cost and resource management are, then, well covered by these tools.

AF and OP allow a scrum management style by enabling the definition of a product backlog (fairly equivalent to a requirements list) and sprint backlogs, along with user stories with the associated priority level and effort story points. AF enables defining dependencies between user stories. Scope and resource management are also fairly addressed, although scope change management is very poorly covered. The notion of a change requests' analysis team, envisaged in PMBOK, is absent in every tool surveyed.

Quality and risk management are not covered in OP. LP, however, has some support for risk related activities such as Monte Carlo simulation. In OP and RM, communicating project information between team members and stakeholders is centered on a Wiki. In OP, meetings may also be scheduled and messages sent through the system.

AF allows creating backlogs for different stakeholders and let them feed in ideas or feedback they have, and after validating the provided items, these can be inserted into the product backlog.

3.4 Existing PM Ontologies

Existing approaches to domain ontologies related to project management comprise PROMONT (Abels *et al.*, 2006), a PM ontology for virtual project organizations, based on German standard DIN 69901 (DIN, 2009). Another standard for PM data exchange is PMXML (Curran *et al.*, 2004).

Other PM ontology approaches include "Project metrics ontology" (www.daml.org/ontologies/349), a simple ontology focusing project metrics that enables performance metrics comparison between projects; Aramo-Immonen (2009) ontology proposal puts together a PM ontology and a project learning model, focusing project learning integrated in project processes; Sheeba *et al.* (2012) proposes an ontology to automatically classify learning materials for the Project Management knowledge domain, in order to facilitate the search for learning materials.

3.5 Unified Modeling Language

UML provides tools for generic system's analysis and design, and implementation of software-based systems (OMG, 2015). It provides a metamodel that supports modeling generic systems, especially software systems, through several diagrams that may be used to model specific views of the system. UML structural semantics provides elements for building a domain model, class model, etc.; UML behavioral semantics provides model elements for building activity diagrams, state machines, and so on; UML supplemental modeling semantic area provides elements for Use Case models, etc. (OMG, 2015).

We consider that a system model should have at least the following model views (Cruz, 2015): Structural View – modeled by a domain model, which contains the entities/concepts from the system or domain being modeled; Functionality View – modeled by a Use Case Diagram, providing a vision of the system functionality modeled as use cases; Behavioral View – provides a more or less formal specification of the system behavior; Presentational View – provides an abstract model of the system's user interface. Since UML doesn't foresee this last kind of models, screen mockups are typically used.

4 SPMDSL LANGUAGE MODEL

The language model proposed in this section, besides considering the PMBOK as a foundation for the ontology building process, incorporates concepts drawn from Agile methodologies, the studied PM tools, and existing PM ontology approaches. The proposed ontology/DSL core language model focuses on project management in use case driven software development projects, and so it also incorporates concepts from software modeling and design, namely model elements from the UML definition (OMG, 2015), although some of them have been renamed to avoid confusion with other concepts (e.g.: Actor, UseCase, DomainEntity).

4.1 Identified Language Elements and the Proposed Core Language Model

In this subsection, the main concepts and their interrelationships, elicited from the PMBOK, Agile methods, studied PM tools and existing PM ontologies, overviewed in the previous section, are put together in a diagrammatic form, to embody the proposed ontology and core DSL model. This is depicted in Figure 2 (some entity attributes are left out of the diagram, for improved readability). The rest of this section should be read with continuous reference to the ontology presented in the figure.

A Project aggregates stakeholders, requirements, work activities and tasks, that breakdown the project's work structure, use cases, domain entities, system models and screens (interaction spaces).

A Team member is a stakeholder that has a working role on the project. Working roles demand skills, which are provided by team members. Other Stakeholders may be actively involved in the project or be positively or negatively affected by it, thus having a role in the project. Stakeholders have communication needs or demands (PMI, 2013).

Project requirements, which are proposed by stakeholders, may specify a degree of importance. Requirements have a priority level of fulfilment, and may pass through a set of defined states, as depicted in the state machine diagram of Figure 3 (this is explained in the next subsection).

Common software processes are use case (UC) driven, having requirements specified as use cases. A UC represents a specific usage of the system from the user's point of view. Each UC is further detailed by including or being extended by other UCs. UCs can also be further detailed through the specification of Use Case Activities, which represent activities within a UML activity diagram, and may be constrained by a precedence order and activity preconditions. When specifying UCs, from requirements, domain entities are also identified and related with other domain entities and with UCs. Actors (users playing a role in the system) interact with UCs through interaction spaces. Within a UC, one or more domain entity is manipulated (Cruz, 2015). A project also aggregates system models, which take the form of a set of diagrams of some type. In UC driven software processes, work activities (work packages) are identified from the UCs specified. Work activities and tasks, as defined in the ontology, enable building a WBS for each UC. So, when a UC is selected for development, being included in an iteration, a set of work tasks must be started by team members, and these also have a precedence order.

