International Journal of Software Engineering and Knowledge Engineering
© World Scientific Publishing Company

DEFINING SOFTWARE PROCESS MODEL CONSTRAINTS WITH RULES USING OWL AND SWRL (EXTENDED VERSION)[*]

Daniel Rodríguez et al[†]

Department of Computer Science, University of Alcala, Ctra. Barcelona km. 33.6,
28871 Alcala de Henares, Madrid, Spain[‡]
daniel.rodriguezg@uah.es[§]
http://www.cc.uah.es/drg

Abstract. The Software & Systems Process Engineering meta-model (SPEM) allows the modelling of software processes using OMG (Object Management Group) standards such as the MOF (Meta-Object Facility) and UML (Unified Modelling Language) making possible to represent software processes using tools compliant with UML. Process definition encompasses both the static and dynamic structure of roles, tasks and work products together with imposed constraints on those elements. However, the latter requires support for constraint enforcement that is not always directly available in SPEM. Such constraint-checking behaviour could be used to detect possible mismatches between process definitions and the actual processes being carried out in the course of a project. This paper approaches the modelling of such constraints using the SWRL (Semantic Web Rule Language), which is a W3C recommendation. To do so, we need first represent generic processes modelled with SPEM using an underlying ontology based on the OWL (Ontology Web Language) representation together with data derived from actual projects.

Keywords: SPEM; Ontologies; OWL; rules; SRWL.

1. Introduction

Process modelling in general concerns the representation of resources, artifacts and dynamic behaviour of activities. As highlighted by Curtis [11] process modelling supports the following objectives: (i) facilitating human understanding and communication; (ii) supporting process improvement; (iii) supporting process management; (iv) providing automated guidance in performing processes, and (v) providing automated execution support. Process modelling is of paramount importance to improve the quality of organisation's processes, and in turn, the quality of products they generate. There are several well established standards used to describe process in descriptive view such as the ISO 12207 [19] as well as improvement frameworks such as the CMMI (Capability Maturity Model Integration) [10] or ISO/IEC 15504 standard [20] (also known as SPICE –Software Process Improvement and Capability dEtermination–).

[*] For the title, try not to use more than 3 lines. Typeset the title in 10 pt Times Roman, uppercase and boldface.
[†] Typeset names in 8 pt Times Roman, uppercase. Use the footnote to indicate the present or permanent address of the author.
[‡] State completely without abbreviations, the affiliation and mailing address, including country. Typeset in 8 pt Times Italic.
[§] Typeset author e-mail address in single line.

More recently, the OMG (Object Management Group)[**] has developed a meta-model to represent software processes, called SPEM (Software & Systems Process Engineering Meta-model) [24]. SPEM, currently in version 2, allow us to formalise all the relevant aspects (roles, products, deliverables, guides, life-cycle, phases, milestones, etc.) of generic software processes. SPEM is supported by different modelling tools such as the Eclipse Process Framework (EPF) Composer [††] aiming at better management and monitoring of projects. Although SPEM is increasingly gaining popularity as it is based on the same standards than UML, it is not the only possibility to represent process. For example, Grüninge and Menzel [15] describe the Process Specification Language (PSL) designed to exchange process information (scheduling, process modelling, process planning, production planning, simulation, project management, work flow, and business-process reengineering) among systems.

In another direction, ontologies [14][33] are explicit representations of domain concepts and their relationships. More formally, an ontology defines the vocabulary of a problem domain and a set of constraints on how terms can be combined to model the domain. Common uses of ontologies include communication between people and organizations and interoperability between systems, i.e., translation of modelling methods, paradigms, languages and software tools. Desirable qualities provided by ontologies include reusability thanks to formal representations, search-ability providing meta-data to information, and reliability performing consistency checking. In software engineering, ontologies can be used by applications require a higher level of formality of definition. For example, cataloguing resources or mapping of vocabularies from different information sources requiring precise definitions, or at least significant characterizations that help in deciding which terms to use in practical situations. Ontologies allow us to add semantics to data so that different software components can share information in a homogeneous way. For example, Sicilia *et al* [29] review of use of ontologies in the engineering domain and how upper ontologies can be of assistance. Furthermore, logic can be used in conjunction with such formal representations for reasoning about the information and facts represented as ontologies.

In this paper, we show how processes modelled using the SPEM framework can be translated into ontologies. Such representation together with actual data from current projects (also translated into ontologies) can provide reasoning capabilities for consistency checking, model validation, project and resource analysis, business rule analysis, etc.

The rest of this paper is structured as follows. Section 2 covers the background. Section 3 summarizes the processes of creating ontologies from SPEM, followed by how

[**] `http://www.omg.org/`
[††] `http://www.eclipse.org/epf`

constrains can be modelled and executed in Section 4. Finally, Section 5 concludes the paper and outlines future work.

2. Background

2.1. *Software Processes and SPEM*

As defined by the SWEBOK [18] a "*software process is a set of activities, methods, practices, and transformations which people use to develop and maintain software and the associated products*". Within the Software Engineering discipline, the definition, implementation, and improvement of processes is becoming increasingly important in what is called Software Process Engineering (SPE) and a large number of standards related to process modelling, assessment and improvement of process have been proposed, for example:

- Process Standards MIL-STD498, RUP (Rational Unified Process), Open UP, XP (Extreme Programming), ISO 12207, etc.
- Quality Standards: ISO 9000, SCAMPI (Standard CMMI Appraisal Method for Process Improvement), etc.
- Capability standards CMMI (capability maturity model integration), ISO/IEC 15504 (SPICE - Software Process Improvement and Capability Determination), etc.
- Guidelines such as PSP (Personal Software Process), TSP (Team Software Process), Six Sigma, etc.

Although many of the software process standards can provide some computer support such as on-line documentation or templates to ease the bureaucratic burden, they are mainly based on paper manuals using natural language presenting several difficulties, e.g., difficulty accessing to the information, many different versions of the same documents, lack of tailored processes to specific environments or projects, etc. When dealing with the actual management of software process, several software systems were proposed as automated prescriptive models prior to SPEM, such as EPOS (Process Centred Software Engineering Environment) [23], Marvel [5], SPADE (Software Process Analysis, Design, and Enactment) [4] which is based on Marvel, etc. However, a major drawback of these systems is the lack of standard representations and formats.

The SPEM specification is the first step towards formalising the engineering of processes. In the same way as we can model software systems using the UML (Unified Modelling Language), it is now possible to define processes formally using SPEM which is in turn based on other OMG standards including UML and MOF (Meta Object Facility). The MOF specification defines a modelling architecture based on four levels as shown in Figure 1. A *n-1* level instantiates the elements from the level *n* (excluding the upper –M3– level). In this way, a process is built on top of more generic concepts until suitable for specific environments.

