A Configuration Management task ontology for semantic integration

Article · March 2012

DOI: 10.1145/2245276.2245344

2 authors:

Rodrigo Calhau
Instituto Federal de Educação, Ciência e Tecnologia do Espírito Santo…
13 PUBLICATIONS **35** CITATIONS
SEE PROFILE

Ricardo de Almeida Falbo
Universidade Federal do Espírito Santo
172 PUBLICATIONS **1,661** CITATIONS
SEE PROFILE

A Configuration Management Task Ontology for Semantic Integration

Rodrigo Fernandes Calhau, Ricardo de Almeida Falbo
Ontology and Conceptual Modeling Research Group (NEMO), Computer Science Department
Federal University of Espírito Santo, Vitória, Brazil
+55-27-4009-2167
{rfcalhau, falbo}@inf.ufes.br

ABSTRACT

Configuration Management (CM) is an important task for developing complex products. It is a complex task and there are many CM systems that aim to support it. However, generally, these systems work in isolation and there is a need for integrating them. In this context, ontologies have an important role, acting as an inter-lingua to help achieving a shared conceptualization that allows semantic integration. This paper presents an ontology of the CM task. This ontology was built with the purpose of supporting semantic integration of CM systems, mainly in service and process layers of integration.

Categories and Subject Descriptors

D.2.12 [**Interoperability**]

General Terms

Management, Languages.

Keywords

Task Ontology; Configuration Management; Semantic Integration.

1. INTRODUCTION

Configuration Management (CM) is a fundamental task for developing complex products. It is a management activity that provides technical and administrative guidelines for the lifecycle of a product, and its configuration items (CIs). CM drives and controls the evolution of the product configuration and provides information to prevent disorder in its development [1]. This control occurs primarily through the process of identifying and defining the product's CIs, controlling changes on them throughout the product lifecycle, recording and reporting the status of the CIs and the change requests, and verifying the completeness and correctness of the items [2, 1].

CM is a complex task and needs to be supported by tools. There are many systems that can be used to support the CM process, such as version control systems, change control systems and issue tracking systems. However, these systems typically support only part of the CM process, and more than one tool has to be used to support the whole CM process [3]. Moreover, these systems usually work in isolation, resulting in rework and inconsistency. If they work together, they could support the CM process more effectively. However, for integrating them, the systems need to share a common conceptualization.

In respect to integration dimensions, Izza [4] points out, among others, three layers: *data integration* deals with how the applications share data; *service integration* addresses how applications share services; finally, *process integration* views the enterprise as a set of interrelated processes and it is responsible for handling message flows, and defining the overall process execution. Thus, semantic integration encompasses the intended meaning of concepts, services and processes [4]. Thus, to achieve semantic integration, it is essential that the parties share a common understanding regarding the CM universe of discourse, including both its concepts and tasks. However, in general, there is a lack of understanding even regarding the key concepts related to CM, and as a consequence, there is not a consensus of the semantics of CM terms yet. This may hinder the achievement of semantic integration and communication between systems supporting the CM process.

In this context, ontologies can be used as an inter-lingua to map concepts and services used by different tools, in a scenario of access to data and services via a shared ontology [5]. An ontology is a formal representation of a common conceptualization of a universe of discussion [6]. Several authors consider that ontologies are at the heart of the modern approaches for semantic integration [4]. Ontologies can focus on describing the concepts of a domain (domain ontologies) or describing general tasks (task ontologies) that are independent of domain. A large amount of domain ontologies have been used in various fields [7], including software CM [8]. Despite the increasing use of domain ontologies, the same does not occur with task ontologies [9]. However, in semantic integration, we should also consider behavioral knowledge. In service and process layers of integration, task ontologies can be used to assign semantics to services, functionalities, activities, and its related information, helping achieving semantic integration.