SPMDSL core model also enables tracing back and forth between any project management processual or work element (e.g.: iteration, work activity, work task) and the use cases that specify the project requirements that justify it, and ultimately from the use cases to the requirements themselves.

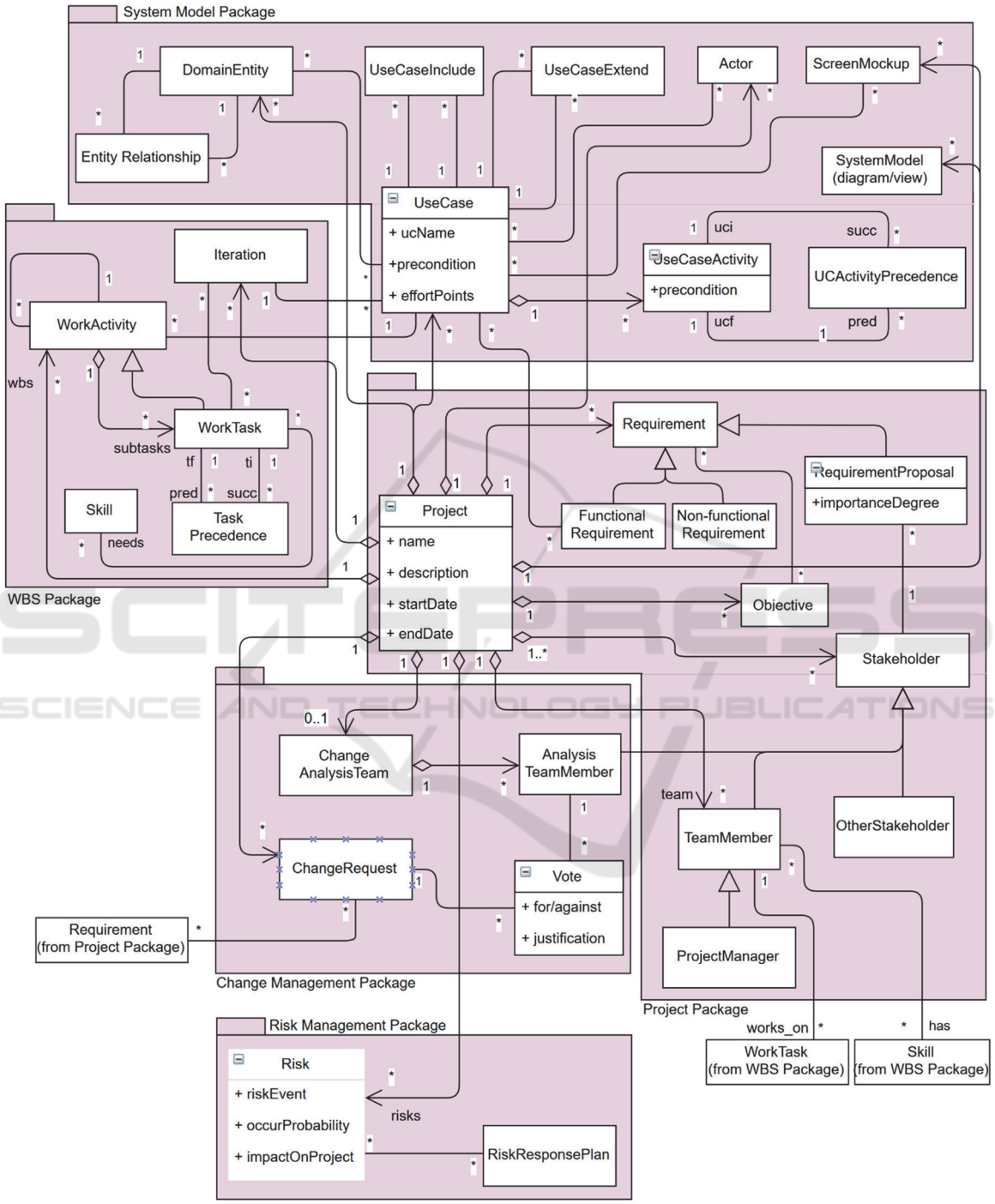

Figure 2: SPMDSL core language model/ontology.