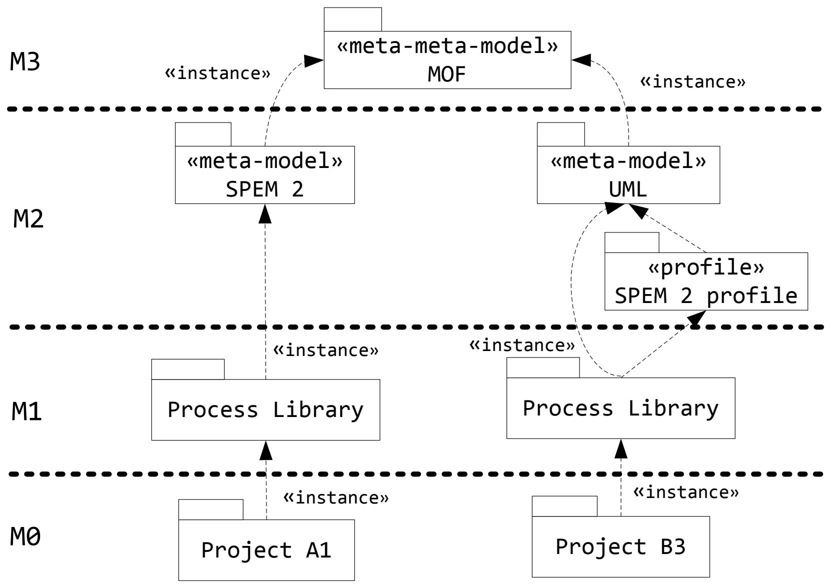


Figure 1. Meta-models hierarchy

In addition to a better management and improvement of processes, SPEM objectives include the improvement of human comprehension of the processes, facilitate process tailoring and reuse as well as the automation of software process execution. SPEM is open specification with all the necessary concepts to design, model, publish and tailor software engineering processes in order to (i) create a repository of reusable content; (ii) support the management and development of software processes; (iii) establishing a process framework within an organisation (e.g., CMM level 3 needs defined as well as tailoring mechanisms) and (iv) generation of templates of actual projects. It is worth noting that SPEM is mainly designed for software processes and not as a general process modelling. Other efforts exist in such direction such as the BPDM (Business Process Definition Metamodel)[‡‡] and BPMN (Business Process Modelling Notation)[§§] which are also maintained by the OMG.

When using the SPEM standard, processes can be defined using two approaches (i) as a UML profile and (ii) as a meta-model. A UML profile defines a series of stereotypes (mainly graphical icons) to represent software engineering concepts without adding constrains, i.e., it does not include any semantics (for example, there is no way to add that the relationship between a task and a role must exist one or more performers). Therefore, when used as a profile, it is mainly a diagrammatical tool using UML artifacts extended with visual icons to represent software process concepts. It has the advantage of allowing

‡‡ `http://www.omg.org/spec/BPDM/1.0/`
§§ `http://www.bpmn.org/`

us to represent a process visually using any UML tool. As a meta-model, SPEM processes can include the semantics of the MOF meta-model and it is possible to automate the translation process between different representations (being MOF the core of the Model Driven Architecture (MDA)[***].

The SPEM specification defines two types of concepts: (i) the *Method Content* with basic elements such as *Role*, *Task* and *WorkProduct*: and (ii) *Process* as a combination of previously defined content elements as a dynamic structure. Such separation promotes the reusability of processes and its adaptation to different software life-cycles thanks to two types of extensibility and variability mechanisms called *method plug-in* and *process plug-in* capable of adding or adapting. The concepts are organised in the following meta-model packages:

- *Core*: It contains common classes and abstractions used to build upon.
- *Process structure*: It represents the static concepts of processes with nesting activities and predecessor and successor dependencies.
- *Process behaviour*: It extends the *Process Structure* package with behavioural models such as activity diagrams for process behaviour or work products with state machines to represent its lifecycle.
- *Managed Content*: This package introduces concepts for managing textual description (natural language) and documentation capabilities for processes.
- *Method Content*: It provides the concepts for defining lifecycle and process independent reusable method content elements that provide a base of documented knowledge of software development methods, techniques, and best practices.
- *Process With-Methods*: It defines new and redefines existing structures for integrating Process Structure concepts with instances of Method Content concepts (Tasks, Roles, and Work Products) into the context of a lifecycle model comprising, for example, phases and milestones.
- *Method Plug-in*: It allows us to introduce the concept of variability in processes, in what is called method configuration, where the user can add or remove elements without modifying the original model.

[***] `http://www.omg.org/mda/`

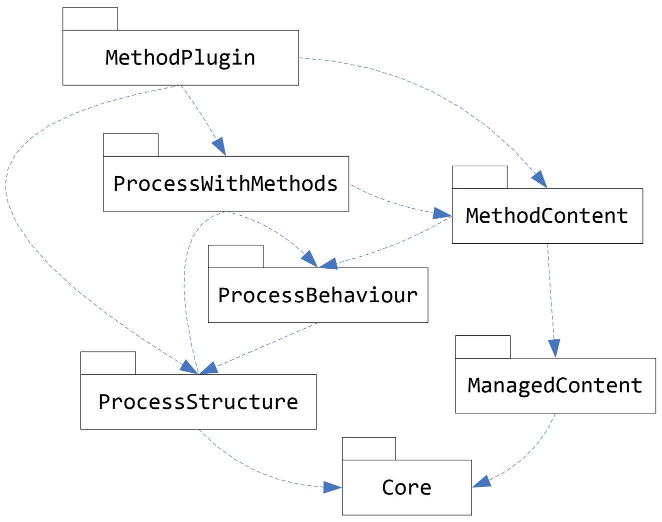


Figure 2 SPEM 2 Meta-packages

There is no need to use all the packages defined by SPEM 2. For example, some organisations could just use the *Core*, *Managed Structure* and *Method Content* as a way of organising a documentation repository for the processes.

A repository (or *Method Library*) is composed of one or more *Plug-ins* and *Method Configurations*. Plug-ins are in turn divided into two components: (i) *Method Content* and (ii) *Processes* modelling static and dynamic concepts respectively. Also, as there is no need to use all process documentation at one given instance in time, a tailored subset can be defined with *Method Configurations*. For example, different views can be shown to different roles within an organization (e.g., developers only need the information related to programming). Figure 3 shows the organization of a process using SPEM –left hand side– and how a tool such the EPF Composer mirrors such structure –right hand side–).

