Models for Representing Task Ontologies

Aline Freitas Martins, Ricardo de Almeida Falbo

[1]Computer Science Department, Federal University of Espírito Santo – Brazil
alinefmart@yahoo.com.br, falbo@inf.ufes.br

Abstract. Knowledge is of general utility and should be captured thinking in reuse. A key idea underlining knowledge capturing for reuse is to consider that there are two major kinds of knowledge: domain and task knowledge. Ontologies can be used for representing both kinds of knowledge. However, while domain ontologies are broadly used and there are many proposals of models for representing them, the same does not occur for task ontologies. In this paper we propose the use of UML activity diagrams for capturing task control-flow, and UML class diagrams for capturing the knowledge roles involved in its activities. We also discuss the interrelationship between these two models and how they can be combined with domain ontologies in order to describe the knowledge involved in a class of applications.

1 Introduction

Nowadays, it is acknowledged that reuse offers an important opportunity to achieve improvements in software development. Software reuse can occur in several levels, from code to knowledge. Greater benefits, however, are achieved by reusing artifacts of higher abstraction level, i.e., knowledge reuse.

Concerning knowledge reuse, two major kinds of knowledge should be considered: domain and task knowledge. For developing knowledge for reuse, we need models to capture both, and ontologies can be used for this purpose. Domain ontologies describe the vocabulary related to a generic domain, like medicine. Task ontologies describe the vocabulary related to a generic task, like diagnosing or selling [1].

Domain ontologies have been extensively used in several areas in computer science, such as Domain Engineering, Artificial Intelligence and Semantic Web [2]. There are many works presenting ontologies for a multitude of domains, such as medicine, e-commerce, enterprise, law, chemistry and engineering [2]. Also a variety of models has been proposed for representing domain ontologies, most of them showing in some way the concepts, relations and properties that are relevant to capture the domain conceptualization through a structural conceptual model.

However, the same does not occur with task ontologies. There are few works presenting task ontologies, and there is not uniformity in representing them.

Task knowledge involves two different facets: task decomposition and knowledge roles involved in the fulfillment of the task. Task decomposition concerns dividing a task into sub-tasks, setting goals for each sub-task, and describing the control-flow among those sub-tasks [3]. The Knowledge roles facet is occupied with specifying concepts and relations appearing in the task of interest [4].

Considering these two facets of task knowledge, in this paper, we propose the use of two UML diagrams for representing task ontologies: activity diagrams, capturing task decomposition and how knowledge roles act in their fulfillment, and class diagrams, modeling the knowledge roles involved and their properties and relations. We also discuss their interrelationship and how they can be combined with domain ontologies in order to describe the knowledge involved in a class of applications.

This paper is organized as follows. Section 2 discusses some issues concerning reuse, task knowledge and ontologies. In section 3 we present our proposal for representing task ontologies, and finally, section 4 presents our conclusions.

2 Task Knowledge and Ontologies

According to Frakes and Kang [5], "software reuse is the use of existing software or software knowledge to construct new software". An approach for software reuse must consider two different perspectives: developing reusable assets (developing *for* reuse) and developing using those reusable assets (developing *with* reuse) [6]. Reusable assets can be either reusable software or software knowledge [5]. However, the major benefits of reuse are achieved when reusing knowledge about how to solve a recurring problem in a given universe of discourse. This can be done by developing abstractions in advance and then applying those abstractions to solve the problems when they are encountered [7].

Concerning knowledge reuse, two major kinds of knowledge should be considered: domain and task knowledge. As pointed out by Guarino [8], it is important to isolate the domain and task knowledge. Domain knowledge reusability across multiple tasks should be systematically pursued, as well as task knowledge reusability across several domains. Mizoguchi et al. [9] also emphasize this view of knowledge decomposition into a task-dependent but domain-independent portion and a task-independent but domain-dependent portion. Thus, attention should be deserved to domain analysis, conceived as an independent activity, as well as to task analysis.

Domain knowledge reuse has been extensively investigated by the communities of Artificial Intelligence (AI) and Domain Engineering, motivated by the need to reinforce software reuse in a higher level of abstraction. A large amount of domain ontologies has been developed for domains such as medicine, law, engineering, enterprise modeling and chemistry [2]. Also several methods for ontological engineering, mainly focusing on domain ontologies, have been proposed [10]. For representing domain ontologies, in general, some sort of structural model is used, such as UML class diagrams [2] [11] [12].