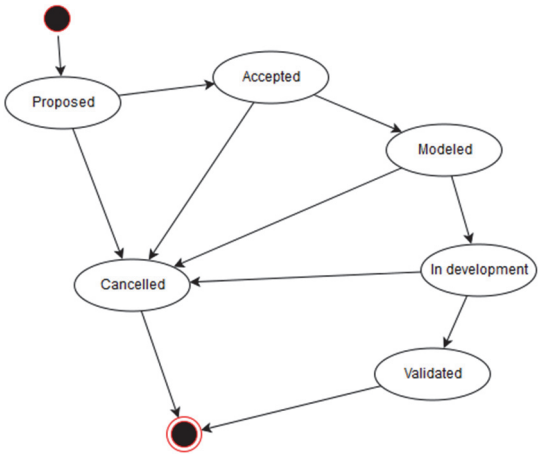

Figure 3: State-machine diagram for Requirement.

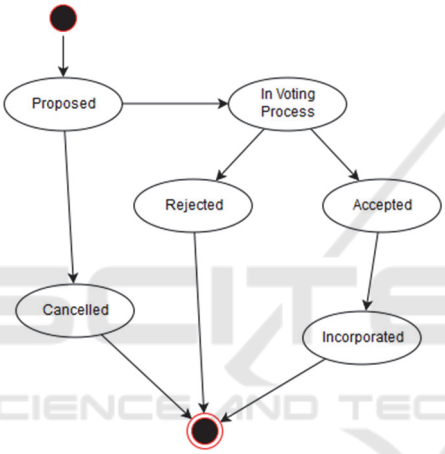

Figure 4: State-machine diagram for Change Request.

Note that the concept of use case, in a UC driven software development project, is twofold. On one hand, UCs specify user requirements. For this, they may be further refined into more fine-grained use cases and into activity diagrams, to specify system level functions or packages of functionality (Cruz, 2015). On the other hand, use cases drive the software development process, being selected for development in each iteration. For this purpose, they are detailed into work activities and work items, which represent work that must be done to develop the system for the functionality represented by the UC. SPMDSL allows this two-folded vision of UCs.

A Change Request may be proposed by any stakeholder, and may target one or more specific requirements. As advocated by the PMBOK (PMI, 2013), a project may have a Change Analysis Team, which is formed by the PM and other key stakeholders, and is responsible for analyzing change requests and deciding about their acceptance.

4.2 Model Constraints

The previous subsection presented the ontology/core language model that underlies the proposed DSL. However, for obtaining the SPMDSL language model, some constraints need yet to be defined. This subsection presents the needed constraints in the form of state-machine diagrams and OCL.

As mentioned before, project requirements may be proposed by stakeholders, and may have a set degree of importance. Requirements have a priority level of fulfilment, and they may pass through a set of defined states, as depicted in the state machine diagram of Figure 3. A Requirement Proposal is, then, a Requirement in state "Proposed".

A Change Request may be proposed by any stakeholder and, if a Change Analysis Team exists, may be voted on by members of the analysis team and, as a result of that voting process, requests can be accepted or rejected. Figure 4 shows the state-machine diagram for Change Request.

Work tasks (*WorkTask*) may have a binary precedence order between them (*TaskPrecedence*). One must make sure that a work task is not preceded by itself, in one or several steps. That is, it must be illegal to define a work task as being transitively preceded by itself. Being *t1->closure(pred.ti)* the set of work tasks predecessors reachable from task t1 in one or more steps, the following invariant constraint must then be defined:

Context WorkTask *inv:*
self->closure(wt : WorkTask |
wt.pred.ti)->excludes(self) (1)

A similar invariant must be defined in the context of *UseCaseActivity*, to prevent loops in relation to *UCActivityPrecedence*.

As mentioned earlier, Work activities and tasks allow to build a WBS (PMI, 2013), which is a tree structure decomposing work. But only the leaves of that structure, the work taks (work packages), contain actual work to be done. These leaves don't have subtasks, and the binary precedence order, *TaskPrecedence*, may only be defined on these kind of work activities, as stated by the next invariant:

Context TaskPrecedence *inv:*
self.ti.subtasks->isEmpty()
and self.tf.subtasks->isEmpty() (2)

Other constraints are needed, including one to ensure that complete use cases are selected for implementation in each iteration, and that every work package of the selected use cases must be automatically associated to the same iteration. A

work package may, though, be also associated to ulterior iterations, when they are postponed. Lack of space impedes us to present these constraints here.

5 CONCLUSIONS

This paper has focused the language model (metamodel/ontology), and its construction process, of SPMDSL, a domain specific language for the domain of agile use case driven software projects' management. Per the Ontology building methodology set by Uschold and King (1995), and the DSL development process proposed by Strembeck and Zdun (2009), the ontology capture has been based on a domain analysis which involved the identification of key concepts and relationships. This has been made essentially from the PMBOK, Agile methods, and UML. Then, from studied PM Tools and ontologies, key terms and concepts were identified and related to the previously known ones.

