Task Representation and Retrieval in an Ontology-Guided Modelling System*

Yuan Ren[1], Jens Lemcke[2], Tirdad Rahmani[2], Andreas Friesen[2], Srdjan Zivkovic[3], Boris Gregorcic[3], Andreas Bartho[4], Yuting Zhao[1] and Jeff Z. Pan[1]

[1]University of Aberdeen, [2]SAP AG, [3]BOC Information Systems GmbH, [4]Technische Universität Dresden

Abstract. A modelling procedure consists of a sequence of modelling tasks. With the increasing size of models during the development, the relations among modelling objects and their corresponding tasks become more and more complex. Therefore, the automatic identification of available tasks for current modelling environments can significantly improve the efficiency of modelling. In this paper, we present a novel approach which integrates task pre- and post-conditions into ontologies that describe the models, and infers availability of tasks through ontology reasoning and query answering. We illustrate this approach with an application on process modelling and show how this can be generalised to other domains.

1 Introduction

In model-driven software development (MDSD), a modelling procedure consists of a sequence of modelling tasks. These tasks are performed by particular modelling engineers to manipulate the modelling objects. With the increasing size of models produced and utilized during the development, the relations among modelling objects and their corresponding tasks become more and more complicated. It is usually difficult for developers to manually identify the tasks. Therefore, the automatic identification of available tasks for current modelling environment can significantly improve the efficiency of modelling.

In this paper, we discuss how to formalise and infer the knowledge about development tasks. With an illustrative example concerning process modelling and refinement, we show that the relations between modelling tasks and objects can be represented by ontology language OWL 2 EL. We also show that with such ontological representation, the tasks can be automatically inferred.

The rest of the paper is organised as follows. Section 2 gives a brief introduction about the syntax of OWL 2 EL. Section 3 introduces the case studies about guidance in modelling system. Section 4 discusses how to formalise the guidance knowledge into ontologies and how to infer task information. At the end section 5 summaries the paper.

* This work has been supported by the European Project Marrying Ontologies and Software Technologies (EU ICT 2008-216691).

Proceedings of OWL: Experiences and Directions 2009 (OWLED 2009),
Rinke Hoekstra and Peter F. Patel-Schneider, editors. http://www.webont.org/owled/2009

2 Ontology Language OWL 2 EL

Web Ontology Langue (OWL) is the de facto standard language for ontology recommended by W3C. In its 2nd version, OWL 2, there are three profiles supporting tractable reasoning, namely OWL 2 EL, OWL 2 QL and OWL 2 RL [6]. OWL 2 EL is based on the $\mathcal{EL}$ family of Description Logics [1, 2]. Its corresponding underpinning logic is $\mathcal{EL}^{++}$.

In OWL 2 EL, concepts can be constructed as top concept, bottom concept, atomic concept, singleton, conjunction or existential restriction of other concepts. General concept inclusions, concept disjointness, domain, range restrictions and property chain axioms are supported by OWL 2 EL.

A important feature of OWL 2 EL is that it disallows the usage of negation, disjunction and universal restriction of concepts. With such restrictions, classification in OWL 2 EL is PTime-complete and efficient algorithms [3] have been developed and implemented. Although the expressiveness is limited to maintain the tractability of the language, OWL 2 EL has been widely applied in real world ontologies such as Gene Ontology, NCI ontology, SNOMED, and others.

Basic reasoning services such as concept subsumption checking, instance retrieval and conjunctive query answering are supported for OWL 2 EL. Due to the introduction of the bottom concept, OWL 2 EL also supports concept satisfiability and ontology consistency checking.

3 Guidance in Modelling Systems

Here we discuss the guidance issue for a computer-aided modelling system. We first introduce some important notions and knowledge assets for guidance in general. Then we illustrate them with process modelling and refinement.

3.1 Concepts and Knowledge Assets of a Modelling Scenario

There are several important notions in modelling procedures, such as artefact, task, role, etc. In this paper, we are mostly interested in capturing the semantics of the following notions:

1. **Tasks** of different types can be performed in a particular modelling environment, such as create a model, remodel a model, refine a model, etc.
2. **Pre- and Post- Conditions** describe the prerequisites and effects of tasks of certain types. In legacy systems, they are usually described in natural languages. For example, the post-condition of creating a model is that a model is created.
3. **Artefacts** are the various entities that can be input or output by the modelling environment. Usually, an artefact can be a model or its component.

The relations between tasks, conditions and artefacts are also interesting. Generally speaking, the status of artefacts satisfy some pre-conditions, thus enable corresponding tasks. When a task is performed, its post-conditions will

result in changes of artefacts. Although trust and security issues such as access control are also important in guidance systems, our major concern in this paper is the pro-active identification of tasks.