Task knowledge is generally associated to the description of a recursive decomposition of a top-level task in sub-tasks, the control-flow of these sub-tasks, and the description of the knowledge roles for the domain knowledge that is used or produced by the sub-tasks [3] [13]. Regarding task knowledge reuse, there are also several works focusing on it, most of them proposed by the AI community, such as CommonKADS [13].

Task ontologies have also been developed with the goal of reusing task knowledge, however, without the same intensity as domain ontologies. Moreover, in contrast to

domain ontologies, there are not widely accepted methods for engineering task ontology neither uniformity in representing them. For example, Mizoguchi et al. [14] use four kinds of concepts to describe a task ontology: (i) generic nouns representing objects reflecting their roles appearing in the problem solving process, (ii) generic verbs representing activities in the problem solving process, (iii) generic adjectives modifying the objects, and (iv) other concepts specific to the task. In [4], the authors include generic constraints and generic adverbs as part of the generic vocabulary used to describe tasks, and use a generic process network as a graphical model to represent the problem solving process in terms of the ontology primitives. Based on [4], Fang [15] uses a graphical model showing activities, inputs, outputs, controls and mechanisms for representing task ontologies. In line with these works, Wang and Chang [16] use UML activity diagrams to represent task ontologies.

On the other hand, Rajpathak and his colleagues propose ontologies for the tasks of scheduling [17] and planning [18] that do not describe flows of activities, but the concepts, relations and properties involved in those tasks. For instance, the scheduling ontology is described in terms of the following concepts: job, resource, constraints, schedule time range, preferences etc. For graphically presenting this ontology, they use a structural conceptual model, showing classes and relations, including sub-type and whole-part relations.

Conciliating in some way the two approaches discussed above, Zlot et al. [19] represent both task decomposition and knowledge roles in a task ontology. In what they call the conceptual level, task concepts (knowledge roles to be replaced by domain concepts) are modeled using a structural conceptual model, and the control-flow over the tasks are described by means of an algorithm written in structured natural language and a simple graphical notation to represent task decomposition.

As we can see, the development of task ontologies, in contrast to the development of domain ontologies, lacks maturity in several aspects. While in the development of domain ontologies, we have several methods, and to some extent an agreement on what is to be captured in a domain ontology (concepts, relations, properties and constraints), the same does not apply to the task ontology development. Thus, we need to go forward towards a systematic approach for developing task ontologies. In this journey, two steps are very important: defining what a task ontology should contain, and establishing models to capture these elements.

In our view, a task ontology should capture two intertwined perspectives: (i) task decomposition into sub-tasks (activities) and control-flow, and (ii) knowledge roles to be played by domain concepts in those sub-tasks. These two perspectives are complementary. For instance, responsibilities, and artifacts used and produced by activities should be both modeled by the control flow and the knowledge roles perspectives. In the first, the aim is to capture who is responsible by doing an activity, and which the artifacts used and produced by it are. In the last, we want to show structural relationships between those elements and other elements, for establishing the structure that the domain entities should have in order to play these knowledge roles. This is somewhat in line with the approach of Zlot et al. [19], but these authors do not elaborate on the relationships between the activities in the control flow view and the knowledge roles. Moreover, the models used by them to represent both the views are not diffused, and especially concerning the control-flow view, are very poor in expressivity, representing only tasks and sub-tasks. The use of more powerful

modeling languages, such as BPMN [20] or UML [21] (in this case, the elements used to represent activity diagrams), allows to better represent the knowledge task, especially the control-flow and data perspectives. This last perspective is very important to be captured by a behavioral model because it links behavioral and structural models. In our view, this is essential for well representing task ontologies. Thus, to overcome these barriers, in this paper we propose the use of two UML diagrams to represent task ontologies, namely: activity diagrams for capturing task control-flow and class diagrams for capturing the knowledge roles.

3 Representing Task Ontologies

A key factor for capturing knowledge is to have models representing it. A model is a simplification of something that we can visualize, manipulate and reason about it [22]. The use of graphical models is broadly recognized as essential for understanding, communication and reuse. Thus, we need graphical models for representing task knowledge by the means of task ontologies.