This paper presents a task ontology for the CM process. This task ontology is used as a reference model for semantically integrating CM systems in service and process layers. The ontology focuses mainly on change control, which is a core process for CM. The proposed ontology aims to capture the most important information concerning the CM process, such as concepts, relationships, tasks, agents, inputs and outputs.

This paper is organized as follows. Section 2 regards the theoretical background of the paper, discussing briefly the CM process, the integration problem and the use of ontologies to deal with this problem. Section 3 presents the CM task ontology. Section 4 discusses its use in an integration scenario of CM systems. Section 5 briefly discusses some related works. Finally, Section 6 presents the conclusions of this paper.

2. SEMANTIC INTEGRATION AND CONFIGURATION MANAGEMENT

Configuration management (CM) applies technical and administrative procedures for developing, producing and supporting the life cycle of a product. This discipline is applicable to hardware, software, processed materials, services, and related technical documentation. Its main goal is to control product evolution [1].

The CM process involves activities for: (i) identifying and documenting characteristics of a product; (ii) controlling changes; (iii) storing and reporting information related to processing changes and; (iv) verifying the compatibility of the changes with the specified requirements [10].

To facilitate controlling its evolution, a product is divided into items. An item is a generic term used to represent parts of the product or information generated in its development**.** A product item which configuration is being managed is called a *Configuration Item* (CI). Changes in CIs occur through formal procedures [1].

A CI presents different states as it evolves. A version represents a specific state of a CI in a particular time point of the product development [3]. The product as a whole can also have different states called the product configuration. Configuration is usually defined as the set of items that form the product. A baseline is a product configuration that was revised and designated to be a basis for future development [1, 10].

Regarding the CM process, it is presented in different ways in different books and standards. Based mainly on [1, 2, 3, 10], the main activities of the CM process are:

- *Configuration Identification*: refers to identifying product items to be controlled (CIs), defining criteria for selecting CIs and their versions, establishing standards for numbering, and defining tools and techniques to be used to control the items;
- *Version control*: combines procedures and tools in order to manage different versions of the CIs;
- *Change Control*: deals with change management during the product life cycle. The change control process includes activities for: (i) requesting changes, (ii) evaluating the change request, (iii) performing checkout of the CIs to be changed, (iv) performing the change itself, (v) performing the check-in of the modified items, and (vi) verifying the changes made.

Besides these activities, the CM process has also activities that involve CM planning, Configuration Audit and Configuration Status Report [1, 2, 3, 10].

Managing the configuration of a product is an important and complex task, and to be properly done, it should be supported by a set of systems. The integration of these systems is a hard problem. The main difficulty is that generally the systems are not developed thinking in integration. Contrariwise, they generally have their own data (structural) and process (behavioral) models. This heterogeneity is pointed as one of the biggest problems in system integration. To solve this problem, it is necessary to resolve syntactic (related to structure) and semantics (related to meaning) conflicts generated by this heterogeneity [4].

Semantic integration involves three main integration layers [4]: data layer (refers to data exchange), service layer (deals with service exchange) and process layer (responsible for combining the systems for an adequate support to the process). To help semantic integration, ontologies can be used to establish a common conceptualization, explaining concepts and their meanings, and avoiding conflicts of understanding. According to [11], an ontology is a conceptual specification that describes the knowledge of a universe of discourse. It defines a specific vocabulary used to describe a certain reality and a set of explicit decisions to establish accurately the intended meaning to this vocabulary [6].

Guarino classifies ontologies into [6]: (i) *foundational ontologies* (or top-level ontologies), which describe very general concepts, such as space, time, object, event, action etc., (ii) *domain ontologies*, which describe the conceptualization related to a generic domain (for instance, medicine, law, and so on), (iii) *task ontologies*, which describe the conceptualization related to a generic task (such as diagnosis and sale), and (iv) *application ontologies* that describe concepts dependent on a particular domain and task. Domain ontologies have been widely used in various areas of computer science, but the same does not occur with task ontologies [9].

