Deliverable T1.3_D1.3 Building explicit semantic domain model : A Rodin Plugin and case studies

December 2016

Project IMPEX

ANR-13-INSE-0001

IRIT, ENSEEIHT, Université de Toulouse
LABRI, Université de Bordeaux & CNRS
LORIA, Telecom Nancy, Université de Lorraine
LRI, CentraleSupelec
SAMOVAR, Telecom Sud Paris
Systerel, Aix-en-Provence
http://impex.loria.fr

16/12/2013-17/12/2017

Contents

Chapter 1

Introduction

In the context of IMPEX, two modeling approaches are proposed to formalize ontology description using the Event-B models [3]. The first one defines a formal approach to encode ontologies directly as Event-B Contexts elements (shallow approach). The second approach uses deep modeling in which, first, ontology generic concepts are defined in Event-B Contexts and then ontologies are defined as specific instances of these generic models in other Event-B Contexts [1].

This report describes the *OntoEventB* Eclipse/Rodin plug-in that implements the proposed approaches. This plug-in has been developed to automatically support the formalisation of ontologies, described with ontology description languages like OWL, using set theory and predicate logic supported by the Event-B method.

This report is organised as follows: chapter 2 presents the *OntoEventB* plug-in internal architecture and chapter 3 describes how to install and to use the *OntoEventB* plug-in. Chapter 4 presents a conclusion and an overview of ongoing work.

Chapter 2

The OntoEventB architecture

2.1 Introduction

The *OntoEventB* plug-in has been developed within the IMPEX project to automatically support the translation of ontologies models, described using ontology description languages such as OWL [4] or Plib [5], into Event-B Contexts. It takes as input an ontology description file and generates, according to the selected approach (shallow or deep approach), the corresponding Event-B Context. In the following sections, we present the *OntoEventB* plug-in internal architecture.

2.2 The OntoEventB plug-in architecture

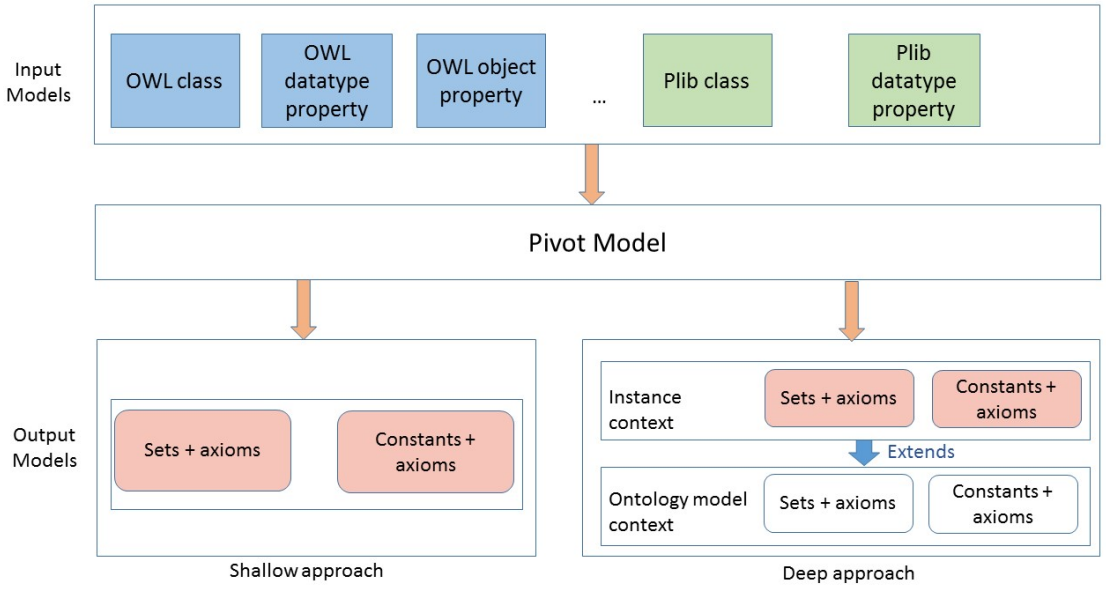

Figure 2.1: The OntoEventB architecture.

The *OntoEventB* plug-in is developed according to an architecture composed of three components: Input Models, Pivot Model and Output Models (Figure 2.1). This internal architecture is described in the following sections.

2.2.1 Input Models component

The Input Models component is devoted to treat the input models described using different ontology description languages such as OWL, Plib ... It browses the input models files in order to extract ontological concepts descriptions (OWL classes, OWL data type properties and OWL object properties in the case of OWL models) and to send them to the Pivot Model component.

This component can integrate any ontology descriptions used as input models. The actual version of *OntoEventB* plug-in treats only OWL models but we expect to extend it to take into account other ontologies description languages like Plib.

2.2.2 Pivot Model component

The Pivot Model component is an intermediate model which summarizes the common pertinent concepts used by ontology description languages. It defines generic concepts that integrate all specific concepts that can be received from the Input Model component. The Pivot Model can be extended to integrate others generic concepts that can be identified if a new language is added as input model in the Input Models component. So, we propose to progressively define the generic concepts integrating one by one the specific languages concepts. For this purpose, we start by integrating OWL and Plib concepts to our Pivot Model.