Looking for several works addressing task ontologies, as discussed in the previous section, we identify two major kinds of knowledge that should be captured by a task ontology: (i) task decomposition, including control-flow, and (ii) knowledge roles played by objects from the domain in the task fulfillment. These kinds of knowledge are very inter-related, although they capture different views of a task. In fact, they represent different model 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.

Task decomposition and control-flow comprise a behavioral view of a task. Problem-solving behavior encompasses a process and, thus, it can be described as a sequence of activities, changing the state of some defined objects, and performed by agents [23]. As a consequence, to represent task decomposition and control-flow, we need models that are able to represent activities (sub-tasks), agents, and objects/artifacts (inputs and outputs).

Knowledge roles, on the other hand, are a structural view of the task. They represent roles to be played by domain entities while performing a certain task [1]. Concepts involved in a task, such as agents and artifacts, are knowledge roles that domains entities can fulfill in the solution of a problem involving the task in a given domain [19]. In fact, knowledge roles link domain knowledge and task control-flow. In face of that, they should be clearly described and their relationships should be explicitly identified and modeled.

Thus, in a task ontology, we need both behavioral and structural models for representing these two dimensions of task knowledge. Since describing behavior involves referencing knowledge roles, we start discussing structural models for representing them. Before, however, we briefly present the lending task that is used as an example for illustrating the ideas proposed in this paper. So this section is organized as follows: subsection 3.1 talks about the lending task; section 3.2 discusses the use of structural models for representing the knowledge roles involved in a task, and their relations and properties; section 3.3 regards the use of behavioral models for

capturing task decomposition and control-flow; finally section 3.4 discusses briefly how a domain ontology can be merged with a task ontology in order to describe the knowledge involved in an application.

3.1 – The Lending Task

Lending is a task that is recurring in several domains. For instance, in a library, users borrow books; car rental companies rent cars to customers; realtors rent homes for renters and so on. In all cases, there are two main sub-tasks involved in a lending task: lend item and return item.

In spite of the domain in which it occurs, lending begins with the lessee choosing the kind of item that she/he wants to borrow. Then, it is necessary to check if there are available items of the selected kind. This availability checking is based on different constraints depending on the particular business. If there is more than one item available, the lessee can choose one of them to borrow. In order to formalize the lending, the lessor develops an agreement and the lessee reads it and verifies if she/he agrees with its conditions. If yes, some sort of record of the agreement is registered for controlling the lending and the item is delivered to the lessee.

In the devolution, the lessee gives the item back to the lessor, who verifies any violation of the agreement conditions. In the case of a condition violation, a penalty is applied in accordance with the established in the agreement and the lessee should pay off. Finally, the devolution is recorded and lending concluded.

3.2 – Modeling the Knowledge Roles involved in a task

As previously mentioned, since describing behavior involves referencing knowledge roles, we discuss first how to represent them.

In general, any structural model that allows representing concepts, relations, properties and some constraints, such as multiplicity constraints, can be used for representing knowledge roles. In order to not reinvent the wheel and further the integration with domain knowledge, we can look for the models typically used for constructing domain ontologies. Those should also apply for representing knowledge roles in task ontologies.

Among the structural models used for representing domain knowledge in domain ontologies, we can cite the UML's class diagram [11] and some extensions made on it, such as the one proposed by the SABiO's UML profile [12] [24], and OntoUML, a UML profile for the Unified Foundational Ontology [2]. For simplicity, in Figure 1 we use a UML's class diagram for representing the knowledge roles involved in the lending task. However, we are not advocating its use. Contrariwise, we are currently studying how to use OntoUML as the modeling language for both domain and task ontologies, but due to space limitations it is not possible to discuss such aspects here.

Figure 1 shows the main concepts and relations involved in the lending task. In a lending, a lessee borrows an item from a lessor. An item is classified by an item kind that defines availability constrains for items of such type. The availability of a particular item is governed by both the availability constrains of its kind and its own

specific availability constrains. Since the structural model is not able to capture these constrains, an axiom should be written: if an item kind *ik* defines an availability constraint *ac* and an item *i* is classified in *ik*, then *i* has also *ac* as an availability constraint: $(\forall i,ik,ac)\ (\ defines(ik,ac) \wedge classifies(ik,i)) \rightarrow has(i,ac)$.

