A software process ontology as a common vocabulary about software processes

Ricardo de Almeida Falbo* and Gleidson Bertollo

Computer Science Department,
Federal University of Espírito Santo,
Vitória – ES, Brazil
E-mail: falbo@inf.ufes.br E-mail: gleidsonbertollo@yahoo.com.br
*Corresponding author

Abstract: Nowadays, several process quality models and standards, such as ISO/IEC 12207 and CMMI, are used to guide software organisations in their software process improvement efforts. Unfortunately, the vocabulary used by those models and by software organisations 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 for software organisations to 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 organisations to use those standards in their software process improvement efforts.

Keywords: software process; ontology; process quality models.

Reference to this paper should be made as follows: Falbo, R.A. and Bertollo, G. (xxxx) 'A software process ontology as a common vocabulary about software processes', *Int. J. Business Process Integration and Management*, Vol. X, No. Y, pp.000–000.

Biographical notes: R.A. Falbo is an Associate Professor at the Computer Science Department from Federal University of Espírito Santo, Brazil. He is currently heading the Software Engineering Laboratory (LabES) of Federal University of Espírito Santo. His main research interests are on software quality, ontologies, software engineering environments, and knowledge management.

G. Bertollo received his MSc in Computer Science from the Federal University of Espírito Santo, Brazil, in 2006, under the supervision of R.A. Falbo.

1 Introduction

Developing quality software is a challenge to software organisations. Since the quality of a software product depends heavily on the quality of the software process used to develop it, software organisations 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 (ISO/IEC, 1995), ISO/IEC 15504 (ISO/IEC, 2003), rational unified process (RUP) (Kruchten, 1998) and CMMI (Chrissis et al., 2003), are used to guide software organisations efforts towards quality software processes.

Unfortunately, the vocabulary used by those models and by software organisations 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 for software organisations to talk about software processes, and was developed as an evolution of the software process ontology partially presented in Falbo et al. (1998). One of our main goals is that this ontology is use as an interlingua for mapping concepts from different models and standards, helping software organisations to use them jointly. To show how this can be done, a basic mapping between the software process ontology and the concepts used in ISO/IEC 12207, ISO/IEC 15504, CMMI and RUP are also presented.

This paper is organised as follows: Section 2 discusses briefly software processes and some of the most popular standards. Section 3 talks about ontologies and presents the method adopted for developing the software process ontology. Sections 4 and 5 present the software process ontology. Section 6 presents a basic mapping between the ontology and the concepts of some process quality standards. Section 7 discusses related works, and finally in Section 8, we report our conclusions and future work.

2 Software process

According to Fuggetta (2000), a software process can be defined as a coherent set of policies, organisational structures, technologies, procedures and artefacts 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 artefacts, 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.

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 of an organisation. This set of process assets is called an organisational standard software process. The project's defined software process is developed by tailoring the organisation's standard software process to fit the specific characteristics of the project (Paulk et al., 1993). The standard process defines a common structure to be followed by all projects, no matter which characteristics the software to be developed has.

According to Humphrey (1990), there are several reasons for defining a standard process, such as:

- standard software processes minimise problems related to training, revisions and tool support
- the experiences acquired in the projects can be incorporated to the standard process, contributing to improvements in all projects' processes
- less time and effort are spent in defining projects' processes.

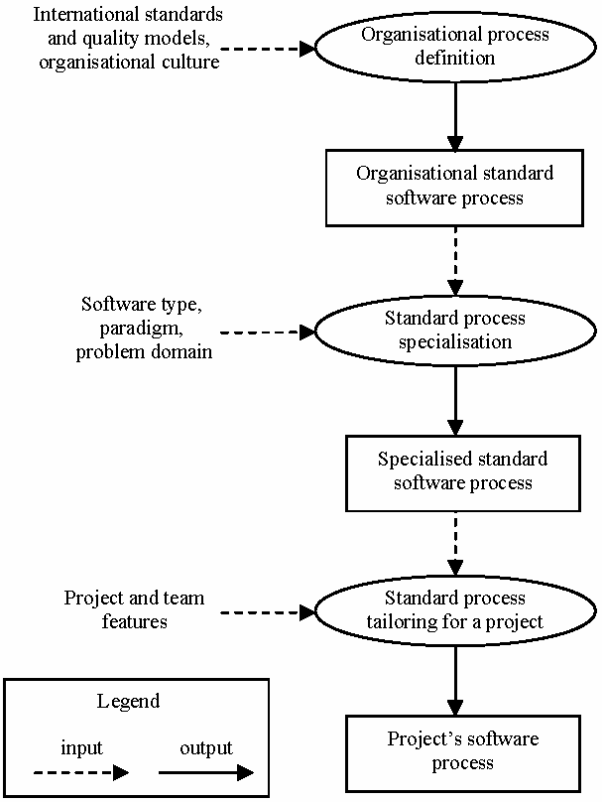
This tendency in using standard processes is advocated by almost every quality models and standards, including ISO/IEC 12207, ISO/IEC 15504 and CMMI. All of them suggest the use of a standard process as the starting point from which project's processes can be established.

This approach based on organisational standard processes can be extended to deal with several levels of standard processes. That is, the organisational standard software process can be specialised to consider some class of software type, paradigm or application domains; giving rise to standard specialised processes.