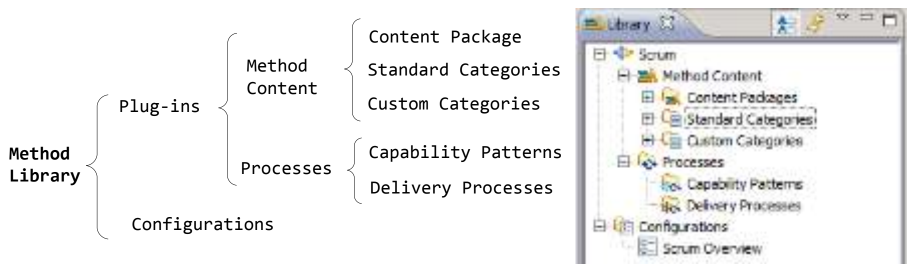


Figure 3 SPEM 2 Organization

Once the organisation is described, we next need to define the main elements of the packages without being exhaustive as many concepts are not visible when modelling process. The basic elements of *Method Content* include:

- *Tasks* are atomic units of work composed of a series of *Steps*.
- *Roles* are defined as set of abilities, competencies and responsibilities related to an individual or group of individuals.
- *Work products* are *artifacts*, *deliverables* or *outcomes*.
- *Guidance* elements provide additional information related to other elements. There is large number of defined guidance elements such as *reusable asset*, *term definitions*, *guidelines*, *whitepapers* and *examples*.
- *Categories* are in turn classified as *Standard Category* and *Custom Category* used to organise and create hierarchies of elements. The *Standard Category* is composed of five predefined categories: *Role set* to group roles (e.g., analysts could group requirements analysts and design analysts); *discipline* to categorize tasks; *Domain* to create hierarchies of work products; *Tool* to categorise tool guides; and *Work Product Kind* to allow us to include a work product under different classes.
- *Associations* between *content elements* such as *Task – Steps* as an ordered list to perform a task, *Task – Roles* which is composed of *primary performer* and *additional performers*, *Task – Work Products* composed of *mandatory inputs*, *optional inputs* and outputs, etc.

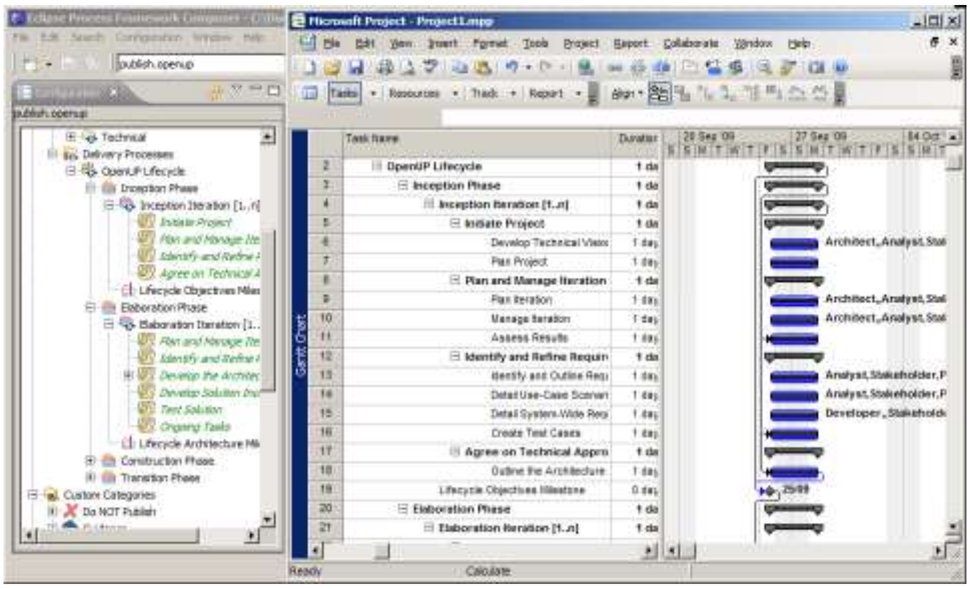
Content Elements (*Tasks*, *Roles* and *Work Products*) that are instantiated in a particular process end with the suffix *"Use"*. For example, we have *Task Use*, *Work Product Use*, and *Role Use* representing actual instances of the definition of a an activity, actual artefact and actual roles in a process respectively. Note that it refers to the generic term in a process definition but it does not correspond to any concrete project (e.g., the requirements document in Scrum is referred generically as the backlog).

On the other hand, we have *Processes* in which the *Method Content* described are combined to define activities and processes. Basic elements include:

- *Work Definition* is an abstract concept that generalises all types of *work definitions*.
- *Breakdown Element* is an abstract generalisation for all other types of process elements, mainly *Process Parameters*, *Process Performers*, *Work Breakdown Elements*, and *Work Sequence* connecting two *Work Breakdown* elements (predecessor and successor). *Work Breakdown Elements,* which are the main type of *Breakdown element*, are composed of *Activities* and *Milestones. Activities* are preformed by *Process Performer*s and can have input and output parameters (*Process Parameters*). Finally, SPEM defines three types of *Activities*: (i) *Phase* a significant non repeatable time span of a project typically ending typically in a milestone; (ii) *Iteration,* a repeatable set of nested tasks; and (iii) *Milestone*.

Based on process patterns, SPEM provides two classes for adapting and dynamically ensemble processes: (i) *Capability Pattern*, a generic and reusable software piece that can be reused across several processes; and (ii) *Delivery Process* which describes a complete and integrated approach for performing a specific project type, i.e., it covers a complete project lifecycle to be used as a reference for executing projects following the same process such us XP, RUP or Scrum.

Currently, there are several tools such as the Eclipse Process Framework (EPF) Composer capable of editing processes using SPEM. The EPF Composer uses XMI (XML Metadata Interchange)[†††], another OMG standard for manipulating, storing and interchanging information between tools and it enables to export a process template to project management tools (e.g., Microsoft Project). It can also generate the process documentation in HTML format to be accessed through the Web. Figure 4 shows how a process modelled using the EPF composer can be exported to MS Project. It can be noted that it is necessary to include further information such as the actual duration of tasks.


```
publish.openup
  Technical
  Delivery Processes
    OpenUP Lifecycle
      Inception Phase
        Inception Iteration [1..n]
          Initiate Project
          Plan and Manage Ite
          Identify and Refine 
          Agree on Technical A
        Lifecycle Objectives Mile
      Elaboration Phase
        Elaboration Iteration [1..
          Plan and Manage Ite
          Identify and Refine 
          Develop the Architec
          Develop Solution Inc
          Test Solution
          Ongoing Tasks
        Lifecycle Architecture Mi
      Construction Phase
      Transition Phase
  Custom Categories
    Do NOT Publish
```

Microsoft Project - Project1.mpp

	Task Name	Duration	
2	OpenUP Lifecycle	1 da	
3	Inception Phase	1 da	
4	Inception Iteration [1..n]	1 da	
5	Initiate Project	1 da	
6	Develop Technical Visio	1 day	Architect,,Analyst,Stal
7	Plan Project	1 day	
8	Plan and Manage Iteration	1 da	
9	Plan Iteration	1 day	Architect,,Analyst,Stal
10	Manage Iteration	1 day	Architect,,Analyst,Stal
11	Assess Results	1 day	
12	Identify and Refine Requir	1 da	
13	Identify and Outline Requ	1 day	Analyst,Stakeholder,P
14	Detail Use-Case Scenari	1 day	Analyst,Stakeholder,P
15	Detail System-Wide Requ	1 day	Developer,,Stakeholde
16	Create Test Cases	1 day	
17	Agree on Technical Appro	1 da	
18	Outline the Architecture	1 day	
19	Lifecycle Objectives Milestone	0 day	25/09
20	Elaboration Phase	1 da	
21	Elaboration Iteration [1..n]	1 da	