Finally, a lending is governed by an agreement that has several clauses. Generally, organizations have templates of agreements that are used as basis to the elaboration of an agreement.

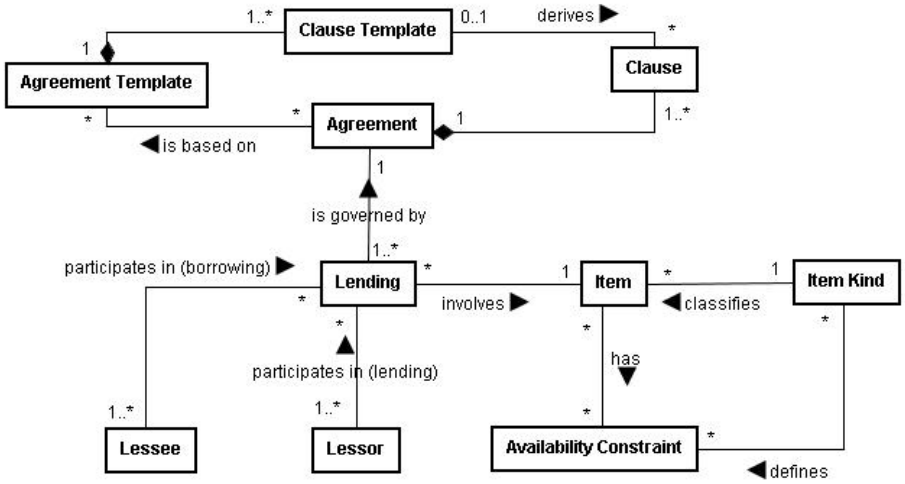

Fig. 1. Structural Model representing concepts and relations involved in the lending task.

It is important to emphasize that the concepts and relations shown in Figure 1 are generic. I.e., they are domain and application independent. In other words, lessee, lessor, item and other concepts presented in the model correspond to roles that can be played by entities of different domains in which the lending task can occur, maintaining their relations and functions. This model, therefore, describes one of the kinds of concepts cited by Mizoguchi et al. [14], namely generic nouns that reflect the object roles in the problem-solving process.

3.3 – Modeling task decomposition and control-flow

The model depicted in Figure 1 captures an important part of the knowledge involved in the lending task, that is its structural aspects. However, it does not capture the behavioral aspects of the lending task. Which are the sub-tasks involved in the lending task? What is their precedence order? When are the elements shown in Figure 1 used? All these questions remain unanswered. For representing this other part of the knowledge involved in a task, we need models that are able to represent processes, such as the ones proposed in the Business Process Modeling (BPM) area. Examples of such models include the BPMN[1] Business Process Diagram [20] and the UML's

[1] Business Process Modeling Notation [20]

Activity Diagram [21]. Both are standards widely diffused that provide mechanisms for representing dynamic aspects involved in task execution.

White [25] examined these two graphical process modeling diagrams in the light of 21 workflow patterns and concluded that both notations could adequately model most of the patterns. According to White, they provide similar solutions to most of the patterns, what indicates how close the notations are in their presentation. They inclusively share many of the same shapes for the same purposes.

Russel et al. [26] also examined the suitability of UML 2.0 activity diagrams for business process modeling. Their conclusions indicate that whilst activity diagrams have merit as a notation for business process modeling, they are not suitable for all aspects of this type of modeling. Although they offer comprehensive support for the control-flow and data perspectives, their suitability for modeling resource-related and organizational aspects of business processes is limited. Yet according to Russel et al., these limitations are shared with most other business process modeling formalisms and reflect the overwhelming emphasis that has been placed on the control-flow and data perspectives in process modeling notations.

Since in a task ontology the focus is on control-flow and data perspectives, and since in this paper we are using UML's class diagrams for representing the structural view of a task ontology, we decided to represent the behavioral view using the UML's activity diagrams. An activity diagram is a behavior model that allows capturing control-flow, i.e., the order of activities (sub-tasks) and their conditions to execution. It also allows representing the relation with identified knowledge roles. Thus, it is possible to model the agents that are responsible for performing the sub-tasks, and the inputs and outputs of them, specifying the previous and subsequent states of each object (role) participating in a sub-task.

