Establishing a Common Vocabulary for Software Organizations Understand Software Processes

Ricardo de Almeida Falbo, Gleidson Bertollo
Computer Science Department, Federal University of Espírito Santo, Vitória – ES, Brazil
falbo@inf.ufes.br, gleidsonbertollo@yahoo.com.br

Abstract

Nowadays, several process quality models and standards, such as ISO/IEC 12207 and CMMI, are used to guide software organizations in their software process improvement efforts. But unfortunately, the vocabulary used by those models and by software organizations is diverse. This leads to misunderstanding and problems related to the jointly use of different process quality models. In this paper, we present a software process ontology, which aims to establish a common vocabulary to software organizations talk about software processes. A mapping between the concepts presented in the ontology and the concepts of some of these standards is also done in order to help software organizations to use those standards in their software process improvement efforts.

1. Introduction

Developing quality software is a challenge to software organizations. Since the quality of a software product depends heavily on the quality of the software process used to develop it, software organizations are more and more investing in improving their software processes. In this context, several process quality standards, methodologies, and maturity models, such as ISO/IEC 12207 [1], ISO/IEC 15504 [2], RUP [3] and CMMI [4], are used to guide software organizations efforts towards quality software processes.

But unfortunately, the vocabulary used by those models and by software organizations is diverse. This leads to misunderstanding and problems related to the jointly use of different standards. To deal with these problems, we developed a software process ontology that is presented in this paper. This ontology aims to establish a common vocabulary to software organizations talk about software processes, and was developed as an extension of the software process ontology presented in [5]. It can be used as an interlingua to map concepts from different models and standards, helping software organizations to use them jointly. To show how this can be done, an initial mapping between the software process ontology and the concepts used in ISO/IEC 12207, ISO/IEC 15504, CMMI and RUP is also presented.

This paper is organized as follows: Section 2 discusses briefly software processes and ontologies. Section 3 presents the software process ontology developed. Section 4 presents a mapping between the ontology and the concepts of some process quality standards. Section 5 discusses related works, and, finally, in section 6, we report our conclusions.

2. Software Process and Ontologies

According to Fuggetta [6], a software process can be defined as a coherent set of policies, organizational structures, technologies, procedures, and artifacts that are needed to conceive, develop, deploy, and maintain a software product. A process should be defined considering: the activities to be accomplished, the required resources, the input and output artifacts, the adopted procedures (methods, techniques, templates and so on) and the life cycle model to be used.

To be effective and to lead to good quality products, a software process should be adequate to the application domain and to the specific project itself. Thus, processes should be defined considering several features, such as the type of software being developed, the paradigm adopted, the application domain, team features, and so on.

But, although different projects require processes with specific features, it is possible to establish a set of software process assets that should be present in all project processes. This set of process assets is called an organization's standard software process. Thus, an organizational standard process encompasses the essential process assets (activities, artifacts, resources, procedures) that should be incorporated to all software processes of the organization. Ideally, this process should be defined considering international standards, such as CMMI and ISO/IEC 12207.

This approach can be extended to deal with several levels of standard processes. That is, the organizational standard software process can be specialized to consider some class of software type (such as information system), paradigms (for example, object-oriented paradigm) or specific application domains, giving rise to standard specialized processes.

During process specialization, process assets can be added or modified, according to the context of the specialization (software type, paradigm or application domain). Process specialization can be done recursively. For example, the organizational standard process can be specialized to derive a standard process for object-oriented development, which, in turn, can be specialized for developing object-oriented web applications.

The project's defined software process is developed by tailoring the organization's standard software process or one of its specialized standard processes to fit the specific characteristics of the project. During process tailoring, particularities of the project and team features, among others, should be considered. At this moment, the life cycle model to be followed should be defined, and new activities, as well as consumed and produced artifacts, required resources and procedures, can be added to the project's process.