As shown in Figure 1, initially a software organisation should define its organisational standard software process. This process encompasses the essential process assets (activities, artefacts, resources, procedures) that should be incorporated to all software processes of the organisation. Ideally, it should be defined considering international standards, such as CMMI and ISO/IEC 12207, and the organisational culture.

Next, the organisational standard process can be specialised to better fit some class of software type (such as information system, web application and so on), paradigm (for example, object-oriented paradigm) or specific application domains. As a result of the standard process specialisation step, a set of standard specialised processes is defined.

Figure 1 Multi-levelled approach for process definition



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

Finally, the organisational process or one of its specialised standard processes can be tailored for a specific project. In other words, the project's defined software process is defined by tailoring the organisational standard process or one of its specialised standard processes, taking into account 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 in the project should be defined, and new activities, as well as consumed and produced artefacts, resources required and procedures to be adopted can be added to the project's process.

Successful organisations continuously improve their processes. Like organisational standard process definition, 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 organisations achieve excellence by following the processes and activities adopted by the most successful organisations.

However 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 organisation to use or implement more than one standard at the same time. In this situation, it is better to implement them simultaneously. Such approach enables process engineers to capitalise on the commonalities between the standards and uses the strengths of one standard to offset the weaknesses in the other (Mutafelija and Stromberg, 2003). Some of these standards are discussed below.

ISO/IEC 12207 (ISO/IEC, 1995) provides a comprehensive set of life cycle processes, activities and tasks for software engineering. Its process reference model provides definitions of processes, described in terms of process purposes and outcomes, together with an architecture describing the relationships between the processes. It sets out the activities and tasks required to implement the high level life cycle processes in order to achieve desirable capability for acquirers, suppliers, developers, maintainers and operators of systems containing software. Three life cycle process categories are considered: primary, organisational and supporting. The process model does not represent a particular process implementation approach nor prescribes a life cycle model, methodology or technique. Instead the reference model is intended to be tailored by an organisation based on its business needs and application domain.

CMMI (Chrissis et al., 2003) 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 institutionalisation 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 institutionalising processes in a particular PA. Each PA still contains specific goals and specific practices, which describe activities that are important to achieve the specific goals.

The RUP (Kruchten, 1998) is represented using four primary modelling elements: workers, activities, artefacts 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 artefacts and can be broken into steps. An artefact is a piece of information that is produced, modified or used by a process, and can be composed of other artefacts. Artefacts are used as input by workers to perform an activity and are the result (output) of an activity. 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. Other elements are added to make the process easier to understand and use. Among them are: guidelines (rules, techniques, recommendations, or heuristics that describes how to perform an activity), templates ('models' of artefacts, such as a template for the project plan), and tool mentors (special guidelines showing how to perform an activity using a specific software tool).

In ISO/IEC 15504 (ISO/IEC, 2003), 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 organisation. 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 organisation. A tailored process is a project's defined process, developed by tailoring a standard process. A work product is an artefact associated with the execution of the process.

There is 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 by analysing 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 this common conceptualisation about software processes, we advocate the use of ontologies.

3 Ontology

According to Chandrasekaran et al. (1999),

> "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".

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.

Ontology as an engineering artefact 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 (Guarino, 1998).

As any software engineering artefact, ontologies must be developed following software engineering practices. To build the software process ontology, we used Systematic Approach for Building Ontologies (SABiO) (Falbo et al., 1998; Falbo, 2004). SABiO encompasses the following activities:

- Purpose identification and requirement specification: concerns to clearly identify the ontology purpose and its intended uses, i.e., the competence of the ontology.

- Ontology capture: the goal is to capture the domain conceptualisation based on the ontology competence. Relevant concepts and relations should be identified and organised. A model using a graphical language and a dictionary of terms should be used to aid communication with domain experts.
- Ontology formalisation: aims to explicitly represent the conceptualisation captured in a formal language.
- Integration of existing ontologies: during ontology capture or formalisation, it could be necessary to integrate the current ontology with existing ones, in order to use previously established conceptualisations.
- Ontology evaluation: the ontology must be evaluated to check whether it satisfies the specification requirements.
- Documentation: all the ontology development must be documented.

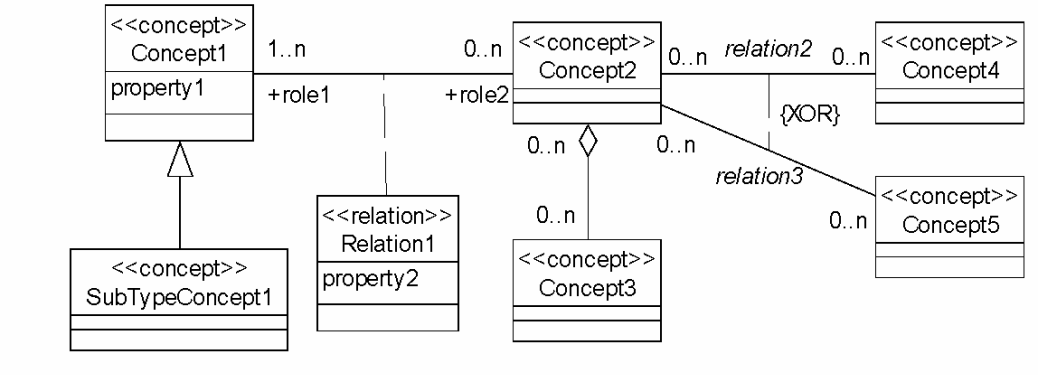
In the requirement specification phase, to establish the competence of the ontology, SABiO suggests the use of competency questions, i.e., the questions that the ontology should be able to answer (Gruninger and Fox, 1995).