Task knowledge involves two major kinds of knowledge that should be captured by a task ontology [9]: (i) task decomposition, including control flow, and (ii) knowledge roles played by entities from the domain in the fulfillment of the task. These kinds of knowledge are very inter-related, although they capture different views of a task. In fact, they represent different modeling aspects, i.e. different dimension of modeling that emphasizes particular views of the same portion of the reality. Thus, we need different models for representing them [9]. Martins and Falbo [9] proposed the use of two UML diagrams for representing task ontologies: activity diagrams, capturing task decomposition into sub-tasks and how knowledge roles act in their fulfillment, and class diagrams, modeling the knowledge roles involved and their properties and relations.

In the next section, we present a task ontology that describes aspects of these both perspectives regarding the CM process. It is worthwhile to point out that, although we use the term "task ontology", which is already consecrated in the field of ontologies, in fact we are talking about a process ontology, in the sense that we are interested in describing the CM process as a whole, and not tasks with low granularity level. Moreover, albeit our ontology focus on the CM's change control sub-process, we decided to name it a CM task ontology, because we also consider some activities and concepts that are part of the version control sub-process. However, due to space limitations, in this paper we present only the core of our ontology.

Finally, we should emphasize that our approach to semantic integration is focused on the conceptual level, as advocated in [12]. Thus, our ontology is a reference ontology, i.e., an ontology that is constructed with the sole objective of making the best possible description of the universe of discourse, with regard to a certain level of granularity and viewpoint [11]. A reference ontology is a special kind of conceptual model representing a model of consensus within a community. It is a solution-independent specification with the aim of making a clear and precise description of entities in the universe of discourse, for the purposes of communication, learning and problem-solving. We are not interested in an implementation of this ontology for purposes of reasoning, for instance.

3. A CM TASK ONTOLOGY

As a process ontology, we are interested in answering the following competency questions with our CM Task Ontology (CMTO): (i) Which are the activities of the CM process? (ii) Who is responsible for performing these activities? (iii) How these activities are decomposed into sub-activities? (iv) What is the control flow between them? (v) What are the inputs and outputs of each activity?

Following the guidelines given in [9], for capturing the conceptualization involved in the CM process, we developed two conceptual models. The first is a structural conceptual model capturing the knowledge roles involved in the CM process. The second is a behavioral model capturing the activities of the CM process and related aspects.

Figure 1 shows the main knowledge roles involved in the CM process. This model is built in OntoUML, a UML profile that captures some distinctions done in the Unified Foundational Ontology – Part A (UFO-A) [7]. By using this notation, we are showing the use of UFO as a foundational ontology for grounding our CMTO, as advocated by Guarino [6] and Guizzardi and colleagues [13].

Item represents the elements that compose a product (or even the product itself). Item is a *category* in UFO, since it represents different kinds of elements. When an item is submitted to configuration management, it is said to be a **Configuration Item (CI)**. Thus, CI is a role (more precisely a *role mixin* in UFO) played by an item due to the fact that it has being submitted for CM, during a "Identify Configuration" activity (not shown in Figure 1). A CI can be composed of other CIs (**Composite CI**) or not (**Atomic CI**). A CI is characterized by **Versions**. Each version represents a specific state in the evolution of the CI. Version is represented as a property (*mode*, in UFO) of a CI, since it is intrinsically dependent on it. **Configuration** is a specific type of version that is composed of other versions. Some configurations play the role of **Baseline**. A baseline is generated when a configuration manager labels a configuration as a baseline in a "Define Baseline" activity (not shown in Figure 1). Finally, concerning the relations shown in Figure 1, the parthood relationships between Composite CI and CI, and between Configuration and Version are both of the type *component of* in UFO. In both cases, the parts are shareable, since a version can be part of different configurations, as well as a CI can be part of different composite CIs. The relationship between CI and Version, and the ones specialized for their subtypes, are *characterization* relations, occurring between *modes* (versions) and the objects (CIs) they characterize.