Successful organizations continuously improve their processes, and systematic process improvement is more effective and efficient if it is done guided by process quality models and standards. The purpose of most standards is to help software organizations achieve excellence by following the processes and activities adopted by the most successful organizations. But it is not easy to select suitable standards. There are many choices, with a large overlap between them. Several times, it is worthwhile for a software organization to use or implement more than one standard at the same time. In this situation, it is better to implement them simultaneously. Such an approach enables process engineers to capitalize on the commonalties between the standards and use the strengths of one standard to offset the weaknesses in the other [7]. But in this case, vocabulary problems arose. Let's take a look at some of these standards.

ISO/IEC 12207 [1] provides a comprehensive set of life cycle processes, activities and tasks for software. Its Process Reference Model provides definitions of processes described in terms of process purpose and outcomes, together with an architecture describing relationships between the processes. It sets out the activities and tasks required to implement the high level life cycle processes to achieve desirable capability for acquirers, suppliers, developers, maintainers and operators of systems containing software. Three life cycle process categories are considered: Organizational, Primary and Supporting. The process model does not represent a particular process implementation approach nor does it prescribe a life cycle model, methodology or technique. Instead the reference model is intended to be tailored by an organization based on its business needs and application domain.

CMMI [4] [7] is structured in terms of process areas (PAs), which consist of related practices that collectively satisfy a set of goals. A generic goal describes the institutionalization required to achieve a capability (continuous representation) or maturity (staged representation) level. Each generic goal is associated with a set of generic practices that describes activities required for institutionalizing processes in a particular PA. Each PA still contains specific goals and specific practices, which describe activities important to achieve the specific goals.

The Rational Unified Process (RUP) [3] is represented using four primary modeling elements: workers, activities, artifacts and workflows. A worker is a role an individual or a group of individuals plays in a project. An activity of a specific worker is a unit of work that an individual in that role may be asked to perform. Activities produce artifacts and can be broken into steps. An artifact is a piece of information that is produced, modified, or used by a process, and can be composed of other artifacts. Artifacts are used as input by workers to perform an activity and are the result or output of such activities. Finally, a workflow is a sequence of activities that produces a result of observable value. These four primary elements represent the backbone of the RUP static structure. But other elements are added to make the process easier to understand and use. These additional elements are: guidelines – rules, techniques, recommendations, or heuristics that describes how to perform an activity or a step; templates – "models" of artifacts, such as a template for the project plan; tool mentors – special guidelines showing how to perform an activity or a step using a specific software tool; and concepts that are introduced in separate sections of the process, usually attached to a core workflow.

In ISO/IEC 15504 [2], process is defined as a set of interrelated or interacting activities which transforms inputs into outputs. Analogous to CMMI and ISO/IEC

12207, standard processes are defined as the set of definitions of the basic processes that guide all processes in an organization. These process definitions cover the fundamental process elements (and their relationships to each other) that must be incorporated into the defined processes that are implemented in projects across the organization. A tailored process is a defined process developed by tailoring a standard process definition. A work product is an artifact associated with the execution of the process.

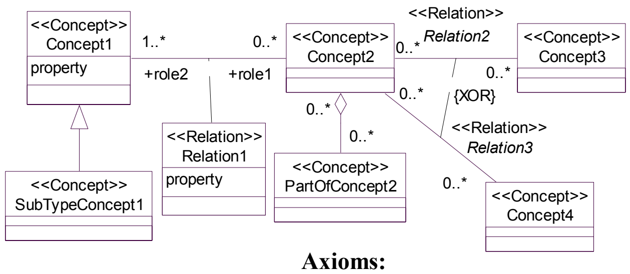
There are a large number of process standards, each one using a slightly different terminology, sometimes with different meaning for the same term, as we can see analyzing the terms and definitions of the four standards previously presented. Thus, we need to establish a common understanding of what is a software process, and which are its main assets. To achieve it, we advocate the use of ontologies.

An ontology is a representation vocabulary, often specialized to some domain or subject matter. More precisely, it is not the vocabulary as such that qualifies as an ontology, but the conceptualizations that the terms in the vocabulary are intended to capture. Ontologies are quintessentially content theories, because their main contribution is to identify specific classes of objects and relations that exist in some domain [8]. Ontologies are used to describe ontological commitments for a set of agents (humans and software applications), that is, agreements to use a shared vocabulary in a coherent manner, so that they can communicate about a domain of discourse.