During ontology capture, a graphical language for expressing ontologies should be used to facilitate the communication between ontology engineers and experts. In its current version, SABiO defines an UML profile for ontologies (Mian and Falbo, 2003) that represents concepts, relations and properties, and that defines some formal axioms for some kinds of relations, such as subsumption and whole-part relations. Figure 2 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 domain independent axioms) are supposed to be incorporated, and then they do not need to be written down. For instance, the axioms (A1) to (A5) in Figure 2 are imposed by the whole-part relation, and are assumed to be incorporated to the ontology whenever the aggregation notation of UML is used.

A graphical model, even associated to domain independent axioms, is useful, but it is not enough to completely capture an ontology. Besides the domain independent axioms, other axioms can be used to represent knowledge that is domain specific. These axioms can be of two types: *consolidation axioms* and *derivation axioms*. The former aims to impose constraints that must be satisfied for a relation to be consistently established. The latter intends to represent declarative knowledge that is able to derive knowledge from the factual knowledge represented in the ontology, describing domain specific constraints. For formalising those axioms, SABiO suggests the use of first order logics.

In ontology evaluation, SABiO suggests checking the ontology against its competency questions, and to verify some quality criteria, as those proposed by Gruber (1995).

Figure 2 UML's profile for building ontologies and its axioms



Axioms:

Whole-part:

(A1) $(\forall x)\ \neg\, partOf(x, x)$
(A2) $(\forall x, y)\ partOf(y, x) \leftrightarrow wholeOf(x, y)$
(A3) $(\forall x, y)\ partOf(y, x) \rightarrow \neg\, partOf(x, y)$
(A4) $(\forall x, y, z)\ partOf(z, y) \wedge partOf(y, x) \rightarrow partOf(z, x)$
(A5) $(\forall x, y)\ partOf(y, x) \rightarrow ((\exists z)\ partOf(z, x))$

Sub-type-of:

(A6) $(\forall x, y, z)\ subTypeOf(x, y) \wedge subTypeOf(y, z) \rightarrow subTypeOf(x, z)$
(A7) $(\forall x, y)\ subTypeOf(x, y) \rightarrow superTypeOf(y, x)$
(A8) $(\forall x, y)\ subTypeOf(y, x) \rightarrow \neg\, subTypeOf(x, y)$

Or-exclusive (XOR):

(A9) $(\forall a \in Concept2)\ ((\exists b)\ (b \in Concept4) \wedge relation2(a, b)) \rightarrow \neg\, ((\exists c \in Concept5) \wedge relation3(a, c)))$
(A10) $(\forall a \in Concept2)\ ((\exists c)\ (c \in Concept5) \wedge relation3(a, c)) \rightarrow \neg\, ((\exists b \in Concept4) \wedge relation2(a, b)))$

Finally, for documentation purposes, SABiO advocates the use of hypertexts. Using a hypertext, concepts can be easily linked to relations, properties, ontology diagrams, dictionaries of terms, axioms, and competency questions. This way, people can browse the ontology to learn about the domain (Falbo et al., 2002; Falbo, 2004).

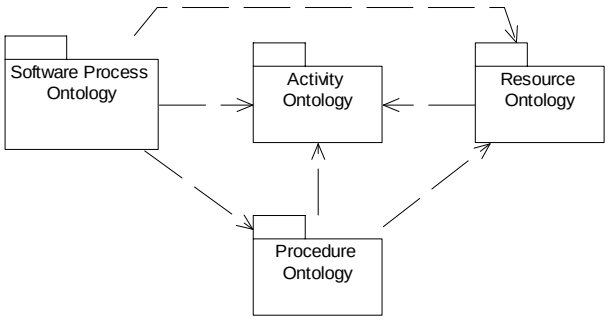
4 Software process sub-ontologies

Analysing the elements involved in software processes, as discussed in Section 2, 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 (Gruber, 1995). Based on that approach, the ontology should describe only general aspects, valid for the process field of study, 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 the building of 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 were used in an integrated way to establish a more complete conceptualisation about software processes.

Figure 3 shows the software process ontology and its three main sub-ontologies: activity, resource and procedure ontologies. This decomposition was inspired in TOVE´s Project (Fox and Gruninger, 1994), which considers activity and resource ontologies as the basis for other ontologies for enterprise modelling. In Figure 3 the dependency relations indicate that concepts and relations of an ontology are used by another. Thus, the activity ontology is the core ontology, which concepts and relations are used by the others. The software process ontology, in turn, is developed using the three sub-ontologies.

Figure 3 Software process ontology and its sub-ontologies



The software process ontology was originally published in Falbo et al. (1998). However, the software process area has evolved in the last years, and we needed also to evolve the ontology, capturing and defining new concepts, relations and constraints. Thus, in this paper we are presenting, in fact, an evolution of the software process ontology.