File Edit Search Configuration Window Help
File Edit View Insert Format Tools Project Report Collaborate Window Help
Tasks Resources Track Report Align
Gantt Chart
20 Sep 09
27 Sep 09
04 Oct
Ready
Calculate

Figure 4 A project in the EPF Composer (left) exported to MS Project (right)

2.2. *Ontologies and reasoning*

Ontologies, as the shared representation of domain concepts and their relationships can be represented in different formalisms for quite a long time. Since the inception of the Semantic Web, in which ontologies are the principal recourse to integrate and deal with online information, a new set of standards has been proposed. The Ontology Web Language (OWL) [31] is one of such standards that belongs to a family of knowledge representation languages prepared for the Semantic Web that has reached status of W3C

[†††] `http://www.omg.org/technology/documents/formal/xmi.htm`

(World Wide Web Consortium) recommendation. From technical point of view OWL extends the RDF (Resource Description Framework) and RDF-S (RDF Schema) allowing us to integrate a variety of applications using XML as interchange syntax. There are three OWL flavours, OWL Lite, OWL DL (Description Logics) and OWL Full, being the OWL depending on the expressiveness and reasoning capabilities provided. In short, OWL ontologies are composed of (i) *classes* as sets of individuals, (ii) *individuals* as instances of classes, i.e., objects of the domain and (iii) *properties* as binary relations between individuals. It is possible to specify property domains, cardinality ranges and reasoning on ontologies. Reasoning in OWL can be performed at a class, property or instance level and reasoning examples include class membership, equivalence of classes, consistency, classification of the information, obtaining additional properties using transitiveness or equivalent, etc.

Another W3C standard, the Semantic Web Rule Language (SWRL) [17], based on RuleML[‡‡‡], extends the OWL providing logic based rules, and in consequence, more expressiveness. Rules have the form of antecedent implies a consequent. Figure 5 shows the use of SWRL rules with OWL ontologies. Rules together with stored facts (knowledge base) are executed as inputs to by the rule engine inferring new facts as an output. Also, if the inference engine infers knowledge using forward chaining, the new knowledge can be used for further inference (in contrast with backward chaining where the search for knowledge starts from the consequent to the antecedent).

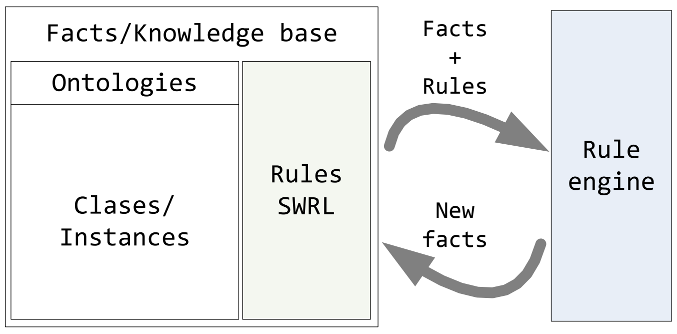


Figure 5. Execution of rules adding new knowledge/constrains from rules

The open source Protégé tool[§§§] is one of the possible tools that can be used for creating ontologies. It includes the SWRLTab which is an extension for editing and executing of SWRL in conjunction with JESS[****], a rule engine.

[‡‡‡] http://ruleml.org/
[§§§] http://protege.stanford.edu/
[****] http://www.jessrules.com/

3. Representing processes with OWL Ontologies

As stated previously, SPEM is generally used to design generic software processes such as the Open Unified Process, XP or Scrum. In this section, we discuss a first approach to create an ontology from the Scrum process [26] defined using SPEM. Scrum is a relatively simple process with a reasonable number of classes and properties.

Although SPEM models can be translated to a representation in OWL while retaining the modelling semantics specified in SPEM, the creation of ontologies is not straight forward. There are no standard modelling methodologies but a mix of guidelines that are combined with techniques from the database modelling and object oriented modelling to iteratively achieve the desired representation [12]. In any case, the translation should not aim at substituting the original model, but to serve as a complement for adding reasoning and inference support to SPEM based models.

Typically, a SPEM model would be translated into one or several OWL modules, in addition to other OWL modules with basic mappings that would be imported by these. It should be noted that there is no need to translate each SPEM package into an OWL module as simplicity has been preferred in contrast to mirroring every SPEM element. Many terms in the standard are linked to other through inheritance to provide the necessary semantic meaning which can be directly defined when creating the ontology. For example, *ExtensibleElement* is an abstract class for all the SPEM elements and the ontology can be focused on more visible elements such as *task* or *role.* The packages, however, can help to organise the different concepts in different ontologies and files). For example, from the *Core* package, we just selected the *ParameterDirectionKind* to create and enumeration of *input*, *output* or *inputOutput* linked through a property (*hasParameterdirection*). Furthermore, the *Task*, *Role* and *WorkProduct* elements that appear in the *Method Content* package are defined in the method-content ontology. In the same way the *Activity*, *Milestone*, etc. classes from the *Process* package in another *process* ontology.

Roles in scrum are divided into: (i) *Chicken* roles which are not part of actual scrum process but need to taken into such as account stakeholders and customers; and (ii) *Pig* roles which are committed to the project. The *Pig* roles in Scrum are the *scrum master*, i.e., the project manager, the *product owners* that represent the stakeholders; and the *team*, which carries out the actual project usually in relatively small teams of around 7 people capable of self-organized. In the ontology, the *Role* class from the method-content ontology can be extended with in the Scrum ontology, i.e., *method-Content*:*Role* can have *scrum:Pig* and *scrum:Chicken* as subclasses. The *productOwner*, *scrumMaster* and *team* will be instances of the Scrum *Chicken* role.