Guidelines for evaluating the ontology/language model built, involve making a technical judgement with respect to a frame of reference (Uschold and King, 1995; Strembeck and Zdun, 2009). For this, SPMDSL has been used as basis for developing two prototype applications for the domain of use case driven software project management (Ribeiro, 2015; Barros, 2016). A comparative discussion between the developed prototype and the studied PM tools can be found in (Ribeiro, 2015).

The proposed DSL enables the description of software project management artefacts, facilitating the archiving and easy retrieval of these artefacts for closed projects, contributing to a more effective sharing of lessons learned and good practices from previous projects within an organization or among organizations in the software projects domain.

SPMDSL's language model also aims to contribute for establishing a common language for use case driven software projects' management and to contribute to a complete Software PM Ontology.

Besides a well-defined Language Model, a DSL needs a concrete language syntax or notation. SPMDSL concrete notation is based on XML. This suffices for representing past projects' information and serving as an exchange language between software PM tools. Possible future research directions may include the establishment of a more human-readable concrete notation for SPMDSL.

REFERENCES

Abels, S., Ahlemann, F., Hahn, A., Hausmann, K., Strickmann, J., 2006. PROMONT - A Project Management Ontology as a Reference for Virtual Project Organizations. In *OTM 2006 Workshops* (vol. 1, pp. 813-823), Montpellier, France, Oct 29 - Nov 3.

Ambler, S.W., 2001, A Closer look at Extreme Modeling, www.drdobbs.com/xm/184414729. Acc. 15April '16.

Aramo-Immonen, H., 2009. *Project Management Ontology – The Organizational Learning Perspective*. (PhD Thesis). Tampere Univ. of Technology, Finland.

Barros, A., 2016. *Agile Management System for User Requirements Mappable into Software Models*. Project Report, I. Politécnico de Viana do Castelo, Portugal.

Beck, K., 2004. Extreme Programming Explained: Embrace Change. Addison-Wesley, 2nd ed., Nov. 26.

Curran, K., Flanagan, L., Callan, M., 2004. PMXML: An XML Vocabulary Intended for the Exchange of Task Planning and Tracking Information. *Information Technology Journal; 3 (2)*: pp. 192-195.

Cruz, A.M.R., 2015. Use Case and User Interface Patterns for Data Oriented Applications. In: Hammoudi, Pires, Filipe, Neves (Eds.), MODELSWARD'14, Revised Selected Papers, CCIS, vol. 506, p. 117-133, Springer.

DIN, 2009. DIN 69901-4 Project management - Project management systems - Part 4: Data, data model, Deutsches Institut für Normung. http://www.din.de.

Moody, D. L., 2009. The "Physics" of Notations: Towards a Scientific Basis for Constructing Visual Notations in Software Engineering. In IEEE TSE, vol. 35, no. 5.

OMG, 2015. OMG Unified Modeling Language (OMG UML). Version 2.5, March 2015.

PMI, 2013. A Guide to the Project Management Body of Knowledge (PMBOK, 5th ed.), P.M.I., PA, USA.

PMI, 2015. Lexicon of Project Management Terms– Version 3.0, Project Management Institute, PA, USA.

Sheeba, T., Krishnan, R., Bernard, M., 2012. An Ontology in Project Management Knowledge Domain. *Int'l Journal of Computer Applications*, vol 56–no 5, Oct.

Strembeck, M., Zdun, U., 2009. An approach for the systematic development of domain-specific languages. In Software-Practice and Experience, vol. 39, pp. 1253-1292. John Wiley & Sons, Ltd.

Ribeiro, G.G.G., 2015. *Software para Gestão de Projetos de Software*. (MSc Thesis, in portuguese), Instituto Politécnico de Viana do Castelo, Portugal.

Uschold, M., King, M., 1995. Towards a Methodology for Building Ontologies. *Workshop on Basic Ontological Issues in Knowledge Sharing*.

Palmer, S.R., Felsing, J.M., 2002. A Practical Guide to Feature-Driven Development. NJ, Prentice-Hall.

De Luca, J., n.d.. Feature Driven Development Overview, www.nebulon.com/articles/fdd. Accessed 15Apr 2016.

Schwaber, K., Sutherland, J., 2016. The Scrum Guide. July 2016. Available at http://www.scrumguides.org/.

Voelter, M., 2013. DSL Engineering: Designing, Implementing and Using Domain-Specific Languages. Available at http://dslbook.org/.