Since the software process ontology depends heavily on the other three ontologies shown in Figure 3, we start presenting these sub-ontologies in this section. In Section 5, we present the software process ontology properly said. Due to space limitations, the sub-ontologies are presented only partially.

4.1 Activity ontology

The concept of activity is at the core of any software process model. Activities can occur in several levels, from an elementary task to a phase of the development process. An activity is a piece of work to be done that uses artefacts as input and produces artefacts as output. To be performed, an activity requires resources and adopts procedures.

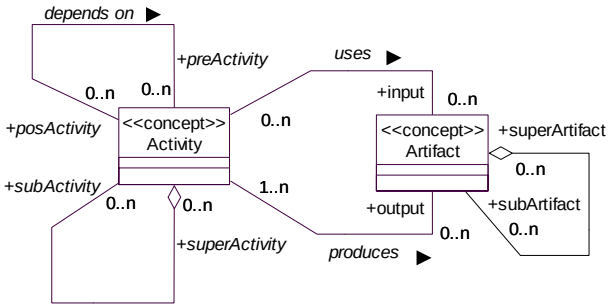
Basically, an activity ontology should be able to answer the following competency questions:

- In which sub-activities is an activity decomposed?
- Which activities must precede a given activity?
- Which artefacts are input to a given activity?
- Which artefacts are produced by a given activity?
- Which resources are required by an activity to be performed?
- Which procedures can be adopted to perform an activity?

The last two competency questions show, respectively, the interactions between the activity ontology and the resource and procedure ontologies, and are discussed latter.

Figure 4 shows a partial model of the activity ontology.

Figure 4 Activity ontology (partial model) (see online version for colours)



As shown in this figure, an activity can be decomposed in other activities, said sub-activities. Activities use and produce artefacts, and some activities depend on the accomplishment of other activities, said pre-activities. Artefacts can be decomposed in sub-artefacts.

As discussed in Section 3, the use of an UML profile for expressing ontologies impose some axioms that do not need to be written down. To illustrate the domain independent axioms instantiation, consider the whole-part relation between activities. According to the UML profile adopted,

the following axioms hold. They are derived from axioms (A1) to (A5) in Figure 2, respectively:

$(\forall\ a) \neg\ subActivity\ (a, a)$

$(\forall\ a1, a2)\ subActivity\ (a2, a1) \leftrightarrow superActivity\ (a1, a2)$

$(\forall\ a1, a2)\ subActivity\ (a1, a2) \rightarrow \neg\ subActivity\ (a2, a1)$

$(\forall\ a1, a2, a3)\ subActivity\ (a1, a2) \wedge subActivity\ (a2, a3) \rightarrow subActivity\ (a1, a3)$

$(\forall\ a1, a2\ )\ subActivity\ (a1, a2) \rightarrow (\exists\ a3)\ subActivity\ (a3, a2)$

There are several other axioms that are not captured by the UML profile notation. Those axioms must be written, such as the one that says that if an activity $a1$ produces an artefact s that is used by an activity $a2$, then $a1$ is a pre-activity of $a2$.

$(\forall\ a1, a2, s)\ output\ (s, a1) \wedge input\ (s, a2) \rightarrow preActivity\ (a1, a2)$

Like the whole-part relation, the precedence relation (*depends on*) is also anti-reflexive, asymmetric and transitive. Thus, the following axioms hold:

$(\forall\ a) \neg\ preActivity\ (a, a)$

$(\forall\ a1, a2)\ preActivity\ (a1, a2) \rightarrow \neg\ preActivity\ (a2, a1)$

$(\forall\ a1, a2, a3)\ preActivity\ (a1, a2) \wedge preActivity\ (a2, a3) \rightarrow preActivity\ (a1, a3)$

$(\forall\ a1, a2)\ preActivity\ (a1, a2) \leftrightarrow posActivity\ (a2, a1)$

4.2 Resource ontology

As briefly commented in the activity ontology, some elements are required to the accomplishment of an activity, such as people, hardware and software. These elements are said resources. The resource ontology should deal with the following competency questions:

- Which resources are required by an activity?
- What is the nature of a resource?

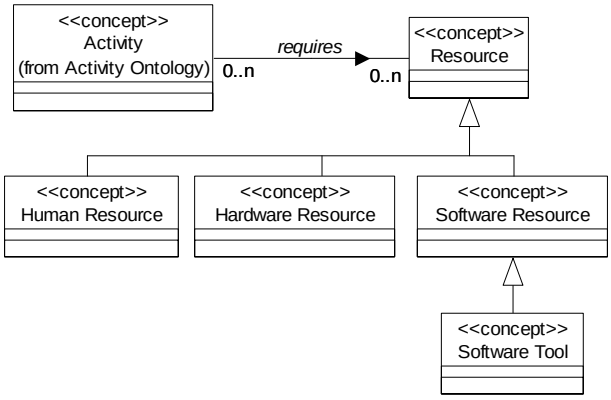
Figure 5 shows the resource ontology. As shown in this figure, resources are used during, or to support, the execution of activities. These resources can be grouped into three main categories:

1 Human resource are the roles that human agents is required to perform in an activity, such as requirement analyst, project manager, client and so on.