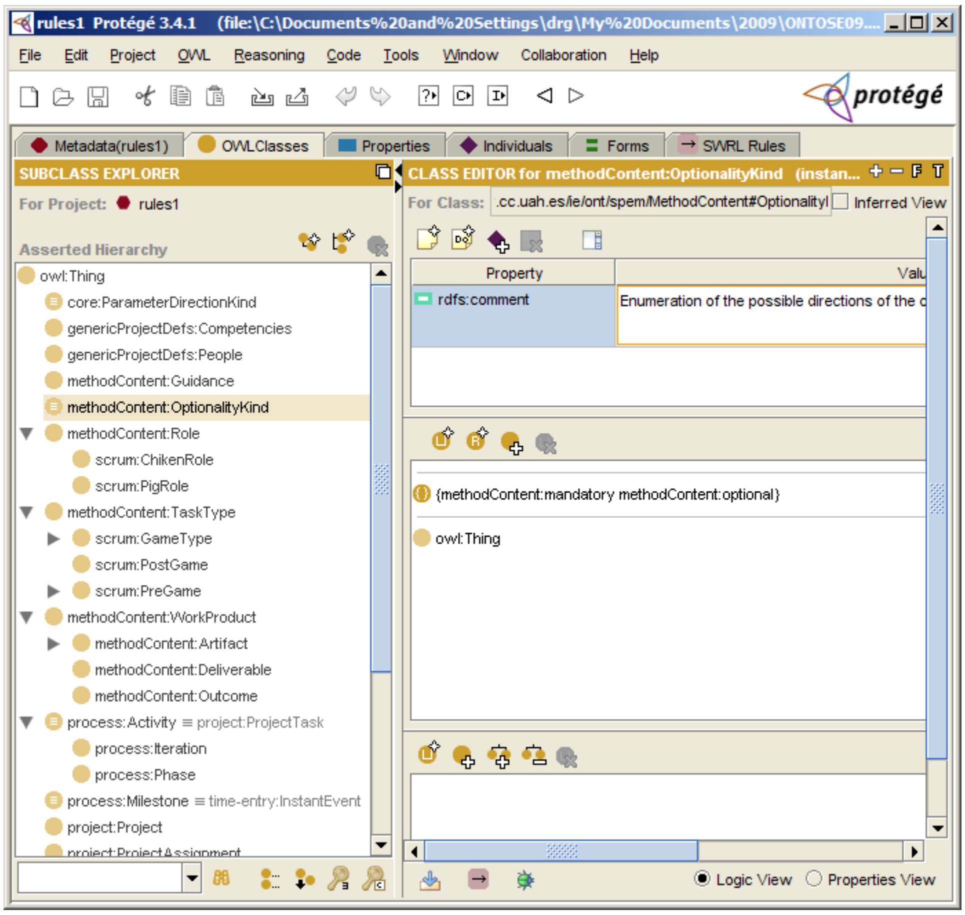
The development of a project using Scrum is performed iterative and incrementally in cycles called *sprints*. Each sprint is supposed to end up with a working system that could be potentially delivered to the client. Requirements are prioritised in what is called the *Product Backlog* which is regularly updated and new items, detailing items, estimates and so on. Before starting each sprint, the functionality from the product backlog to be included in the next sprint (*sprint backlog*) is decided during the *sprint planning meeting*. All these terms can be represented in OWL a as a generic process extending the basic terms. In the ontology, these terms are defined as *WorkProduct* in the *method content* ontology with *Artifact*, *Deliverable* and *Outcome* as subclasses. Then, an instance or individual of *Artifact* will be the *sprintBacklog* as part of the *scrum* ontology.

The execution of a project using the Scrum process is composed of the following three phases: (i) *Pre-game* composed of two phases *Planning* which defines the system to be built (*Product Backlog*, estimates, etc.) and *High-level Design* of the system based on the *Product Backlog* and the *Design Review Meeting*, a preliminary planning for the releases is outlined; (ii) *Game* or *Development Phase*. The development is performed in sprints. There are several predefined meetings, the *sprint planning meeting* at the beginning of each sprint, a *daily scrum* stand-up meeting and a *sprint review* meeting; and the (iii) *Post-game phase* as the closure of the project. The *process* ontology has the *Activity* class with *Iteration* and *Phase* classes defined as subclasses. The *PreGame*, *Game* and *PostGame* are defined in the *scrum* ontology.

Scrum already defines a set of rules that must be followed and many of them are related to timing constrains. For example, once a sprint has started, the items from the sprint backlog cannot be modified, another one is that at the beginning of each day, a *stand up meeting*, called the *daily scrum* must take place and only people committed to the project can talk (not those only involved). There are also other rules related to time; for example, duration of each sprint can vary between 15 days to a month but no more and there is a sprint planning meeting at the beginning that should not last more than eight hours. There is also a sprint review meeting with a time limit of four hours. There is also another meeting defined by Scrum, the *sprint retrospective*, in which all team members analyse what went well and what can be improved in the next sprint with a time limit of three hours. In the ontology, we can deal with time and time constrains for such classes either using the built-in types or merge developed ontologies such as the time ontology[††††] developed by Hobbs and Pan [16]. The same applies to generic terms in the SPEM standards such as *Metric*. The *Metrics* class is the only concept defined in SPEM to contain measurements such as effort estimations of activities, or maximum duration of Scrum meetings and further refined ontologies for metrics have defined by [13] among others. Another example is the matching between roles, people and competences that can be exported from other ontologies [28].

[††††] `http:// www.w3.org/TR/owl-time/`

When an actual project is represented in the ontology, we need to include concrete people, task and time information possibly from project management tools and new classes in the ontology are necessary to provide the links. For example, the actual personnel from an organization could be stored in an ontology which we called *genericProjectDefs* which can have the *People* class and all the organisation personnel as instances. As stated previously, all properties (links between classes) are binary, therefore, in order to link personnel, roles and tasks we need to define an *n*-ary relationship. To do so, we need define a new class in which instances and other such tools can be linked through properties (called reified relations).


rules1 Protégé 3.4.1 (file:\C:\Documents%20and%20Settings\drg\My%20Documents\2009\ONTOSE09....
File Edit Project OWL Reasoning Code Tools Window Collaboration Help
protégé
Metadata(rules1) OWLClasses Properties Individuals Forms SWRL Rules
SUBCLASS EXPLORER
CLASS EDITOR for methodContent:OptionalityKind (instan...
Asserted Hierarchy
Property
Valu
Inferred View
Logic View
Properties View

For Project: rules1

owl:Thing
core:ParameterDirectionKind
genericProjectDefs:Competencies
genericProjectDefs:People
methodContent:Guidance
methodContent:OptionalityKind
methodContent:Role
scrum:ChikenRole
scrum:PigRole
methodContent:TaskType
scrum:GameType
scrum:PostGame
scrum:PreGame
methodContent:WorkProduct
methodContent:Artifact
methodContent:Deliverable
methodContent:Outcome
process:Activity ≡ project:ProjectTask
process:Iteration
process:Phase
process:Milestone ≡ time-entry:InstantEvent
project:Project
project:ProjectAssignment

For Class: .cc.uah.es/ie/ont/spem/MethodContent#OptionalityI

Property	Valu
rdfs:comment	Enumeration of the possible directions of the c

{methodContent:mandatory methodContent:optional}

owl:Thing

Figure 6 Process Ontology modeled in Protégé

For the translation, the OWL ontologies were elaborated using the Protégé tool as shown in **Error! Reference source not found.**. Although the nomenclature used in the

SPEM standard or Scrum was followed whenever was possible, it should be noted that other translation approaches could be devised in the future with a more comprehensive alignment to the OMG meta-modelling specifications. This includes the use of the recently proposed ODM (Object Definition Metamodel)[‡‡‡‡] specification which can be used for translations between metamodels. A large number of terms are generic to software engineering processes and methodologies and defined in numerous standards, guides and other ontologies. Such works can be used in conjunction with SPEM as starting point of the ontological process and merging of ontologies. For example, the *Metric* concept in SPEM 2 is could be further expanded with much richer descriptions from other ontologies such as the SMO (Software Measurement Ontologies) [13]. However, merging ontologies and terminology from the software engineering standards is not a trivial task. For example, *Activity* and *Task* definitions in SPEM do not exactly comply with the ISO 12207 standard as it defines activity as life cycle phase and a task as something performed within an activity.