Figures 2, 3 and 4 show the behavioral perspective of the lending task ontology, showing its control-flows, inputs, outputs and agents that perform each sub-task.

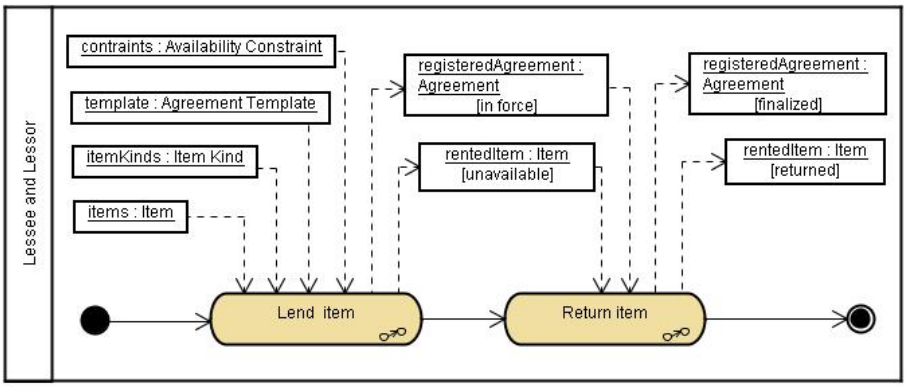

Fig. 2. The main sub-tasks of the lending task.

Figure 2 presents the activity diagram for the whole lending task. Two main sub-tasks were identified: "Lend item" and "Return item", which are further refined in other sub-tasks, shown in figures 3 and 4, respectively. The swimlanes show the roles agents play in the task, i.e. which activities (sub-tasks) they perform or act in.

The data perspective in an activity diagram is shown by means of objects. Objects are instances of the concepts in the structural model of the ontology task, i.e. they are instances of the knowledge roles identified in the structural perspective. Optionally, states of those objects can be shown.

As depicted in Figure 3, the lending task starts with the lessee choosing, among the item kinds, the kind she/he wants to borrow an item. Once informed the selected kind, the lessor looks for available items among the items of the selected kind. Availability constraints should be checked to decide if an item is available or not. If there are available items, the lessee or the lessor (depending on the specific problem at hands) chooses one of them to borrow. Chosen the item, the next step is to establish an agreement. The lessor can use a template in order to propose the agreement. The lessee should verify the clauses of the proposed agreement. If they are accepted by the lessee, the lessor registers the agreement, which comes in force. The chosen item are considered rented and thus unavailable.

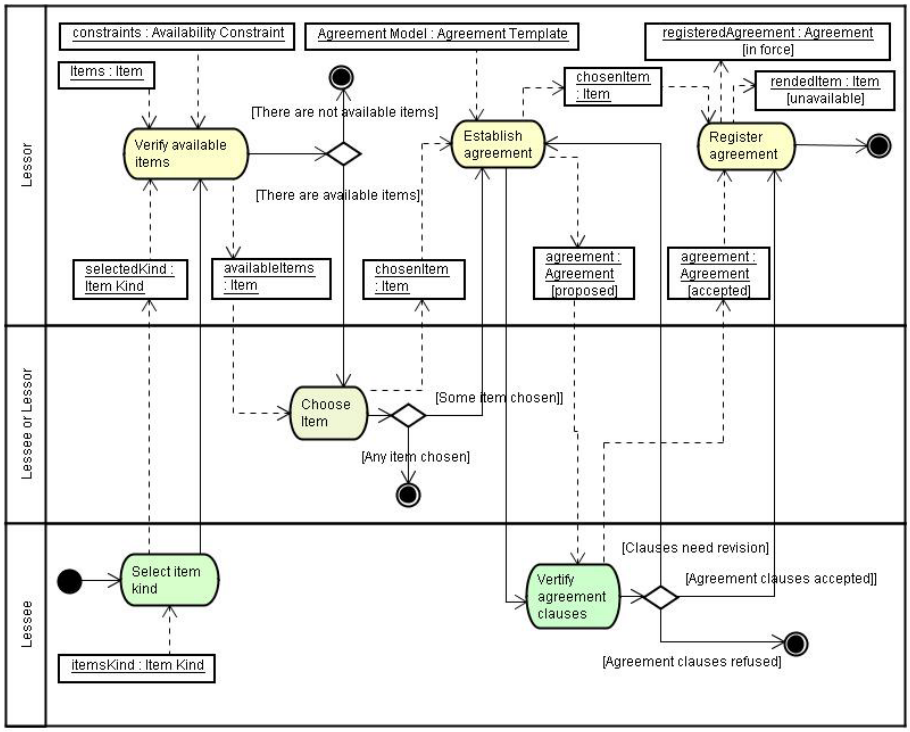

Fig. 3. The sub-task “Lend item”.