2 Hardware resources include any hardware equipment required to perform an activity, such as computers and printers.

3 Software resources concern any software product that is used in the accomplishment of an activity, such as a network management software or a database management system. Since in the context of software processes CASE tools are very important, they are considered explicitly as a subtype of software resources.

Figure 5 Resource ontology (partial model)



When a resource r is required by an activity $a1$ that is part of an activity $a2$, then we should say that the whole activity $a2$ also requires r.

$(\forall\ a1, a2, r)\ requires\ (a1, r) \wedge subActivity\ (a1, a2) \rightarrow$ requires $(a2, r)$

4.3 Procedure ontology

During software process definition, it is important to define how the activities will be performed. To do that, we must establish procedures to be adopted in the accomplishment of the activities.

There are several types of procedures. Methods, for example, define a set of steps to be performed in an activity. Tool mentors, on the other hand, describe how to use a software tool in the accomplishment of an activity. Templates are models to be followed when preparing an artefact in an activity. Thus, a procedure ontology should be able to answer the following competency questions:

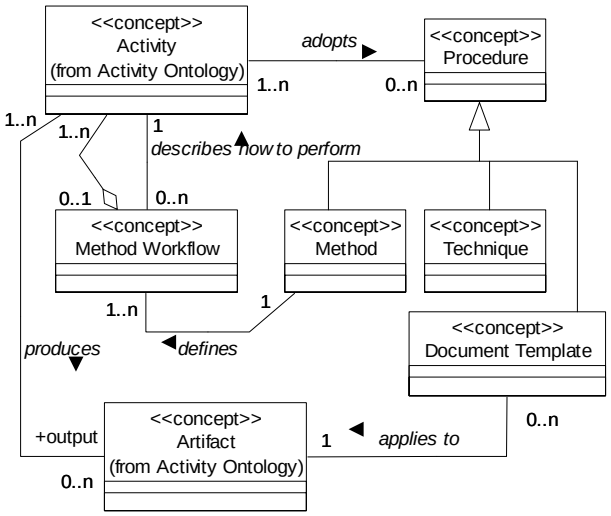
- Which are the procedures that can be adopted in the accomplishment of an activity?
- What is the nature of a procedure?
- How does a specific kind of procedure guide a developer in performing an activity?

Figure 6 shows part of the procedure ontology. As shown in this figure, procedures are adopted to aid developers in performing activities. These procedures can be grouped into several categories. It is not our intention to fix a rigid procedure taxonomy. Some classes of procedures that we envision include methods, techniques and document templates. Others could be added to this taxonomy, such as tool mentors, scripts, naming conventions and so on.

As previously said, a method is a systematic procedure that defines a workflow of activities (a set of steps) and heuristics to perform one or more activities. When a method can be adopted in the accomplishment of more than one activity, it has a workflow of activities for each one. For example, several object-oriented methods prescribe different

workflows of activities for analysis and design. Thus the following consolidation axiom should be valid:

Figure 6 Procedure ontology (partial model)



1 If a method workflow *mw* describes how to perform an activity *a*, then *a* can not be part of the method workflow *mw*:

$(\forall\ mw, a)\ describesHowToPerform\ (mw, a) \rightarrow \neg\ partOfMethodWorkflow\ (a, mw)$

2 If a method workflow *mw* describes how to perform an activity *a*, then *a* should adopt the method *m* that defines the method workflow *mw*:

$(\forall\ mw, a)\ describesHowToPerform\ (mw, a) \rightarrow adopts\ (a, m) \wedge defines\ (m, mw)$

3 If an activity *a*1 is part of the method workflow *mw* that describes how to perform the activity *a*, then *a*1 should be a sub activity of *a*:

$(\forall\ mw, a, a1)\ partOfMethodWorkflow\ (a1, mw) \wedge describesHowToPerform\ (mw, a) \rightarrow subActivity\ (a1, a)$

A technique is a procedure to perform an activity, which is less rigid and detailed than a method, in the sense that it does not prescribe a set of activities. However it still provides some heuristics to perform an activity. Testing techniques, such as black-box and white-box testing are examples of techniques.

Finally, a document template aims to establish a standard way for preparing some artefact (in this case, a document) and thus the following consolidation axiom should be considered: If an activity *a* adopts a document template *dt* that applies to an artefact *s*, then *s* should be an output of *a*.

$(\forall\ a, dt, s)\ adopts\ (a, dt) \wedge appliesTo\ (dt, s) \rightarrow output\ (s, a)$

5 The software process ontology

Since the software process ontology is the main focus of this paper, it is presented in more details than the other ontologies presented in the previous section.

The main competency questions considered in the development of the software process ontology are:

CQ1 How can a process be decomposed?

CQ 2 Which are the assets that compose a software process?

CQ 3 Which are the inputs and outputs of a process?

CQ 4 How can a process be classified?

CQ 5 Which is the abstraction level of a process?

CQ 6 How can a process be tailored?

CQ 7 How do processes interact?

CQ 8 How are the activities of a project's software process organised?

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).