4. Modelling process constraints with SWRL

As stated previously, the main motivation of this work is to actually check and verify constrains that can be defined as part of a SPEM process models and other information that can be obtained from project management tools. It can be devised, for example, that other information gathered from software repositories (e.g., metrics) could be included in as OWL ontologies and constrains could be verified using rules.

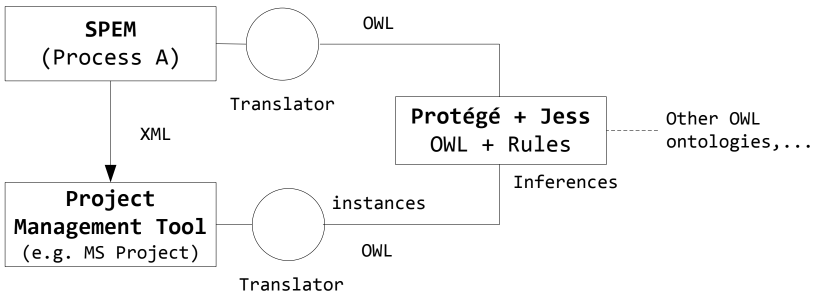


Figure 7 Extension of SPEM with Semantic Knowledge and Rules

Figure 7 shows this approach, where ontologies represented using OWL and rules with SWRL are combined in order to better manage the project. On the one hand, we have generic process information from SPEM models using tools such as the Eclipse Process Framework. Also, configurations of a concrete project can be exported to project management tool (e.g., MS Project) in which the concrete process specification can be populated with information about personnel, information about start and end dates of

[‡‡‡‡] http://www.omg.org/docs/formal/09-05-01.pdf

activities, their duration, etc. Information from both sources can populate instances in ontologies that can be enriched with constrains in the form of rules using tools such as Protégé in conjunction with rules engines such as JESS. In this environment can execute such rules to verify constrains and inconsistencies in a project as well as possible incorporate new knowledge into the project management tools to better monitor the project. It is worth noting that although many of the constrains in UML can be defined using the Object Constrain Language (OCL), however, it currently lacks the maturity and tool support provided by the semantic Web. Following the example described in the SPEM specification [24] as a precondition: "*Input Document X has been reviewed and signed by customer AND the work defined by Work Definition 'Management Review' is complete*". Such precondition is expressed in natural language and associated to the `WorkDefinition` class compositional association (Figure 8 shows the UML class diagram for the WorkDefinition class). Even if expressed in OCL, we are not aware of any environment that allows their execution.

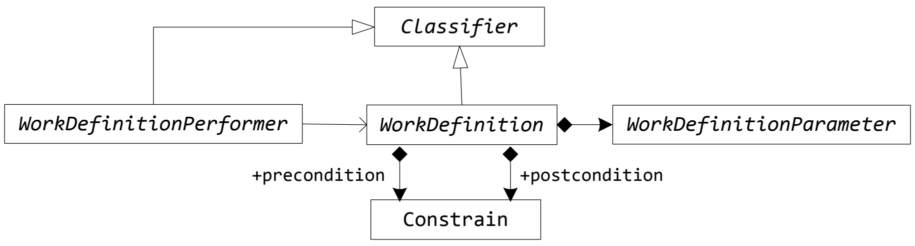


Figure 8 Constrains as part of the *WorkDefinition* abstract class

After defining the SPEM ontology, we can now provide an overview of how SPEM and project constrains can be expressed using the SWRL as a rule language capable of checking and verifying constrains. It is possible to run rules at the same level or between different levels in the ontological hierarchy shown in Figure 9.

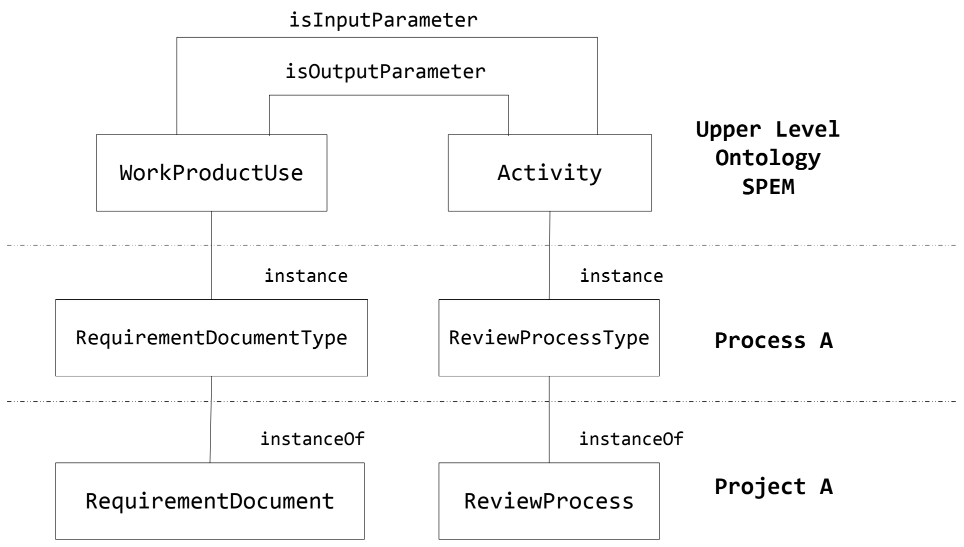


Figure 9 Different Semantic Levels when Creating SPEM Ontologies

An example of executing rules at the same level could be as follows. When an activity has a work product as input (`isInputParameter`) and output (`isOutputParameter`), a rule could automatically include another property that such work product is both input and output parameter. In the UML SPEM profile this is defined as an enumeration (`ParamterDirectionKind`).

```
WorkProduct(?a)  ∧
ProjectTask(?t)  ∧
isInputParameter(?a, ?t)  ∧
isModifiedBy(?a, ?t)
  → hasParameterDirection(?a, core:inOut)
```

Another example of rule could be when a work product goes through the process of review; in such a case, there could exist a property (`isReviewedBacklog`) which automatically is updated to true.

```
   methodContent:WorkProduct(sprintBacklog1)  ∧
   project:isInputParameter(sprintBacklog1,
sprintPlanningMeeting1)
     → isReviewedBacklog(sprintBacklog1, true)
```

We could specify concrete examples using SCRUM. For example, when running a project if the backlog for a sprint has been agreed and the sprint started, then we could assign to the Boolean property `modifiable` the value false. Note that such a rule could be part of some guideline when specifying the process.

```
WorkProduct (sprintBacklog1) ∧
isAgreed(sprintBacklog1) ∧
sprintStarted(sprintBacklog1, true)
⇒ modifiable(sprintBacklog1, false)
```