An ontology, as an engineering artifact, is constituted by a vocabulary used to describe a certain reality, plus a set of explicit assumptions (formal axioms) regarding the intended meaning of the vocabulary words. This set of assumptions has usually the form of a first-order logical theory, where vocabulary words appear as unary or binary predicate names, respectively called concepts and relations [9].

As any software engineering artifact, ontologies must be developed following software engineering practices. To build the software process ontology, we used SABiO (Systematic Approach for Building Ontologies) that encompasses the following activities [5, 10]: purpose identification and requirement specification, ontology capture, ontology formalization, integration of existing ontologies, ontology evaluation, and documentation.

In the requirement specification phase, SABiO uses competency questions to establish the competence of the ontology. During ontology capture, a graphical language for expressing ontologies is used to facilitate the communication between ontology engineers and experts. In its current version, SABiO proposes the use of an UML profile for ontologies [11]. This UML profile uses some UML's model elements playing the same role of the elements of LINGO, the original language proposed [10]. I.e., these UML's model elements are applied using the same semantics imposed by the corresponding elements in LINGO, for which there were some axioms defined. For instance, the axioms (AE1) to (AE4) in Figure 1 are imposed by the whole-part relation, and are assumed to be incorporated to the ontology whenever the aggregation notation of UML is used. Figure 1 shows a summary of the UML profile for expressing ontologies and some of the axioms imposed for the corresponding notation. When any of these notations are used, the corresponding axioms (said epistemological axioms) are supposed to be incorporated, and then they do not need to be written down.



Axioms:

Whole-part:

(AE1) $\forall x\ \neg partOf(x,x)$

(AE2) $\forall x,y\ partOf(y,x) \leftrightarrow wholeOf(x,y)$

(AE3) $\forall x,y\ partOf(y,x) \rightarrow \neg\ partOf(x,y)$

(AE4) $\forall x,y,z\ partOf(z,y) \wedge partOf(y,x) \rightarrow partOf(z,x)$

Sub-type-of:

(AE5) $(\forall x,y,z)\ (subTypeOf(x,y) \wedge subTypeOf(y,z) \rightarrow subTypeOf(x,z))$

(AE6) $(\forall x,y)\ (subTypeOf(x,y) \rightarrow superTypeOf(y,x))$

Or-exclusive (XOR):

(AE7) $(\forall a \in C2)\ ((\exists b)\ (b \in C3) \wedge R2(a,b)) \rightarrow \neg((\exists c \in C4) \wedge R3(a,c)))$

(AE8) $(\forall a \in C2)((\exists c)\ (c \in C4) \wedge R3(a,c)) \rightarrow \neg((\exists b \in C3) \wedge R2(a,b)))$

Figure 1. UML's Profile and its Axioms.

A graphical model, even associated to epistemological axioms, is useful, but it is not enough to completely capture an ontology. Other axioms, called ontological axioms [10], should be provided in order to fix the semantics of the terms, and to establish domain constraints. For formalizing those axioms, SABiO suggests the use of first order logics.

Finally, for ontology evaluation, SABiO suggests checking the ontology against its competency questions, and to verify some quality criteria, as those proposed by Gruber [12].

3. A Software Process Ontology

Analyzing the elements involved in software processes, we can notice that it is a complex domain for building an ontology. As a basic premise, it is essential to follow an approach focusing on minimum ontological commitment. Based on that approach, the ontology should describe only general aspects, valid for any process, with only their essential assets. Including many details in an ontology can make it too specific and thus less reusable.

However, even considering the minimum ontological commitment criterion, this domain is still extremely complex. Therefore, it was necessary to apply a decomposition mechanism allowing building the ontology in parts. The adopted strategy was to define sub-domains of the software process domain, and build sub-ontologies for each sub-domain. Once defined the basic ontologies, these were used in an integrated way to establish a more complete conceptualization about software processes.