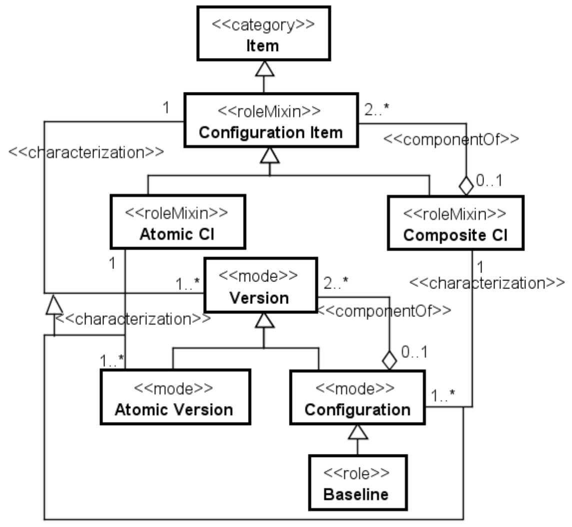

Figure 1. Main knowledge roles involves in the CM Process.

Figure 2 shows the knowledge roles involved in the Change Control sub-process. This model mainly captures the registering of the activities that occur in the change control process. For instance, the change control process starts with a **requester** requesting changes in a set of **versions**. This activity gives rise to an entity **Change Request** that registers the occurrence of this activity, including the point in time (date and time) when it occurred (not shown in Figure 2).

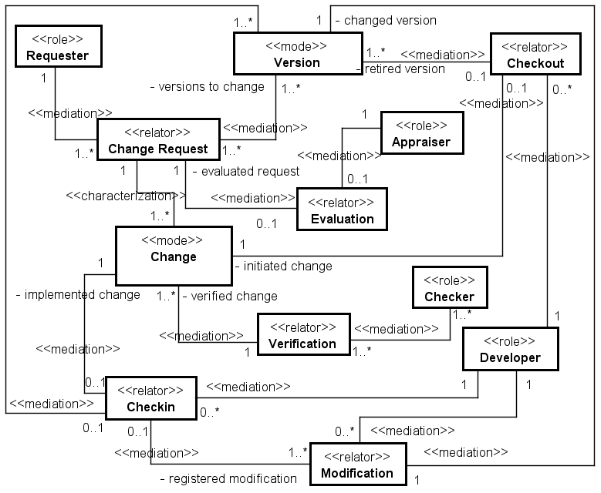

Figure 2. Knowledge roles involves in Change Control.

In UFO, the registering of events is done by means of *relators*. Relators are entities with the power of connecting other entities [7]. In the previous example, **Change Request** is a *relator* that connects a **Requester** with the **Versions** she/he thinks that need to be changed (**versions to change**). *Relators* are the foundation for *material relations*, such as the relation "requires change in" between **Requester** and **Version**. In other words, *material relations* have material structure on their own. The relata of a *material relation* are mediated by *relators*. Thus, the relationships between a *relator* (e.g., **Change Request**) and the entities that it connects (e.g., **Requester** and **Version**) are *mediation relations* [7]. As pointed in [7], a mediation is a formal relation that takes place between a relator and the entities it mediates.

Figure 3 presents the activity diagram used to show the activities of the Change Control process and how the knowledge roles shown in figures 1 and 2 act in this process. Some stereotypes were added to capture distinctions made in UFO (Part C) [13] concerning to the types of object participations in actions, namely: creation, indicating that an object is created by the action; termination, indicating that the object is destroyed by the action; change, indicating that some property of the object changed; and usage, when the object is used without changing any of its properties.

As said before, the Change Control process starts with a **Requester** requesting changes in a set of **Versions** (**versions to change**). A **Change Request** describes a set of **Changes** that are supposed to be made. Thus, a **Change** is a *mode* (in UFO) of the **Change Request**, and thus the relationship between them is a *characterization relation* (see Figure 2). Both the change request and its corresponding changes are created as a result of the "Request Change" activity.