As stated, many of the restrictions apply to time. One possible solution could be to create all temporal needed properties using the simple built-in datatypes from XML Schema. However, a more powerful approach and possibly more elegant is to include the data entry ontology [16].

```
ScrumPlanningMeeting(?x)
startDate (?tartMtingDt1, ?x) &
temporal:add(?endMtingDt1, ?startMtingDt1, 8, “Hours”)
temporal: durationGreaterThan(
⇒ nonConformingMeeting(?x)
```

Other possible example could be more generic to all types of projects. For example, assuming that a person is always assigned full time to an activity, we could verify that a person can not be assigned to two overlapping activities:

```
People (?p) ∧ WorkProduct(?plan)
Activity(?act1) ∧ Activity(?act2) ∧
assignedRelation(?p, ?act1) ∧
assignedRelation (?p, ?act2) ∧
activityOverlaps(act1, act2)
⇒ conformingPlan (?plan, false)
```

Similar restrictions can be applied to the duration of each of the iteration in a Scrum project as the can not be longer than 30 days:

```
activitySprint(?sp1) ∧ startDate(dt1?, ?spr1) ∧
temporal:durationGreaterthan (30, ?dt1, “Days”)
⇒ conformingPlan (?sp1, false)
```

Other examples could be generic about personnel, competencies of the personnel and types of jobs that could be assigned and even inferred knowledge could be used to perform further inference finding suitable combinations of personnel, schedule and tasks.

```
inTeam(?x) ^
role(DBDesigner, ?x) ^
experienceCompetencyLevel (?x, DataBases, "High")
⇒ suitableTeamMember(?x, true)
```

5. Related Work

One of the initial ontologies in software engineering is REFSENO (Representation Formalisms for Software Engineering Ontologies) developed by Tautz and von Wangenheim [32]. This ontology has been applied for modelling experience factories [6] using the Goal-Question-Metric paradigm [4] and to software maintenance process by Vizcaíno *et al* [34]. Kitchenham *et al* [21] also defined an ontology for software maintenance using UML as a formalism for identifying and defining several domain factors (e.g., product, process, people and organization) and attributes that influence the maintenance process. Based on Kitchenham *et al* work, Ruiz *et al* [27] defined another ontology for the management of software maintenance. Althoff *et al* [2] describe an architecture oriented to reuse the experience in software engineering that use ontologies as the underlying formalism. In relation to process ontologies, Ceravolo *et al.* [9] describe the Extreme Programming Ontology (XPO) specifying the main concepts of the XP methodology. Authors aim is to analyse agile processes, mining process data about developer's activity and repositories content in order to extract new concepts potentially identifying critical factors in agile software development. As SPEM is increases its popularity as a way of representing models more formally, several researchers are using SPEM as a foundation for defining ontologies. For example, García *et al* [13] developed an ontology to represent software engineering measurement concepts merging SPEM with other ontologies.

Researchers have also developed ontologies based on current standards such as the Guide to the Software Engineering Body of Knowledge (SWEBOK) [18], which is also an ISO standard (ISO/IEC TR 19759:2005). Standards provide an agreement on the content of what compose the software engineering discipline opening new possibilities to ontology engineering in the field of software engineering, since they represent a shared consensus on the contents of the discipline. For example, Abran *et al* [1] report on the developed of a software engineering ontology based on the SWEBOK and the process for its creation.

Although these works represent a very important and starting point to define terms and processes related to software engineering, most of cited ontologies mainly define concepts and their relationships without providing formalisms for reasoning. In most cases, ontologies consist of definitions defined using UML classes and attributes with textual descriptions of the definitions without reasoning capabilities. When referring to reasoning with models are mainly focused on the UML (in which SPEM can be based) and the OCL (Object Constrain Language) as a way of verifying models. For example, Cabot, Clarisó and Riera [8] describe how to transform UML class diagrams together with OCL specifications into Constrain Satisfaction Problems (CSP) in order to validate them. A similar approach is taken by Queralt and Teniente [25] to validate UML models and by Simmonds *et al* [30]. As a result of the REWERSE project, Milanović *et al* [22] have defined an approach for meta-model transformation between UML/OCL and OWL/SRWL, based on the R2ML (REWERSE Rule Markup Language) which is a MOF-defined pivotal language for the translation. The MOF is a meta-modeling language for specifying models, i.e., it allow us to specify models of modeling languages. As stated by Milanović *et al* , there are benefits of the bridging the gap between OMG models such as UML or SPEM and the semantic Web with OWL. On the one hand, OWL has become the de-facto standard for specifying ontologies and on the other hand, models which they define as set statements of can be verified using the reasoning technologies provided by the Semantic Web. From the same project, Aβmann, Zshaler and Wagner [3] also present a schema combining ontologies and metamodels for the MDE (Model Driven Engineering) approach as a complementary techniques.

6. Conclusions and Future Work

This paper describes how software process ontologies can be derived from the Software and Systems Process Engineering Meta-model (SPEM) models. SPEM standardises and formalises the way of representing software engineering processes in relation to both their static and dynamic concepts such as activities, roles, tasks and work products. Ontologies in turn can be extended with rules representing constrains over elements of a concrete software project and those rules can be executed to verify such constrains and discover possible problems during the execution of a project. We presented a basic approach as a proof of concept using the Ontology Web Language (OWL) combined with SWRL (Semantic Web Rule Language) rules to represent constraints as rules. This approach provides the benefit of representing certain information that can not be represented in SPEM alone and furthermore it can be automatically verified.

Future work includes the further development of the ontologies and rules for existing software processes that started here as a proof of concept. In this work, we created the ontology manually; it may however be possible to obtain a first version of the ontologies using model transformations. As both SPEM and the new ODM (Ontology Definition Metamodel) are defined using meta-models, the translation can be performed using M2M (Model-to-Model) using for example the ATL[§§§§] (Atlas Transformation Language) or the QVT[*****] (Query/View/Transformation) specifications. Also, there are several existing ontologies related to software engineering or other disciplines (e.g., measurement ontologies, competencies) that can be integrated and form an important activity in the development of new ontologies and tool support for processes and project

[§§§§] `http://www.eclipse.org/m2m/atl/`
[*****] `http://www.omg.org/spec/QVT/1.0/`

managers. The extension of process frameworks such as the Eclipse Process Framework to include rules as well as translators between project management and ontology based tools for the introduction of rules and their verification. Another open research issue is not only to verify the constrains of a project, but also how new inferred information could fed back to the project management tools in order to improve the control of a project.

References

[1] A. Abran, , J. Cuadrado, E. Garcia-Barriocanal, O. Mendes, S. Sanchez-Alonso, M.A. Sicilia, Engineering the ontology for the software engineering body of knowledge: issues and techniques. Ontologies for Software Engineering, Springer Verlag, New York, NY, (2006).