Figure 2 shows the software process ontology and its three sub-ontologies: activity ontology, resource ontology and procedure ontology.

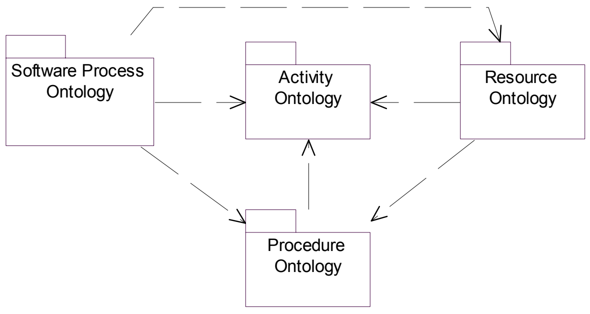


Figure 2. Software Process Ontology and its sub-ontologies.

The software process ontology was originally published in [5]. However, the software process area evolved in the last years, and we needed to evolve this ontology, capturing and defining new concepts, relations and constraints. The ontologies for software activities, procedures and resources were not modified, but only the software process ontology has changed. Thus in this paper we only present the reviewed software process ontology.

Some of the competency questions considered in this new version of the software process ontology includes:

CQ1. How can a process be decomposed?
CQ2. Which are the assets that compose a software process?
CQ3. Which are the inputs and outputs of a process?
CQ4. How can a process be classified?
CQ5. Which is the abstraction level of a process?
CQ6. How can a process be tailored?
CQ7. How do processes interact?
CQ8. How are the activities of a project's software process organized?

To treat these competency questions, some aspects should be taken into account:

- Process Decomposition and Interaction (CQ1 and CQ7);
- Process Definition (CQ2 and CQ3);
- Process Type and Abstraction Level (CQ4 to CQ6);
- Project Process Life Cycle Model (CQ8).

Following, each one of the aspects listed above are discussed and the corresponding models and axioms presented.

3.1. Process Decomposition

A process is defined to establish a systematic approach for developing software, and it can be decomposed into activities or other processes, called sub-processes. For example, according to ISO 12207, the software process can be decomposed into processes for acquisition, supply, development, operation and maintenance, among others. The development process can be further decomposed into other sub-processes, such as requirements engineering process, and so on. The requirements engineering process, in turn, can be decomposed into activities such as requirement elicitation, analysis and negotiation, modeling, documentation, evaluation, and management. Activities can also be decomposed into sub-activities, as shown in Figure 3.

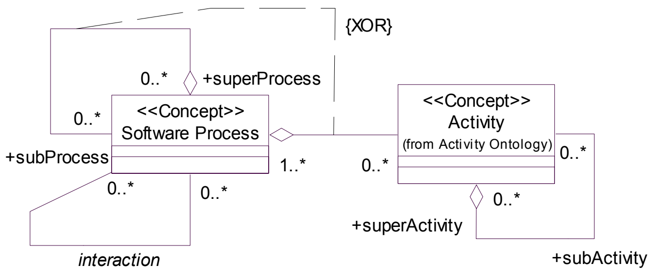


Figure 3. Process decomposition and interaction.

A super-process is the one that is composed by other processes. It cannot be executed directly through activities, as shown by the constraint {XOR} in the model. A sub-process is a software process that composes a larger process, its super-process.

Only to illustrate the epistemological axioms instantiation, the constraint {XOR} in the model of Figure 3 imposes the following axioms, derived from axioms (AE7) and (AE8) in Figure 1.

$$(\forall p1)\ ((\exists p2)\ subProcess(p2,p1)) \rightarrow ((\neg\exists a)\ partOf(a, p1))$$

$$(\forall p1)\ ((\exists a)\ partOf(a, p1)) \rightarrow ((\neg\exists p2)\ subProcess(p2,p1)$$

Finally, a software process can interact with other processes. This interaction can be in several ways, among them: a process can precede the execution of other, two processes can be executed in parallel, or a process can be executed in a specific moment during the execution of another process.