Once a change request is made, it should be evaluated by an **Appraiser** ("Evaluate Request" activity). The *relator* **Evaluation** registers the occurrence of this activity, when the change request and its changes are said to be **evaluated**. If the request is rejected, the process finishes; otherwise, it follows to the "Implement Change" activity.

The "Implement Change" activity is a complex activity that involves three sub-activities, as shown in Figure 4. All these sub-activities are performed by **Developers**.

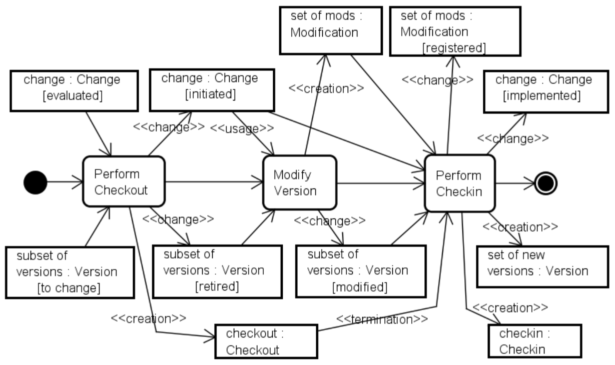
set of mods : Modification
set of mods : Modification [registered]
change : Change [evaluated]
change : Change [initiated]
<<creation>>
<<change>>
change : Change [implemented]
<<change>>
<<usage>>
<<change>>
Perform Checkout
Modify Version
Perform Checkin
<<change>>
<<change>>
<<creation>>
subset of versions : Version [to change]
subset of versions : Version [retired]
subset of versions : Version [modified]
set of new versions : Version
<<creation>>
<<termination>>
<<creation>>
checkout : Checkout
checkin : Checkin

Figure 4. Sub-activities of the "Implement Change" activity.

During a change implementation, first the **Developer** performs a checkout, retiring the set of **versions to change**. These versions are then considered **retired**. A checkout is performed to implement an **evaluated change** that is **initiated**. As shown in Figure 2, **Checkout** is a *relator* connecting three types of entities: **Developer** (who performs the checkout), **Change** (which is **initiated**), and **Version** (indicating the **retired versions**).

The next activity is "Modify Version". In this activity, a **Developer** makes modifications in each of the **retired versions**. A **Modification** registers the occurrence of modifications in each **Version**, which is considered a **modified version**. Thus, Modification is a *relator* connecting **Developer** (who performs the modification) and **Version** (what is **modified**) (see Fig. 2).