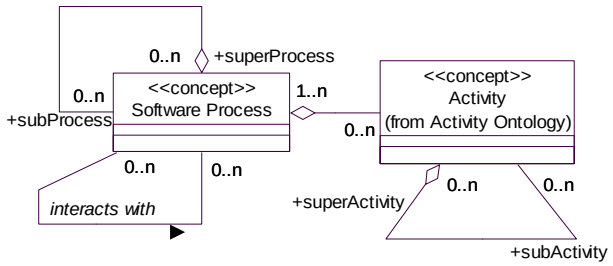
Following, each one of the aspects listed above are discussed and the corresponding models and axioms are presented.

5.1 Process decomposition and interaction

A process is defined to establish a systematic approach for developing or maintaining software and it can be decomposed into activities or other processes, called sub-processes. For example, according to ISO/IEC 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 modelling, documentation, evaluation and management. Activities can also be decomposed into sub-activities, as shown in Figure 7.

Processes can be composed by other processes, called sub-processes, and activities. A software process can interact with other processes. This interaction can occur in several ways, among them: a process can precede the execution of another, two processes can be executed in parallel, or a process can be executed in a specific moment during the execution of another process.

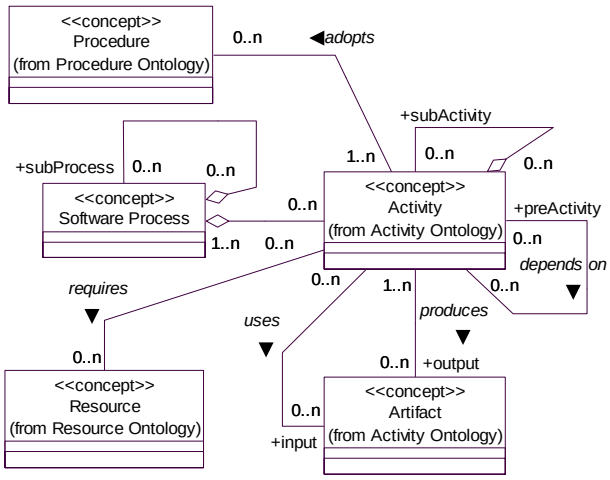
Figure 7 Process decomposition and interaction (see online version for colours)



5.2 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 artefacts, required resources (humans, software and hardware) and the procedures (methods, techniques, etc.) to be followed when performing the activity. Figure 8 presents the process assets involved in software process definition.

Figure 8 Process definition (see online version for colours)



The major part of this model corresponds to the sub-ontologies presented in Section 4. Thus, we focus here in the issues concerning processes.

As discussed in Section 4.1, an activity is a piece of work that can produce artefacts. To be performed, an activity requires resources, adopts procedures and consumes artefacts. 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 *a*1, part of a software process *p*, requires as input an artefact *s*, and there is no other activity *a*2, part of the same process *p*, that produces this artefact, then *s* is said an input to *p*.

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

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

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

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.

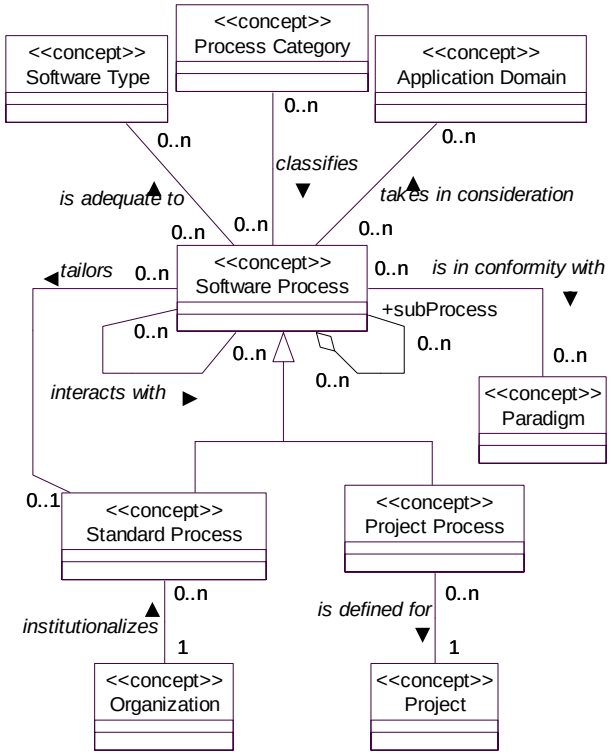
$$\forall (p1, p2, s)\ subProcess(p2, p1) \wedge output(s, p2) \wedge ((\neg\exists\ p3)\ subProcess(p3, p1) \wedge input(s, p3)) \rightarrow input(s, p1)$$

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

5.3 Process type and abstraction level

As shown in Figure 9, processes can be classified in process categories. For example, if an organisation follows the ISO/IEC 12207 classification, the categories could be primary processes, supporting processes, and organisational processes. Furthermore, processes are in different levels of abstraction. A standard process refers to a generic process institutionalised in an organisation, establishing basic requirements for processes to be performed in that organisation. A project process refers to the process defined for a specific project, considering the particularities of that project.

Figure 9 Process types and abstraction level (see online version for colours)



Software processes (standard or project processes) can be defined tailoring as a standard process. When a standard process tailors another standard process, the tailored process

is called a specialised process, and every process assets defined in the standard process become part of the specialised process. However new assets can also be included to deal with features of a specific software type, paradigm or application domain.

$$\forall(p1, p2)\ (tailors\ (p2, p1) \wedge standardProcess\ (p2)) \rightarrow specialisedStandardProcess\ (p2)$$

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 out that the decomposition and interaction between processes must occur at the same level of abstraction; i.e., a standard process can be composed only by other standard process and a project process can be composed only by other project process. Analogously, a standard process can interact only with other standard process and a project process can interact only with other project process.