3.2. Process Definition

As discussed above, a process is composed by sub-processes or activities. During process definition, several other process assets should be defined. For each activity of the software process, we should define its sub-activities, pre-activities, input and output artifacts, required resources (humans, software and hardware) and the procedures (methods, techniques etc) to be followed when performing the activity. Figure 4 presents the process assets involved in software process definition.

The major part of this model corresponds to the activity ontology presented in [5]. Since in this paper we are focusing only on the evolution of the software process ontology, we will discuss only those aspects related to this review.

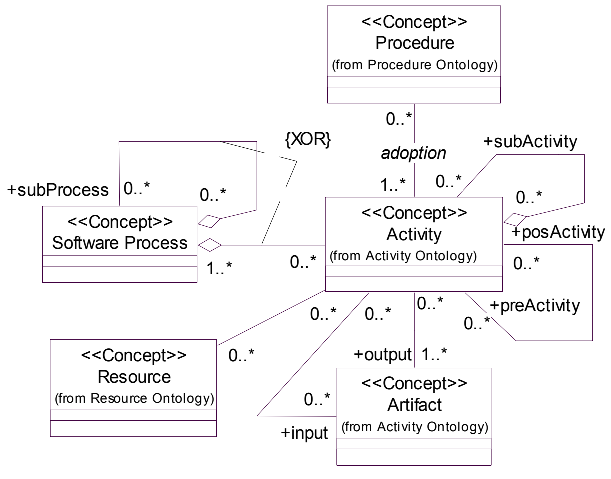


Figure 4. Process definition.

An activity is a transformational action that can produce artifacts. To be performed, an activity requires resources, adopts procedures and consumes artifacts. In a similar way, we can say that a software process has inputs and outputs. Its inputs and outputs are directly related to its activities' inputs and outputs. That is, if an activity *a1*, part of a software process *p*, requires as input an artifact *s*, and there isn't another activity *a2*, part of the same process *p*, that produces this artifact, then *s* is said an input of *p*.

$$\forall (p, a1, s)\ partOf\ (a1, p) \wedge input(s, a1) \wedge ((\neg\exists a2)\ partOf(a2, p) \wedge output\ (s, a2)) \rightarrow input\ (s, p)$$

Concerning outputs, we can say that the outputs of a process correspond to the outputs of their activities.

$$\forall (p, a1,s)\ partOf\ (a1,p) \wedge output\ (s,a1) \rightarrow output(s,p)$$

In an analogous manner, a super-process has its inputs and outputs defined through the inputs and outputs of its sub-process, as described by the following axioms:

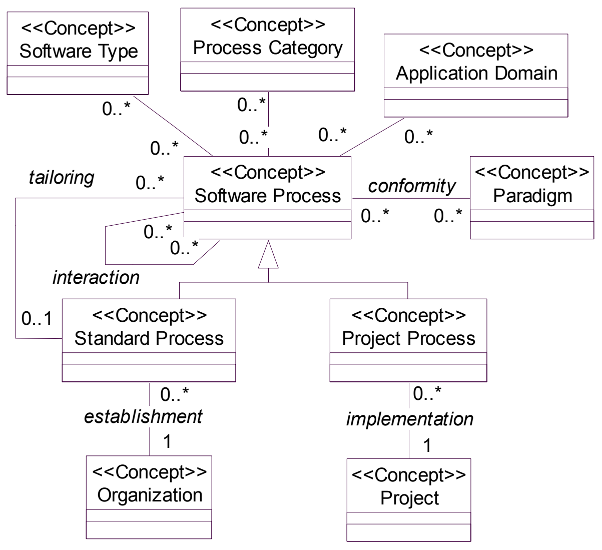
$$\forall (p1,p2,s)\ subProcess\ (p2,p1) \wedge output\ (s, p2) \wedge ((\neg\exists p3)\ subProcess\ (p3, p1) \wedge input(s, p3)) \rightarrow input\ (s, p1)$$

$$\forall (p1,p2,s)\ (subProcess(p2,p1) \wedge output(s, p2) \rightarrow output\ (s, p1)$$

3.3. Process Type and Abstraction Level

As shown in Figure 5, processes can be classified in process categories. For example, if an organization follows the ISO/IEC 12207 classification, the categories could be primary processes, supporting processes, and organization processes. Furthermore, processes are in different levels of abstraction. A standard process refers to a generic process institutionalized in an organization, establishing basic requirements for processes to be performed in that organization. A project process refers to the process defined for a specific project, considering the particularities of that project.