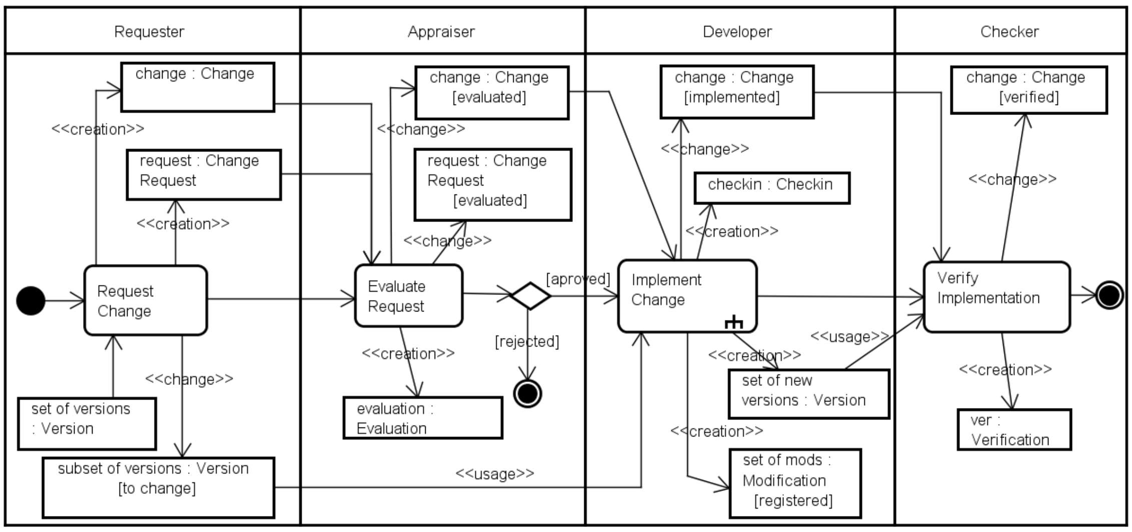

Figure 3. Behavioral Conceptual Model of the Control Change Process.

Since all the modifications have been made and the change is completely addressed, the **Developer** should perform a checkin, creating new versions based on the modified ones. **Checkin** is the *relator* that registers the occurrence of this activity. As shown in Figure 2, it connects four types of entities: **Developer** (who performs the checkin), **Change** (which has been **implemented**), Modification (indicating which modifications were **registered**) and **Version** (corresponding to the new versions created). Moreover, when a checkin occurs, the corresponding checkout is destroyed.

Finally, after implementing a change, it must be verified. In the "Verify Implementation" activity (Fig. 3), a **Checker** verifies if the change was properly implemented. As a result, the change is **verified** and the *relator* **Verification** is created, connecting the **Checker** who performed the verification, and the **Change** that was **verified**.

It is worthwhile to highlight that the structural conceptual models of a task ontology are incomplete by their nature. They do not represent the concrete entities that actually play the knowledge roles shown in the model. As pointed in [7], anti-rigid entities should be subtypes of rigid types. Since *roles* (in UFO) are anti-rigid entities, the entities in CMTO that are stereotyped with <<role>> (namely **Requester**, **Evaluator**, **Developer** and **Checker**) must be subtypes of other entities that, in turn, are rigid entities. For instance, if in some domain the *role* **Requester** is played by human agents, then **Requester** must be a subtype of Person (a rigid entity). However, this complement can only be done when integrating a task ontology with a domain ontology, giving rise to an ontology of a class of applications, when we should identify which role a domain concept should play in the structural model of the task ontology [9].

4. USING THE CM TASK ONTOLOGY IN A SEMANTIC INTEGRATION EFFORT

The proposed ontology was used in a semantic integration effort aiming at integrating Subversion (SVN) [14] and CM-ODE. The first is a well known open-source version control system that manages files and folders, and the changes made on them over the time. The second is a tool supporting change management in ODE (Ontology-based software Development Environment) [15].

The integration scenario aims to provide automated support for part of the Software Configuration Management (SCM) process defined at NEMO (Ontology and Conceptual Modeling Research Group). More specifically, the following activities were considered: Identify Configuration, Control Configuration, and Control Change. The Control Change activity is decomposed in the following sub-activities: Request Change, Evaluate Change and Perform Change.

Since CM-ODE was developed at NEMO, its conceptual models were already available. The conceptual models of SVN, on the other hand, had to be excavated. Concerning SVN behavior, we extract the provided services by the SVNKit (a Java Subversion Library).

In [12], this integration scenario was already addressed, exploring the use of the SCM domain ontology (SCMDO) proposed in [8] to support integration, using OBA-SI (Ontology-Based Approach for Semantic Integration). In this work, we extend the approach adopted by using the task ontology as a reference model for integration in service and process layers. To do that, first we had to align and integrate our ontology (CMTO) to SCMDO, producing an ontology of the class of applications related to SCM (said application ontology for short). This was an important step for resolving some open issues in the CMTO, regarding roles and categories.