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

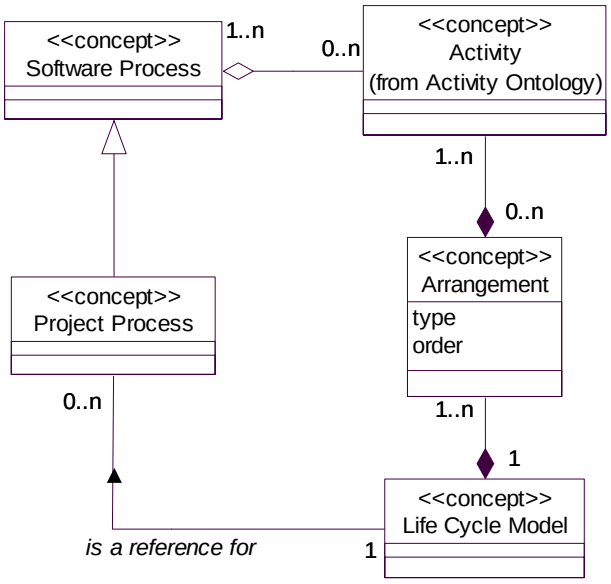
$$\forall\ (p1, p2)\ (interactsWith\ (p1, p2) \rightarrow (standardProcess\ (p1) \wedge standardProcess\ (p2)) \vee (projectProcess\ (p1) \wedge projectProcess\ (p2))$$

5.4 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 organising 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) (Pressman, 2005), for instance, can be described as a single sequential arrangement of all phases. The spiral model (Pressman, 2005) 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 (Pressman, 2005) 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 10.

Figure 10 Project process life cycle model (see online version for colours)



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 *lcm*, then each activity *a* that is part of an arrangement *c* of the life cycle model *lcm* must also be part of *p*.

$$\forall\ (p, lcm, c, a, n)\ (isReferenceFor\ (lcm, p) \wedge partOfLifeCycleModel\ (c, lcm) \wedge partOfArrangement\ (a, c) \rightarrow partOfProcess\ (a, p))$$

6 Mapping standards to the ontology

Once the software process ontology is defined, we mapped the structure of the standards into the concepts of the ontology. Although some standards, such as ISO 9001:2000 (ISO, 2000) and CMMI, are not software specific, our mapping focuses on software organisations, and thus we refer only to the aspects related to software processes.

Table 1 ISO × software process ontology

ISO	*Software process ontology*
Process	Software process
Standard process	Standard process
Tailored process	Project process
Work product	Artefact
Activity/task	Activity
Process category	Process category
Process	Software process
Life cycle model	Life cycle model

Table 1 shows a basic 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 2 shows a basic mapping between the vocabulary used by CMMI and the concepts of the ontology presented in this paper.

Table 2 CMMI × software process ontology

CMMI	*Software process ontology*
Process	Software process
Standard process	Standard process
Defined process/project's defined process	Project process
Work product	Artefact
Practice	Activity
Process area	Software process
Project	Project
Life cycle model	Life cycle model

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

Table 3 RUP × software process ontology

RUP	*Software process ontology*
Process/workflow	Software process
Process framework	Standard process
Worker	Human resource
Artefact	Artefact
Activity/step	Activity
Guideline/tool mentor/template	Procedure

7 Related work

There are several works exploring the mapping between standards. Mustafelija and Stromberg (2003), 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) (OMG, 2005), which can be used to describe a concrete software development process or a family of related software development processes.

SPEM follows an object-oriented approach for modelling a family of related software processes. It follows the four-layered architecture of modelling as defined by the OMG. A performing process (a project process in our ontology) is at level M0. The definition of the corresponding process is at level M1. It can be considered analogous to our standard process, although it is not exactly the same, since both a generic process, like RUP, and a specific customisation of this process used by a given project are at level M1. The process metamodel stands at level M2 and serves as a template for level M1. This process metamodel is comparable with our ontology. The SPEM specification is structured as a UML profile, and provides a complete MOF-based metamodel. The meta object facility (MOF) stands at level M3 and it is comparable to our meta-ontology.

It is worthwhile to point out that, although we use an UML profile as a modelling language for expressing ontologies, we do not follow an approach like SPEM. In our case, we defined a UML profile using stereotypes to capture our meta-ontology, which includes concepts such as concept, relation, property and so on (Mian and Falbo, 2003) (see Figure 2). We are not using the UML meta-model as basis for defining our software process ontology, as SPEM does. Concerning this, we should stress that we apply a formal semantics in our meta-ontology in contrast to the informal definitions of the MOF.

Like our ontological approach, SPEM intends to define the minimal set of process modelling 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*, which perform operations (called *activities*) on concrete, tangible entities called *work products*.

Abstract concepts, such as model element, package, work definition and process performer, are used as basis for the definition of concrete classes in SPEM. Looking for these concrete classes, 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.