Software processes (standard or project processes) can be defined tailoring a standard process. When a standard process is tailoring another standard process, the tailored process is called a specialized process, and every process assets defined in the standard process become part of the specialized process. But new assets can also be included, to deal with features of a specific software type, paradigm or application domain.



<<Concept>> Software Type
<<Concept>> Process Category
<<Concept>> Application Domain
<<Concept>> Software Process
<<Concept>> Paradigm
<<Concept>> Standard Process
<<Concept>> Project Process
<<Concept>> Organization
<<Concept>> Project
tailoring
conformity
interaction
establishment
implementation
0..*
0..1
1

Figure 5. Process Types and Abstraction Levels.

When a project process is defined by tailoring a standard process, it is composed by all the standard process assets, and new assets can be included considering the project characteristics, such as complexity, size, and team experience, among others.

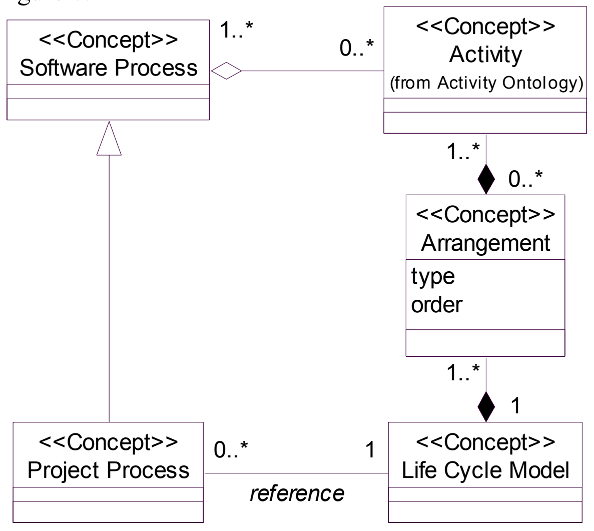
It is worthwhile to point that the interaction between processes must occur at the same level of abstraction. I.e. a standard process can interact only with other standard process and a project process can interact only with other project process.

$$\forall(p1,p2)\ (interaction\ (p1,p2) \rightarrow (standardProcess\ (p1) \wedge standardProcess\ (p2)) \vee (projectProcess\ (p1) \wedge projectProcess\ (p2))$$

3.4. Project Process Life Cycle Model

The project process definition starts with the choice of a life cycle model to be used as reference. A life cycle model structures the project activities in phases (or macro-activities), establishing an approach for organizing those macro-activities. Looking for the main life cycle models described in the literature, we can notice that macro-activities are grouped in arrangements that follow two basic strategies: sequence and iteration. In sequential arrangements, the phases are just accomplished once, returning to the previous phase only for correcting possible problems detected. In iterative arrangements, a set of phases is accomplished several times, according to some established criterion.

The waterfall life cycle model (or linear sequential model) [13], for instance, can be described as a single sequential arrangement of all phases. The spiral model [13] can be described also by only one arrangement, but in this case it is an iterative arrangement. Other life cycle models, like the recursive / parallel model [13] or the incremental model, can be described as several arrangements of activities, some of them sequential, some iterative. Thus, all life cycle models can be mapped as hybrid (sequential and iterative) arrangement of macro-activities. This way, a life cycle model defines a set of macro-activities (or phases) that a development process should present and the order of execution in the form of arrangements, as shown in figure 6.



<<Concept>> Software Process
<<Concept>> Activity (from Activity Ontology)
<<Concept>> Arrangement
type
order
<<Concept>> Project Process
<<Concept>> Life Cycle Model
reference
1..*
0..*
1

Figure 6. Project Process and Life Cycle Models.