As discussed above, CMTO does not establish who plays the roles of **Requester**, **Evaluator**, **Developer** and **Checker**. According to SCMDO, Person (*kind* in UFO) plays these roles. Thus, in the resulting application ontology, **Requester**, **Evaluator**, **Developer** and **Checker** are subtypes of **Person**. Moreover, the CMTO says nothing about which types of entities can have their configuration managed. In SCMDO, Artifacts (*category* in UFO) and software tools (*kind* in UFO) are those items. Thus, we introduced **Artifact** and **Software Tool** as subtypes of **Item**. We also introduced some kinds of artifacts as subtypes of **Artifact**, namely **Source Code**, **Document** and **Diagram**. Other concepts present in the SCMDO were also aligned to the CMTO concepts, but due to space limitations, this alignment is not discussed here.

Using the resulting application ontology, we revised the structural mappings done in [12]. For instance, Configuration Item (CM-ODE) and Repository Item (SVN) were mapped to **Configuration Item** (Ontology). File and Folder in SVN are types of Repository Item. They were mapped, respectively, to **Atomic CI** and **Composite CI** (Ontology). Besides mapping the concepts of SVN and CM-ODE to concepts of the application ontology, we did the same with the relationships.

After mapping the structural models, we performed the mappings of the behavioral conceptual models of the process to be supported, and the systems to be integrated. Table 1 illustrates the mappings between the model elements of CMTO, NEMO's SCM process, SVN and CM-ODE, regarding the "Perform Checkout" activity. Activities / services are shown in white, whereas inputs and outputs are shown in light and dark gray, respectively. The mapping of inputs and outputs were based on the structural mappings previously performed.

Table 1. Behavioral Mappings: Perform Checkout

Ontology	SCM Process	SVN	CM-ODE
Perform Checkout	Retire CIs	checkout	retire_change
change: Change [evaluated]			ch: Change
subset versions: Version [to_change]	current versions of CIs	dir: Repository Folder	--
change: Change [initiated]			ch: Change [retired]
subset versions: Version [retired]	current versions of CIs [retired]	copies: Working Copy Item	--
checkout: Checkout		checkout: Checkout	ret: Retirement

Looking for these mappings, we can notice that both the services **checkout** (SVN) and **retire_change** (CM-ODE) can be used to support the "Retire CIs" activity from SCM process. Moreover, these mappings show how to map inputs and outputs for invoking the services. These mappings guided the choice of which tools to use in each circumstance, and how to integrate them.

5. RELATED WORK

Our first intended use of the CM task ontology was to assist in the semantic integration of CM systems. As pointed in [4], ontologies are at the heart of the modern approaches for semantic integration. However, at the best of our knowledge, the existing approaches to semantic integration use only domain ontologies. This is the case, for instance, of ONAR, an ontology-based framework for Enterprise Application Integration [16]. Moreover, most of the existing approaches are too much focused on the technological aspects of a semantic integration solution, mainly working at extensions of the web service technology, such as ODSOI (Ontology-Driven Service-Oriented Integration) [4], or using semantic web technologies in order to enrich the semantics of the exchanged information, such as [16].

6. CONCLUSIONS

More and more, task ontologies are receiving attention. However, differently of domain ontologies, which have several works using them, the use of task ontologies is still timid. In this paper, we presented a task ontology that aims to capture the conceptualization involved in the Configuration Management process. It focuses mainly on the Change Control sub-process.

The primary purpose of the proposed CM task ontology is to make the best possible description of the CM process, elaborating a model of consensus within this community. It is a solution-independent specification with the aim of making a clear and precise description of entities in this universe of discourse, for the purposes of communication, learning and problem-solving. To achieve this goal, we used as sources of knowledge international standards, such as ISO 10007 [1], books and handbooks devoted to the subject, and some CM systems, such as SVN [14]. Moreover, experts in this universe of discourse evaluated the resulting ontology. We looked also for ontologies describing the CM universe of discourse, but we only found the Software CM domain ontology presented in [8].