As proof of concept, it is said that the (meta) model and UML profile defined in SPEM supports at least some well known processes, such as RUP. Our focus is a little bit different. We focused on more general standards, such as ISO/IEC 12207 and CMMI.

Table 4 SPEM × software process ontology

SPEM	*Software process ontology*
Process role	Human resource
Work product	Artefact
Activity/step	Activity
Guidance	Procedure
Categorises dependency	Process category
Process	Software process
Life cycle	Life cycle model

8 Conclusions and future work

Nowadays, software process improvement is being considered essential for software organisations to survive in a competitive market, and 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. However 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 conceptualisation 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 organisations to better understand the commonalities and differences between the various standards.

Finally, we used this ontology in the development of a process infrastructure for Ontology based software Development Environment (ODE) (Falbo et al., 2005), a process-centred software engineering environment that is developed based on ontologies. ODE is developed following a systematic approach for deriving object models from ontologies (Falbo et al., 2002), and the ontology presented in this paper was used to derive the core classes of process control in ODE, supporting tool integration and interoperability in it. This environment is being used by different organisations in practice, corroborating that the ontology can be applied to practical software projects.

As a future work, we intend to do a more complete mapping between our ontology and some standards, especially CMMI, in order to use it to allow ODE to be configurable for using the most adequate vocabulary, given by the choice of a standard that a software organisation commits to.

We are also planning to investigate generic process ontologies, in order to check the degree of conformity of our ontology with them, and evaluate the possibility of reusing previous conceptualisations, especially concerning process interaction.

Acknowledgements

This work was accomplished with the support of CNPq, an entity of the Brazilian Government reverted to scientific and technological development. It is also supported by FAPES, a foundation supporting science and technology from the government of the state of Espírito Santo – Brazil.

References

Chandrasekaran, B., Josephson, J.R. and Benjamins, V.R. (1999) 'What are ontologies, and why do we need them?', *IEEE Intelligent Systems*, January, Vol. 14, No. 1, pp.20–26.

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

Falbo, R.A. (2004) 'Experiences in using a method for building domain ontologies', *Proc. of the 16th International Conference on Software Engineering and Knowledge Engineering (SEKE´2004)*, International Workshop on Ontology In Action, Banff, Canada, pp.474–477.

Falbo, R.A., Guizzardi, G. and Duarte, K.C. (2002) 'An ontological approach to domain engineering', *Proc. of the 14th International Conference on Software Engineering and Knowledge Engineering (SEKE´2002)*, Ischia, Italy, pp.351–358.

Falbo, R.A., Menezes, C.S. and Rocha, A.R.R. (1998) 'A systematic approach for building ontologies', *Proceedings of the 6th Ibero-American Conference on Artificial Intelligence*, *Lecture Notes in Computer Science*, Lisbon, Portugal, Springer-Verlag Berlin Heidelberg, Vol. 1484, pp.349–360.

Falbo, R.A., Ruy, F.B. and Dal Moro, R. (2005) 'Using ontologies to add semantics to a software engineering environment', *Proc. of the 17th International Conference on Software Engineering and Knowledge Engineering – SEKE´2005*, Taipei, China, July, pp.151–156.

Fox, M.S. and Gruninger, M. (1994) 'Ontologies for enterprise integration', *Proceedings of the 2nd Conference on Cooperative Information Systems*, Toronto, Ontario.

Fuggetta, A. (2000) 'Software process: a roadmap', *Proceedings of the Conference on the Future of Software Engineering* (Limerick, Ireland, 04–11 June 2000), ICSE '00, ACM Press, New York, NY, pp.25–34.

Gruber, T.R. (1995) 'Toward principles for the design of ontologies used for knowledge sharing', *International Journal of Human-Computer Studies*, December, Vol. 43, pp.5–6, pp.907–928.

Gruninger, M. and Fox, M. (1995) 'Methodology for the design and evaluation of ontologies', *Proceedings of the Workshop on Basic Ontological Issues in Knowledge Sharing – IJCAI´1995*, Montreal, Canada.

Guarino, N. (1998) 'Formal ontology and information systems', in N. Guarino (Ed.): *Formal Ontologies in Information Systems*, IOS Press, pp.3–15.

Humphrey, W.S. (1990) *Managing the Software Process*, Addison-Wesley Publishing, Company, Massachusetts, USA.

ISO 9001 (2000) *Quality Management Systems – Requirements*.

ISO/IEC 12207 (1995), Amd 1 (2002), Amd 2 (2004), *Information Technology – Software Life Cycle Processes.*

ISO/IEC 15504 (2003) *Information Technology – Process Assessment.*

Kruchten, P. (1998) *The Rational Unified Process: An Introduction*, Addison Wesley.

Mian, P.G. and Falbo, R.A. (2003) 'Supporting ontology development with ODEd', *Journal of the Brazilian Computer Science*, November, Vol. 9, No. 2, pp.57–76.

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

OMG (2005) *Software Process Engineering Metamodel Specification*, Version 1.1, January.

Paulk, M.C., Weber, C.V., Garcia, S.M., Chrissis, M.B. and Bush, M. (1993) *Key Practices of the Capability Maturity Model*, Version 1.1, Technical Report CMU/SEI-93-TR-025.

Pressman, R.S. (2005) *Software Engineering: A Practitioner's Approach*, 6th ed., McGraw Hill.