The guidance functionality should be designed and developed independently from concrete modelling scenarios. For a specific application, except for the above notions, the following knowledge assets have to be considered:

1. **Domain Meta-model** defines the syntax of the models and global constraints that are independent from concrete task types. Because in the modelling systems, concrete models are regarded as model instances, the meta-model actually corresponds to TBox in an ontology.
2. **Model Knowledge** is the concrete status of the models under development. In contrast to the meta-model, models correspond to the ABox of an ontology. Given the fact that models will be constantly changed during the development procedure, the ABox will also be frequently updated.
3. **Task Knowledge** characterises the pre-/post-conditions of specific task types. This knowledge is the interaction between artefacts and task types, therefore they can not be solely defined in the meta-model. The proper interpretation of task knowledge and its integration with meta-model/model is the major challenge of an ontology-guided modelling system.
4. **Queries** are used to retrive tasks and artefacts. They should be designed in a way that they are as independent from concrete domains as possible.

For an application scenario, the above knowledge assets should be integrated into a generic guidance engine to form an dedicated system. In the next subsection, we use process modelling and refinement as an example to introduce the above notions in detail. Our focus will be knowledge about various task types.

3.2 Process Modelling and Refinement

Processes are important types of models in software development. They are generalised representations of control flow, data flow, etc. The results of a development are usually produced and utilized through a process. Therefore the modelling of processes is crucial for the planning and organization of development.

In MDSD, processes are usually designed step by step on different levels of abstraction. This creates refinement chain of the process models. In [7], an ontological solution for validating BPMN process refinement is presented.

The artefacts in this example include, among others, *Process*, *Activity*, *Component Behavior Model*, etc. The meta-model includes constraints such as "A process contains only activities (including start and end) and gateways". The task types include *Remodel Process*, *Refine Process*, *Ground Process*, etc. Knowledge about some typical tasks and their pre- and post-conditions can be described in natural language as follows:

1. **Remodel Process**: an engineer can always remodel an existing process.
   Pre-condition: a process exists.
   Post-condition: the process is remodeled.

2. **Refine Process**: when a process is neither refined nor grounded, the process needs to be refined by another process.
   Pre-condition: a process neither refined nor grounded exists.
   Post-condition: another process is created or referred to as the refinement of the current process.
3. **Ground Process**: any process that can be refined can also be grounded to a component behavior model.
   Pre-condition: a process neither refined nor grounded exists.
   Post-condition: a component behavior model is created or referred to as the grounding of the current process.

When an user is modelling processes, the system should automatically tell which task is available for which artefact. When the user performs a task and hence changes the models, task availability will also be updated accordingly.

4 Task Representation and Retrieval via Ontologies

In the specification of the case studies, the pre- and post- conditions of a task are described in natural language. However, a machine-readable specification requires representation in a formal language. In this paper, we use an ontology to represent the knowledge. Then, the challenges become: (1) Formalising the task knowledge by ontologies; (2) Reduction of task retrieval to ontology reasoning problems. In the following, we present our answers to these two challenges. We use the process modelling and refinement use case to illustrate the solution. After that, however, we will generalise the solution to provide a generic approach. At the end, we will discuss the computational aspect of this approach.

4.1 Formalising Guidance Knowledge into Ontologies

Domain Ontology: Meta-model and Model Intuitively, various artefacts can be categorized into concepts such as $Process$, $Activity$, $ComponentBehaviorModel$, etc. These concepts have a common super concept $Artefact$. The relation between these concepts are modeled as object properties. The concrete modelling entities will be instances. We call such an ontology the domain ontology. Its TBox (ABox) corresponds to the meta-model (Models) of the domain. As we will show, domain ontology has little influence on the inference mechanism of tasks, so we can regard them as separate.

Task Ontology Various tasks can be categorized into concepts such as $RefineProcess$, $GroundProcess$, etc. These task types have a common super type $Task$. Once a task is performed, Task ontology ABox will be updated.

The pre-conditions of a task type can be described by axioms. For example, when a process is **NOT** refined or grounded, it should be refined or grounded. This actually implies that, **EVERY** process should be refined or grounded, to either an existing process/component behavior model, or an implicit one. The

former implies that a **Refine Process** or **Ground Process** task has already been performed. The latter implies that a **Refine Process** or **Ground Process** tasks has to be performed. Therefore, the existence of a process actually becomes the pre-condition of a **Refine Process** or **Ground Process** task.