Since the starting point for defining project processes is the life cycle model adopted, the initial project process' structure must correspond to the set of macro-activities that compose the life cycle model. That is, if a project process *p* adopts as a reference the life cycle model *m*, then each activity *a* that is part of an arrangement *c* of the life cycle model *m* must also be part of *p*.

$$\forall(p, m, c, a, n)\ (reference\ (p,m) \wedge partOf\ (c,m) \wedge partOf\ (a, c)) \rightarrow partOf\ (a, p))$$

4. Mapping Standards to the Ontology

Once defined the software process ontology, it is worthwhile to map the structure of the standards into the concepts of the ontology. It is worthwhile to point that some standards, such as ISO 9001:2000 and CMMI, are not software specific. But our mapping is focusing only on software organizations and, thus, we looked for those standards using only this perspective.

Table 1 shows a preliminary mapping between the vocabulary used by ISO/IEC 12207, ISO 9001:2000 and ISO/IEC 15504 and the concepts of the ontology presented in this paper.

Table 1. ISO x Software Process Ontology

ISO	Software Process Ontology
Process	Software Process
Standard Process	Standard Process
Tailored Process	Project Process
Work Product	Artifact
Activity / Task	Activity
Process Category	Process Category
Process	Software Process
Life Cycle Model	Life Cycle Model

Table 2 shows a preliminary mapping between the vocabulary used by CMMI and the concepts of the ontology presented in this paper.

Table 2. CMMI x Software Process Ontology

CMMI	Software Process Ontology
Process	Software Process
Standard Process	Standard Process
Defined Process (or Project's Defined Process)	Project Process
Work Product	Artifact
Practice	Activity
Process Area	Software Process
Project	Project
Life Cycle Model	Life Cycle Model

Finally, Table 3 shows a preliminary mapping between the vocabulary used by RUP and the concepts of the software process ontology.

Table 3. RUP x Software Process Ontology

RUP	Software Process Ontology
Process / Workflow	Software Process
Process Framework	Standard Process
Worker	Human Resource
Artifact	Artifact
Activity / Step	Activity
Guideline / Tool Mentor / Template	Procedure

5. Related Work

There are several works exploring the mapping between standards. Mustafelija and Stromberg [7], for example, maps ISO 9001:2000 sections to CMMI and vice versa. These mappings, however, are based on the content of the standards, and not on their structures. In fact, there are very few works dealing with the problem of establishing a common understanding about software processes. The most important of them is the OMG's Software Process Engineering Metamodel (SPEM) [14], which is used to describe a concrete software development process or a family of related software development processes.

Like our ontological approach, SPEM intends to define the minimal set of process modeling elements necessary to describe any software development process, without adding specific models or constraints for any specific area.

At the core of SPEM is the idea that a software development process is a collaboration between abstract active entities called *process roles* that perform operations called *activities* on concrete, tangible entities called *work products*.

SPEM follows an object-oriented approach for modeling a family of related software processes, and its specification is structured as a UML profile, and provides a complete MOF-based metamodel.

In our point of view, the main problem of SPEM is exactly this approach. Several non intuitive concepts are used to define concepts related to software process. Abstract concepts, such as Model Element, Package, Work Definition and Process Performer, are not intuitive for process engineers. In fact, they are used only because an object-oriented approach, focused on inheritance, is applied. If we look for the concrete classes in SPEM, we can find a great correspondence with our ontology, as we can notice in Table 4. This table shows the mapping of some concepts of SPEM into concepts of the Software Process Ontology.

Table 4. SPEM x Software Process Ontology

SPEM	Software Process Ontology
Process Role	Human Resource / Team
Work Product	Artifact
Activity / Step	Activity
Guidance	Procedure
Categorizes Dependency	Process Category
Process	Software Process
Life Cycle	Life Cycle Model

It is worthwhile to point that, although we use an UML profile as a modeling language for expressing ontologies, we do not follow an approach like SPEM. In our case, we defined an UML profile using stereotypes to capture our meta-ontology, which includes concepts such as Concept, Relation, Property and so on [11]. We are not using UML's meta-model as basis for defining our software process ontology, as SPEM does.