Although the activity diagrams represent the main aspects of the control-flow of a task, it is not possible to represent all its aspects. For example, each sub-task in the Figure 3 can be further decomposed into other sub-tasks. We think that there is a level of discourse in which it is better to provide descriptions in structured natural language for the sub-tasks, in spite of elaborating another activity diagram. This occurs when it

is simple enough to capture the control-flow with few sentences. The sub-task "Verify Available Items", for instance, can be described by the following algorithm:

1. Retrieve items of the selected item kind;
2. Retrieve available constraints that apply for the selected kind and to its instances.
3. For each item, check if any constraint is violated.
4. If for an item any constraint were violated, then consider it an available item.

In the case of the "Return item" sub-task (Figure 4), the sub-task begins with the lessee giving the rented item back. The lessor should then verify it, according to the registered agreement. If any of the agreement clauses are broken, the corresponding penalties should be applied. The lessee should pay off the debits, and after that the devolution is registered. In this last step, the agreement is finalized and the item is returned.

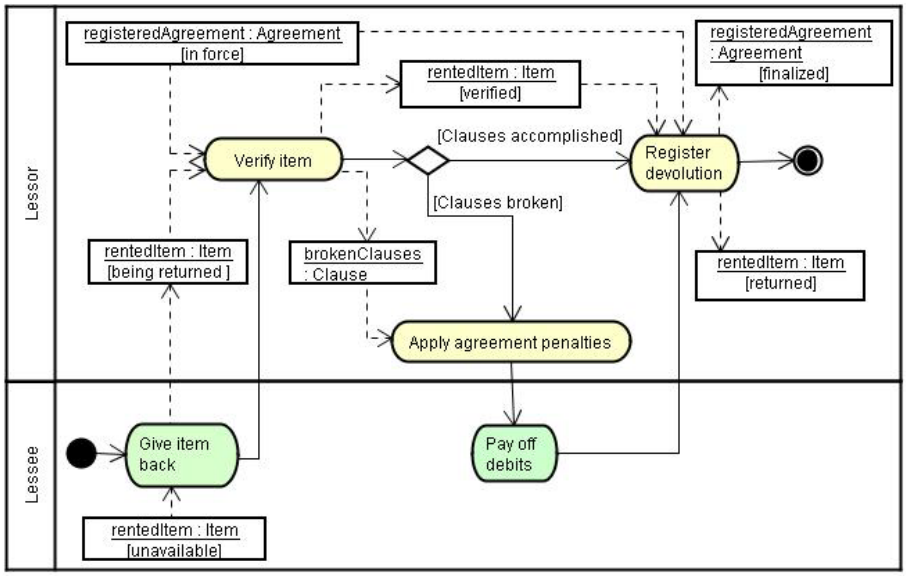

Fig. 4. The sub-task "Return item".

3.4 – Mapping domain ontologies to task ontologies

We can notice that many domains can be mapped to the lending task, such as books, automobiles, DVDs, homes and so on. This happens because the task ontology was developed in a domain-independent way. The descriptions are made using generics terms that describe the roles objects and agents should play in the task. This approach makes the integration of the lending task ontology with domain ontologies easy. Additionally, the usage of same notation for domain ontologies and task knowledge roles models also allows better and easier integration of these two models.

The integration consists in identifying which role a domain concept should play in the structural model of the task ontology. In other words, it is enough to define the functions of the domain entities in task fulfillment, since roles are mapped in the task control-flow.

Let's consider a car rental class of applications. Suppose that an ontology of the automobile domain consists of, among others, the following concepts: automobile, car (a subtype of automobile), automobile model, and owner. We can map those concepts to item, item kind and lessor, respectively.