However, the disjunction implies that, neither **Refine Process** nor **Ground Process** is really **compulsory** for processes, but **Refine or Ground Process** is. We call these two *Alternative Tasks*. If we query for one of a set of alternative tasks, the ontology will not infer its necessity. We have to query for all of them. To solve this problem, we introduce a new task type $RefineOrGroundProcess$ as the super concept of both $RefineProcess$ and $GroundProcess$. Of course, $RefineOrGroundProcess$ will also be a sub-concept of $Task$. We can model such semantics with an axiom in Manchester Syntax [4] as follows:

$$SubClassOf:\ Process,\ preconditionOf\ some\ RefineOrGroundProcess$$

Once a task of $RefineOrGroundProcess$ is found to be needed, we shall generate two tasks $RefineProcess$ and $GroundProcess$ to be displayed for the user. This means $RefineOrGroundProcess$ will not have direct instances. Once a $RefineProcess$ or $GroundProcess$ task is performed on an artefact, a $RefineOrGroundProcess$ is regarded as performed because an instance of $RefineProcess$ or $GroundProcess$ is also an instance of $RefineOrGroundProcess$.

Regarding post-conditions, the effect of the task is the creation of another process or component behavior model. Thus, the existence of such a process or component behavior model actually becomes the post-condition of the task that is either performed or to be performed. Because $RefineOrGroundProcess$ is a "abstract" task type, we only model the postcondition for $RefineProcess$ and $GroundProcess$:

$$SubClassOf:\ RefineProcess,\ hasPostcondition\ some\ Process$$

$$SubClassOf:\ GroundProcess,\ hasPostcondition\ some\ ComponentBehaviorModel$$

Similarly, we will have **Remodel Process:**

$$SubClassOf:\ Process,\ preconditionOf\ some\ RemodelProcess$$

$$SubClassOf:\ RemodelProcess,\ hasPostcondition\ some\ Process$$

From these axioms, we can generalise the formalization patterns as:

$$SubClassOf:\ [Artefact],\ preconditionOf\ some\ [Task]$$

$$SubClassOf:\ [Task],\ hasPostcondition\ some\ [Artefact]$$

where $Task$ is a concrete type of task (or the super type of alternatives), $Artefact$ is a concrete type of artefact. Obviously, one artefact type can be pre-condition of multiple task types. As we can see, these two patterns are independent from concrete task types, and even concrete domains. This implies that we can also design generic patterns to retrieve tasks regardless of which type or domain it is.

4.2 Retrieving Tasks by Query Answering

Once we generate the domain ontology and task ontology by the axioms presented in the previous section, we can use a reasoner to automatically retrieve the tasks. Intuitively, this can be performed by querying the artefacts on which certain task types should or could be performed. For example, if we propose the following query:

$$?x \leftarrow ?x : Artefact, (?x, ?y) : preconditionOf, ?y : RemodelProcess$$

to a query engine, it will return all artefacts $?x$ such that there exists some instance of $RemodelProcess$ $?y$ of which $?x$ is the pre-condition. This literally presents all the processes that can be remodeled.

However, this query can not yet be generalised to other task types. For example, if we use the similar query for $RefineOrGroundProcess$ task, the results will include the processes that have already been refined or grounded. These redundancies are due to the fact that there are actually two categories of tasks:

1. **Compulsory Task**: a task that must be performed. Once performed, it is not necessary to perform it again. Task types such as **Refine Process** and **Ground Process** belong to this category.
2. **Optional Task**: a task that could be performed. Once performed, it can still be performed again. Task types such as **Remodel Process** belong to this category.

Optional tasks can use the similar query pattern presented above. The presented compulsory tasks should only contain tasks that have not been performed yet, i.e. the **implicit** instances of tasks. They can be obtained by subtracting the performed ones from the whole set. Taking $RefineOrGroundProcess$ as an example, we propose the following two queries:

$$?x \leftarrow ?x : Artefact, (?x, ?y) : preconditionOf, ?y : RefineOrGroundProcess$$

$$?x, ?y \leftarrow ?x : Artefact, (?x, ?y) : preconditionOf, ?y : RefineOrGroundProcess$$

in which the first query returns all the processes that should be refined or grounded, the second query returns all the processes that have been refined or grounded, together with the corresponding tasks. The difference of the two will be the processes on which $RefineOrGroundProcess$ must be performed. Therefore the redundancy of compulsory tasks are resolved.

In order to distinguish the compulsory tasks and the optional tasks we introduce two concepts $CompulsoryTask$ and $OptionalConcept$ as the sub-concepts of $Task$ and super-concepts of all the compulsory tasks and optional tasks, respectively. Then the query can be generalised as follows:

1. for each direct sub-concept T of $OptionalTask$, propose query

$$?x \leftarrow ?x : Artefact, (?x, ?y) : hasOptionalTask, ?y : T$$

The solution will be the artefacts on which task type T could be performed.

2. for each direct sub-concept T of $CompulsoryTask$, propose two queries

$$?x \leftarrow ?x : Artefact, (?x, ?y) : hasCompulsoryTask, ?y : T$$

$$?x, ?y \leftarrow ?x : Artefact, (?x, ?y) : hasCompulsoryTask, ?y : T$$

The difference of solution $?x$ will be the artefacts on which task type T should be performed. Note that individual alternative concepts will not be tested, but their common super concept will be. Due to the introduction of common super-concepts for alternative concepts, T needs to be translated before presented to user.