6. Conclusions and Future Work

Nowadays, software process improvement is being considered essential to software organizations survive in a competitive market. But systematic process improvement is achieved only if it is done guided by process quality models and standards. Several times, it is important to use more than one standard, so that the strengths of one standard can be used to offset the weaknesses in the other, and vice versa. But in this case, we face problems related to the vocabularies used by the different standards. Generally, each standard uses its own terminology, adopting different terms to designate the same meaning. To overcome this problem, we need to establish a common conceptualization about software processes, and thus ontologies can be useful. Thus, in this paper, we presented an ontology of software process that defines the main concepts, relations, properties and constraints involved in this complex domain. Also, a preliminary mapping between the concepts in the ontology and the concepts used by some of the most important standards was done. We hope that this mapping can be used by software organizations to better understand the commonalties and differences between the various standards.

As future work, we are planning to do a more complete mapping between these standards, using our ontology as an interlingua. A mapping considering the contents of those standards is also useful.

Finally, we are working on a process infrastructure for ODE [15], a Process-Centered Software Engineering Environment that is developed based on our ontology and that can be configurable for using the most adequate vocabulary, given by the choice of a standard that a software organization commits to. ODE is developed following a systematic approach for deriving object models from ontologies, and the ontology presented in this paper is used to derive the core classes of process control in ODE, supporting tool integration and interoperability in it.

7. Acknowledgments

This work was accomplished with the support of CNPq, an entity of the Brazilian Government reverted to scientific and technological development.

8. References

[1] ISO/IEC 12207:1995, Amd 1:2002, Amd 2:2004, *Information Technology - Software life cycle processes*.

[2] ISO/IEC 15504:2003, *Information Technology – Process Assessment.*

[3] P. Kruchten, *The Rational Unified Process: An Introduction*, Addison Wesley, 1998.

[4] M.B. Chrissis, M. Konrad, S. Shrum, *CMMI: Guidelines for Process Integration and Product Improvement*, Addison Wesley, 2003.

[5] Falbo, R.A., Menezes, C.S., Rocha, A.R.R. A Systematic Approach for Building Ontologies. *Proceedings of the 6th Ibero-American Conference on Artificial Intelligence*, Lisbon, Portugal, Lecture Notes in Computer Science, vol. 1484, 1998.

[6] A. Fuggetta, "Software Process: A Roadmap", In *Proceedings of The Future of Software Engineering (ICSE'2000)*. Limerick, Ireland, 2000, 25-34.

[7] B. Mutafelija, H. Stromberg, *Systematic Process Improvement Using ISO 9001:2000 and CMMI*, Artech House, 2003.

[8] Chandrasekaran, B., Josephson, J.R., Benjamins, V.R. What Are Ontologies, and Why Do We Need Them? *IEEE Intelligent Systems*, January/February 1999.

[9] Guarino, N. Formal Ontology and Information Systems. In: *Formal Ontologies in Information Systems*, N. Guarino (Ed.), IOS Press, 1998.

[10] Falbo, R. A. Experiences in Using a Method for Building Domain Ontologies. In: *Proc. of the 16th International Conference on Software Engineering and Knowledge Engineering, International Workshop on Ontology In Action*. Banff, Canada, 2004.

[11] Mian, P.G., Falbo, R.A. Supporting Ontology Development with ODEd, *Journal of the Brazilian Computer Science*, vol. 9, no. 2, November 2003, 57-76.

[12] Gruber, T. Towards principles for the design of ontologies used for knowledge sharing, *International Journal of Human-Computer Studies*, 43(5/6), 1995.

[13] R.S. Pressman, *Software Engineering: A Practitioner's Approach*, 6th Edition, McGraw Hill, 2004.

[14] OMG, *Software Process Engineering Metamodel Specification*, Version 1.1, January 2005.

[15] R.A. Falbo, F. B. Ruy, R. Dal Moro. Using Ontologies to Add Semantics to a Software Engineering Environment. In: *Proc. of the 17th International Conference on Software Engineering and Knowledge Engineering*. Taipei, China, 2005.