Of course, neither all concepts in a domain ontology map to concepts in a task ontology, and vice-versa. However, using both, it is easier to achieve application ontologies, as defended by Guarino [1]. In the example of a car rental application ontology, we can see that the concept "lessee" in the task ontology does not have a correspondent concept in the domain ontology. Thus, we can include a new concept (customer) for capturing this.

4 Conclusions

More and more task ontologies are receiving attention. However, differently of domain ontologies, which have several methods and notations devoted to them, the development of task ontologies lacks principled approaches. In this paper, we intended to go a step ahead in this direction, focusing on models for representing task ontologies and their inter-relationship. Since task knowledge involves both a structural and a behavioral dimension, we proposed the use of UML's class and activity diagrams, respectively, for representing these two dimensions. We also discussed how these two perspectives complement one another. Finally, we briefly discuss how domain ontologies could be merged with task ontologies in order to produce application ontologies.

Regarding the separation in two views, it is worthwhile to point that this division facilitates the construction and organization of the knowledge task and the later integration with domain ontologies to derive of application ontologies. In addition to that, we think that the use of UML´s notation increases the understandability and reusability of this model, mainly because it is a well diffused notation in software development. As advocate by Wang and Xang [16], UML usage enables users and software engineers to better understand an ontology by using a common language for them. Increasing comprehensibility allows the ontology to be validated more quickly and reused in software development. So, in spite of UML does not dispose of formal semantics, UML's diagrams have been very used to model ontologies.

However, much more needs to be done. First, we need a method for systematically developing task ontologies. Maybe methods for developing domain ontologies can be adapted for this purpose. This is a work that we intend to do soon.

In this paper, we sketched an approach for developing application ontologies based on domain and task ontologies. This should be further studied and improved to give rise to a systematic approach for merging domain and task ontologies in the development of application ontologies. Such approach would be of great interest and would partially implement the leveled-strategy for developing ontologies suggested

by Guarino [1]. He proposes developing ontologies according to their level of generality: top-level, domain, task and application ontologies. Top-level ontologies describe very general concepts, which are independent of a particular problem or domain. Domain and task ontologies describe, respectively, the vocabulary related to a generic domain and a generic task, by specializing the terms introduced in the top-level ontology. Finally, application ontologies describe concepts depending on both a particular domain and task, which are often specializations of both the related ontologies. To completely address Guarino's proposal, we are studying how to use UFO (Unified Foundational Ontology) [2] [27] as a top-level ontology and OntoUML [2], a UML profile developed based on UFO, to represent domain and task ontologies. Initial results concerning domain ontologies have already been achieved, and were published in [27].

Finally, since the main goal for capturing knowledge is to allow its reuse and sharing, we are also working on an approach for reusing task ontologies in the requirements engineering process.

Acknowledgments

This work was accomplished with the financial support of FAPES, a foundation supporting science and technology from the government of the state of Espírito Santo in Brazil, and VixTeam, a partner organization that has also financed this project.

References

1. N. Guarino, "Formal Ontology and Information Systems". In: Formal Ontologies in Information Systems, N. Guarino (Ed.), IOS Press, 3 -15, 1998.
2. G. Guizzardi, "Ontological Foundations for Structural Conceptual Models", Universal Press, The Netherlands, 2005, ISBN 90-75176-81-3.
3. L. Zong-yong , W. Zhi-xue, Y. Ying-ying, W. Yue, L. Ying, "Towards a Multiple Ontology Framework for Requirements Elicitation and Reuse", Proceedings of the 31st Annual International Computer Software and Applications Conference (COMPSAC 2007), Vol. 1, p. 189-195 Beijin, China, 2007.
4. M. Ikeda, K. Seta, O. Kakusho, R. Mizoguchi, "Task ontology: ontology for building conceptual problem solving models". Proceedings of ECAI98 Workshop on Applications of ontologies and problem-solving models, 1998, pp. 126-133.
5. W.B. Frakes, K. Kang, "Software Reuse Research: Status and Future", IEEE Transactions on Software Engineering, vol. 31, no. 7, pp. 529-536, Jul. 2005.
6. R.A. Falbo, G. Guizzardi, K.C. Duarte, "An Ontological Approach to Domain Engineering". Proceedings of the 14th International Conference on Software Engineering and Knowledge Engineering - SEKE'2002, p. 351-358, Ischia, Italy, 2002.
7. J. Greenfield, K. Short et al., "Software Factories: Assembling Applications with Patterns", Models, Frameworks, and Tools, John Wiley & Sons, 2004.
8. N. Guarino. "Understanding, building and using ontologies". Int. Journal Human-Computer Studies, 46(2/3), February / March 1997.
9. R. Mizoguchi, Y. Tijerino, M. Ikeda, "Task Analysis Interview Based on Task Ontology", Expert Systems with Applications, vol. 9, no. 1, pp 15-25, 1995.