[2] K.D. Althoff, A. Birk, S.Hartkopf, W.Muller, M. Nick, D.Surmann, C. Tautz,: Systematic Population, Utilization, and Maintenance of a Repository for Comprehensive Reuse. In *Learning Software Organizations - Methodology and Applications*, Springer Verlag, LNCS 1756, pp. 25-50. (2000)

[3] U. Aßmann, S. Zschaler and G.Wagner, Ontologies, Meta-models, and the Model-Driven Paradigm. In Ontologies for Software Engineering and Software Technology, Calero, C, Ruiz F. and Piattini M., (Edts) Springer Berlin Heidelberg, 2006.

[4] S. C.Bandinelli, A. Fuggetta, and , C.Ghezzi 1993. Software Process Model Evolution in the SPADE Environment. *IEEE Trans. Softw. Eng.* 19(12) (Dec. 1993), pp. 1128-1144.

[5] N. BarghoutiS. 1992. Supporting cooperation in the Marvel process-centered SDE. *SIGSOFT Softw. Eng. Notes* 17, 5 (Nov. 1992), pp. 21-31.

[6] V. Basili, R. G. Caldiera, and H. D. Rombach, The experience factory, in *Encyclopedia of Software Engineering*, ed. J. J. Marciniak, (John Wiley & Sons, 1994), pp. 469–476.

[7] Berardi, D., Calvanese, D., De Giacomo, G., Reasoning on UML class diagrams, Artificial Intelligence, Volume 168, Issues 1-2, October 2005, Pages 70-118.

[8] J. Cabot, R. Clarisó, D. Riera, Verifying UML/OCL Operation Contracts, 7th International *Conference on integrated Formal Methods* (IFM 2009), Düsseldorf, Germany, February 16-19, 2009. Lecture Notes in Computer Science LNCS Volume 5423/2009, pp 40-55.

[9] P. Ceravolo, E. Damiani, M. Marchesi, S. Pinna, F. Zavatarelli, A ontology-based process modelling for XP, In *Software Engineering Conference, 2003. Tenth Asia-Pacific* , pp. 236-242, 2003

[10] CMMI, Capability Maturity Model Integration, Version 1.2. Web site: `http://www.sei.cmu.edu/cmmi/`

[11] B. Curtis, M. I. Kellner, and J. Over, 1992. Process modeling. *Communications of the. ACM* 35, 9 (Sep. 1992)

[12] V. Devedžic, Understanding Ontological Engineering, *Communications of the ACM*, **45**(4) (2002) 136-144.

[13] F. García, F. Ruiz, C. Calero, M.F. Bertoa, A. Vallecillo, B. Mora, M. Piattini: Effective use of ontologies in software measurement. *Knowledge Engineering Review* **24**(1) (2009) 23-40.

[14] T.R., Gruber, Toward Principles for the Design of Ontologies Used for Knowledge Sharing, *International Journal of Human-Computer Studies*, **43**(5-6) (1995) 907-928.

[15] M. Grüninger, and C. Menzel, The process specification language (PSL) theory and applications, *Artificial Intelligence Magazine*. **24**(3) (2003) 63-74.

[16] J.R. Hobbs, F. Pan, 2006, Time Ontology in OWL. W3C Working Draft 27 September 2006, `http://www.w3.org/TR/2006/WD-owl-time-20060927/`

[17] I. Horrocks, P.F. Patel-Schneider, H. Boley, S. Tabet, B. Grosof, M. Dean, SWRL: A Semantic Web Rule Language Combining OWL and RuleML, W3C (World Wide Web Consortium), 2004. Web site: `http://www.w3.org/Submission/SWRL/`

[18] IEEE, SWEBOK, Guide to the Software Engineering Body of Knowledge. 2004, Web site: `http://www2.computer.org/portal/web/swebok`

[19] ISO: ISO/IEC 12207:2008 Systems and software engineering -- Software life cycle processes (2008)

[20] ISO: ISO/IEC 15504 Information technology Process assessment. Parts 1 to 8. Web site: http://www.iso.org/ (2009)

[21] B.A. Kitchenham, G.H. Travassos, A.V. Mayrhauser, F. Niessink, N.F. Schneidewind, J. Singer, S. Takada, R. Vehvilainen, and H. Yang, Towards an Ontology of software maintenance, *Journal of Software* Maintenance*: Research and Practice*, **11** (6) (1999) 365-389.

[22] M. Milanović, D. Gašević, A. Giurca, G. Wagner: On Interchanging Between OWL/SWRL and UML/OCL. In: Proceedings of 6th OCL Workshop at the UML/MoDELS Conference (OCLApps 2006), Genova, Italy (2nd October 2006)

[23] M. N. Nguyen, A. I. Wang, and R. Conradi,. Total software process model evolution in EPOS: experience report. In *Proceedings of the 19th international Conference on Software Engineering* (Boston, Massachusetts, United States, May 17 - 23, 1997). ICSE '97 (1997)

[24] OMG, Software Process Engineering Meta-model (SPEM) Specification. Version 2. Technical Report ptc/2008-04-01, Object Management Group (2008)

[25] A. Queraltand E. Teniente Reasoning on UML class diagrams with OCL constraints. In D. W. Embley, A. Olive and S. Ram, editors, ER, volume 4215 of Lecture Notes in Computer Science, pages 497?512. Springer-Verlag, 2006

[26] L. Rising, N.S.Janoff, The Scrum software development process for small teams, *IEEE Software*, **17**(4) (2000) 26-32.

[27] F. Ruiz, A.Vizcaíno, M. Piattini and F. García, , An Ontology for the Management of software Maintenance Projects, International Journal of Software Engineering and Knowledge Engineering, **14**(3) (2004) 323-349.

[28] Sicilia, M.A. (ed.), *Competencies in organizational e-learning. Concepts and Tools*. (Idea Group publishing, Hershey, PA., 2006)

[29] Sicilia, M.A., Garcia-Barriocanal, E., Sanchez-Alonso, S., Rodriguez, D., Ontologies of engineering knowledge: general structure and the case of Software Engineering, *The Knowledge Engineering Review*

[30] J. Simmonds, , M.C. Bastarrica, N. Hitchfeld-Kahler, and E. Rivas, A Tool based on DL for UML Model consistency Checking, *International Journal of Software Engineering and Knowledge Engineering* Vol. 18, No. 6 (2008) 713–735.

[31] M.K. Smith, C. Welty, and D.L. McGuinness, OWL Web Ontology Language Guide". W3C, 2004, `http://www.w3.org/TR/owl-guide/`

[32] C. Tautz, and C.G. von Wangenheim, REFSENO: A Representation Formalism for Software Engineering Ontologies, Fraunhofer Institute IESE IESE-015.98/E (1998).

[33] M. Uschold, M.Grüninger,: Ontologies: Principles, Methods, and Applications, *Knowledge Engineering Review*, **11**(2) (1996) 93-113.

[34] A. Vizcaino, F. Ruiz, M. Piattini, and F. Garcia, 2004. Using REFSENO to Represent Knowledge in the Software Maintenance Process. In *Proceedings of the Database and Expert Systems Applications, 15th international Workshop* (August 30 - September 03, 2004). DEXA. IEEE Computer Society, Washington, DC, 488-493.