We believe that semantic integration has to be first addressed at the conceptual level, as advocated by OBA-SI [12]. Furthermore, we share Izza's point of view that integration should consider data, service and process layers. In this sense, the use of task ontologies for semantic integration is a step ahead in the direction towards service and process integration at the conceptual level.

As future work, we intend to extend OBA-SI for considering also task ontologies as reference models for integration in service and process layer. Moreover, we also intend to extend the coverage of our CM task ontology, by considering other activities of the CM process.

7. ACKNOWLEDGMENTS

This research is funded by the Brazilian Research Funding Agencies FAPES (Process Numbers 45444080/09 and 52272362/11) and CNPq (Process Numbers 481906/2009-6 and 483383/2010-4).

8. REFERENCES

[1] International Organization for Standardization. *ISO 10007: Quality Management - Guidelines for Configuration Management*, 2003.

[2] Department of Defense of United States of America. *MIL-HDBK-61A: Military Handbook - Configuration Management Guidance*, 1997.

[3] Leon, A. *A Guide to Software Configuration Management*. Artech House Publishers, Norwood, MA, 2000.

[4] Izza, S. Integration of industrial information systems: from syntactic to semantic integration approaches. *Enterp. Inf. Syst.* 3, 1 (February 2009), 1-57.

[5] Jasper, R., Uschold, M. A framework for understanding and classifying ontology applications. *Proceedings of the IJCAI-99, Ontology Workshop* (Stockholm, 1999).

[6] Guarino, N. Formal Ontology and Information Systems. In: *Formal Ontologies in Information Systems*, IOS Press, 1998, 3 -15.

[7] Guizzardi, G. *Ontological Foundations for Structural Conceptual Models*, Universal Press, The Netherlands, 2005.

[8] Arantes, L.O., Falbo, R.A., Guizzardi, G. Evolving a Software Configuration Management, In *Proceedings of the 2nd Workshop on Ontologies and Metamodeling Software and Data Engineering* (Brazil, 2007).

[9] Martins, A.F., Falbo, R.A. Models for Representing Task Ontologies. In *Proceedings of the 3rd Workshop on Ontologies and Their Applications* (Brazil, 2008).

[10] Institute of Electrical and Electronics Engineers. *IEEE Std 610.12 - IEEE Standard Glossary of Software Engineering Terminology*, 1990.

[11] Guizzardi, G. On Ontology, ontologies, Conceptualizations, Modeling Languages, and (Meta)Models, In *Frontiers in Artificial Intelligence and Applications, Databases and Information Systems IV*, IOS Press, Amsterdam, 2007.

[12] Calhau, R.F., Falbo, R.A. An Ontology-based Approach for Semantic Integration, In *Proceedings of the 14th IEEE International Enterprise Distributed Object Computing Conference (EDOC'2010)* (Brazil 2010).

[13] Guizzardi, G., Falbo, R.A., Guizzardi, R.S.S. Grounding software domain ontologies in the Unified Foundational Ontology (UFO): the case of the ODE software process ontology. In *Proceedings of the XI Iberoamerican Workshop on Requirements Engineering and Software Environments* (Brazil, 2008), 244-251.

[14] Collins-Sussman, B., Fitzpatrick, B.W., Pilato, C.M. *Version Control with Subversion*. O'Reilly Media, 2009.

[15] Falbo, R.A., Ruy, F.B., Moro, R.D. Using Ontologies to Add Semantics to a Software Engineering Environment. In *Proceedings of the 17th SEKE* (China, 2005), 151-156.

[16] Tektonidis, D., Bokma, A., Oatley, G., Salampasis, M. ONAR - An Ontology-based Service Oriented Application Integration Framework, In *Proceedings of the 1st International Conference on Interoperability of Enterprise Software and Applications*, (Geneva, Switzerland, 2005), 65–74.