10. A. Gómez-Pérez, M. Fernández-López, O. Corcho, “Ontological Engineering”, Springer-Verlag, 2004.
11. S. Cranefield, M. Purvis. “UML as an Ontology Modelling Language”, In: Proceedings of the IJCAI-99, Workshop on Intelligent Information, 16th International Joint Conference on AI, Stockholm, Sweden, July 1999.
12. R.A. Falbo, “Experiences in Using a Method for Building Domain Ontologies”. Proc. of the 16th International Conference on Software Engineering and Knowledge Engineering, International Workshop on Ontology In Action. Banff, Canada, 2004.
13. J. Breuker, W. Van de Velde, “CommonKADS Library for Expertise Modelling”, IOS Press, 1994.
14. R. Mizoguchi, J. Vanwelkenhuysen, M. Ikeda, “Task Ontology for Reuse of Problem Solving Knowledge”. In Building and Knowledge Sharing 1995 (2nd International Conference on Very Large-Scale Knowledge Bases), Enschede, The Netherlands, pp. 46–59, 1995.
15. K. Fang, “Modeling Ontology-based Task Knowledge in TTIPP”, Proceedings of the 8th WSEAS International Conference on Automation and Information, Vancouver, Canada, June, 2007.
16. X. Wang, C.W. Chan, “Ontology Modeling in UML”. In Proceedings of OOIS 2001, Calgary, Canada, Springer-Verlag, 59-68, 2001.
17. D. Rajpathak, E. Motta, R. Roy, “A Generic Task Ontology for Scheduling Applications”. In Proceedings of the International Conference on Artificial Intelligence (IC-AI'2001), Nevada, Las Vegas, USA, 2001.
18. D. Rajpathak, E. Motta. “An Ontological Formalization of the Planning Task”, Proceedings of the International Conference on Formal Ontologies in Information Systems (FOIS'04), pp. 305-316, Torino, Italy, 2004.
19. F. Zlot, K. M. Oliveira, A.R. Rocha. “Modeling Task Knowledge to Support Software Development”, Proceedings of the 14th International Conference on Software Engineering and Knowledge Engineering (SEKE'2002), p. 35-42, Ischia, Italy, 2002.
20. OMG, “Business Process Modeling Notation Specification”, Final Adopted Specification, February 2006.
21. G. Booch, J. Rumbaugh, I. Jacobson, “Unified Modeling Language User Guide”, 2nd Edition, Addison-Wesley Professional, 2005.
22. S. J.Mellor, K. Scott, A. Uhl, D. Weise. “MDA Distilled”. Addison Weseley Professional, 2004.
23. B. Chandrasekaran, J.R. Josephson, and R. Benjamins. “The Ontology of Tasks and Methods”, In Proceedings of the 11th Knowledge Acquisition Modeling and Management Workshop, KAW'98, Banff, Canada, April 1998.
24. P.G. Mian, R.A. Falbo, ‘Supporting Ontology Development with ODEd’, Journal of the Brazilian Computer Science, vol. 9, no. 2, November, 57-76, 2003.
25. S. A. White. "Process Modeling Notations and Workflow Patterns," BPTrends, March 2004.
26. N. Russell, W.M.P. van der Aalst, A.H.M. ter Hofstede, P. Wohed, “On the suitability of UML 2.0 activity diagrams for business process modeling”, Proceedings of the 3rd Asia-Pacific Conference on Conceptual Modeling, Hobart, Australia, pp. 95-104, 2006.
27. G. Guizzardi, J.O. Vasconcelos, B. Segrini, R.A. Falbo, R.S.S. Guizzardi, “Grounding Software Domain Ontologies in the Unified Foundational Ontology (UFO): The case of the ODE Software Process Ontology”, Proceedings of the XI Iberoamerican Workshop on Requirements Engineering and Software Environments, Recife, Brazil, 2008.