4.3 Generalised Solution for Representation and Retrieval

Now we summarise the above findings to provide a generalised solution:

- Defining the domain ontology.
- For alternative tasks, introducing common super task type.
- Categorizing compulsory and optional tasks.
- For each type of compulsory task, proposing two queries and getting the difference to retrieve the artefacts to which such type of task is necessary.
- For each type of optional task, proposing one query to retrieve the artefacts on which such type of task can be performed.
- Translating the query results to generate the task list.
- Updating the ABox with the relations of artefacts and tasks such that compulsory tasks will not be repeated.

As we can see, the above solution is independent from the concrete task types and even application scenarios, thus it can be generalised. Actually, when applied to different domains, the system only needs to load the Domain Ontology and Task Ontology, then generates queries on the available task types.

4.4 Computational Properties

We first review the language needed. There are two major types of axioms: one for pre-conditions and the other for post-conditions. They are both within the expressive power of OWL 2 EL, especially considering that disjunction of task types must be resolved.

The reasoning services requested include both TBox classification and conjunctive query answering. In order to present a generic solution for guidance, it is necessary to automatically detect all the concrete compulsory task types and optional task types instead of hard-coding them into queries. This can be easily realised by getting all the direct-subconcepts of $CompulsoryTask$ and $OptionalTask$, which is a service provided by TBox classification.

When processing the queries, especially the first query of compulsory task and the query of optional task, it is important to notice that the variable $?y$ is not returned. This implies that $?y$ is a non-distinguished variable, which can

be bound to either an existing individual, or an implicit individual. In terms of ontology reasoning, this arises the requirement of query answering under Open World Assumption (OWA). Query answering with non-distinguished variables is an open issue for expressive DLs. And it is even proved that query answering in arbitrary $\mathcal{EL}^+$ ontology is already undecidable [5]. However, we restrict the task ontology to be regular, for which query answering algorithm and implementation has already been developed [8]. Also, in the current example the query can be rewritten into a instance retrieval of, e.g. $preconditionOf someRemodelProcess$.

5 Conclusion

In this paper we discuss how to represent the pre- and post-condition of tasks in a modelling system with OWL 2 EL ontologies and how to retrieve the tasks with ontology reasoning and query answering. We illustrate our solution with an example of process modelling and refinement and further give principles about generalization.

In the future, we will integrate the task ontology presented in this paper with different domain ontologies and implement a prototype of the ontology-guided modelling system.

References

1. F. Baader, S. Brandt, and C. Lutz. Pushing the $\mathcal{EL}$ envelope. In *Proceedings of the Nineteenth International Joint Conference on Artificial Intelligence IJCAI-05*, Edinburgh, UK, 2005. Morgan-Kaufmann Publishers.
2. Franz Baader, Sebastian Brandt, and Carsten Lutz. Pushing the el envelope further. In Kendall Clark and Peter F. Patel-Schneider, editors, *In Proceedings of the OWLED 2008 DC Workshop on OWL: Experiences and Directions*, 2008.
3. Franz Baader, Carsten Lutz, and Boontawee Suntisrivaraporn. Is tractable reasoning in extensions of the description logic el useful in practice? In *Proceedings of the 2005 International Workshop on Methods for Modalities (M4M-05)*, 2005.
4. Matthew Horridge, Nick Drummond, John Goodwin, Alan L. Rector, Robert Stevens, and Hai Wang. The manchester owl syntax. In Bernardo C. Grau, Pascal Hitzler, Conor Shankey, Evan Wallace, Bernardo C. Grau, Pascal Hitzler, Conor Shankey, and Evan Wallace, editors, *OWLED*, volume 216 of *CEUR Workshop Proceedings*. CEUR-WS.org, 2006.
5. Markus Krötzsch, Sebastian Rudolph, and Pascal Hitzler. Conjunctive queries for a tractable fragment of owl 1.1. In *ISWC/ASWC*, pages 310–323, 2007.
6. Boris Motik, Bernardo Cuenca Grau, Ian Horrocks, Zhe Wu, Achille Fokoue, and Carsten Lutz. Owl 2 web ontology language: Profiles. W3c working draft, W3C, October 2008.
7. Yuan Ren, Gerd Gröner, Jens Lemcke, Tirdad Rahmani, Andreas Friesen, Yuting Zhao, Jeff Z. Pan, and Steffen Staab. Validating process refinement with ontologies. In *Proceedings of the 22nd International Workshop on Description Logics (DL2009)*, 2009.
8. Yuting Zhao, Jeff Z. Pan, and Yuan Ren. Implementing and evaluating a rule-based approach to querying regular el+ ontologies. In *In Proc. of the International Conference on Hybrid Intelligent Systems (HIS2009)*, 2009.