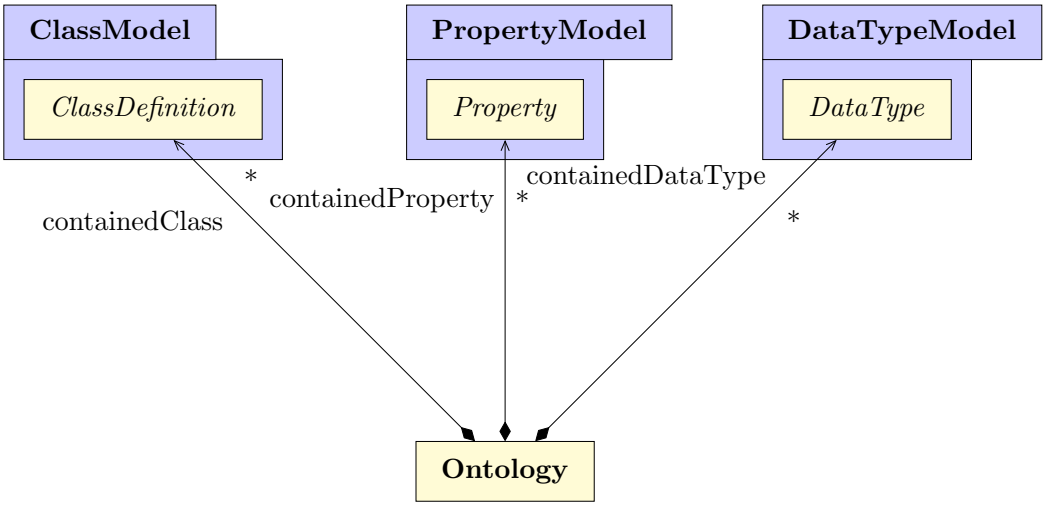

Figure 2.2: The main concepts of the Pivot Model.

Figures 2.2 presents an UML model showing the generics concepts that are defined in the proposed *Pivot Model*. An *Ontology* is defined by a set of classes (defined in the *ClassModel* part), a set of properties (defined in the *PropertyModel* part), and a set of data types (defined in the *DataTypeModel* part).

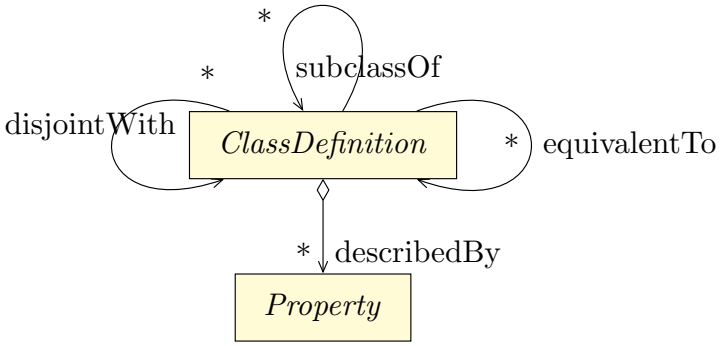

Figure 2.3: The *ClassDefinition* concept defined in the *ClassModel*.

The *ClassModel* part of the *pivot Model* presents the *class* concept that is described by a set of *properties*. This model defines also that a class can be sub-class of other classes (*subclassOf* definition), a class can be equivalent to other classes (*equivalentClassOf* definition) and/or a class can be disjoint with other classes (*disjointWith* definition) (Figure 2.3).

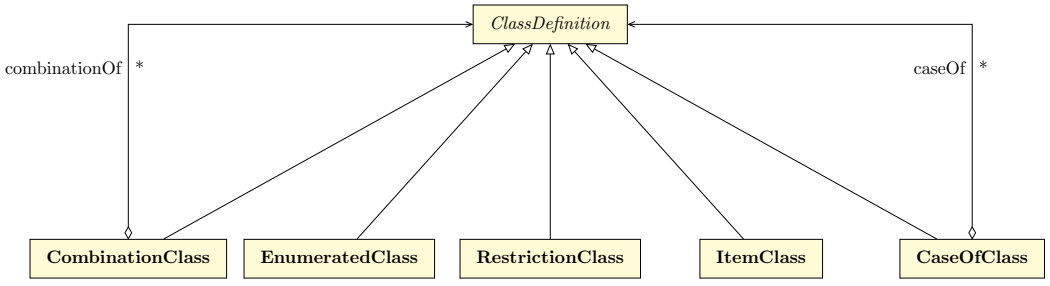

Figure 2.4: Differents types of *classes* defined by the *ClassModel*.

The *ClassModel* defines five types of classes (Figure 2.4): *ItemClass* for defining new named classes, *EnumeratedClass* for defining an enumerated class (list of individuals that are the instances of the defined class), *CombinationClass* for defining a new class as a combination of others classes (using *union, intersection* and *complement* operators), *restrictionClass* for defining a class of individuals that satisfy a restriction on a property (like in OWL language), and *CaseOfClass* for defining a class as a particular case of another classes (like *subClassOf* definition but without importing all properties of the mother classes).

The *PropertyModel* part of the *pivot Model* defines the *Property* concept that can be described by its domain (optional if property is not linked to a class definition) and its range. This model defines also that a property can be sub-property of others properties (*subPropertyOf* definition), a property can be equivalent to others properties (*equivalentTo* definition) and/or a property can be inverse of other property (*inverseOf* definition) (Figure 2.5).

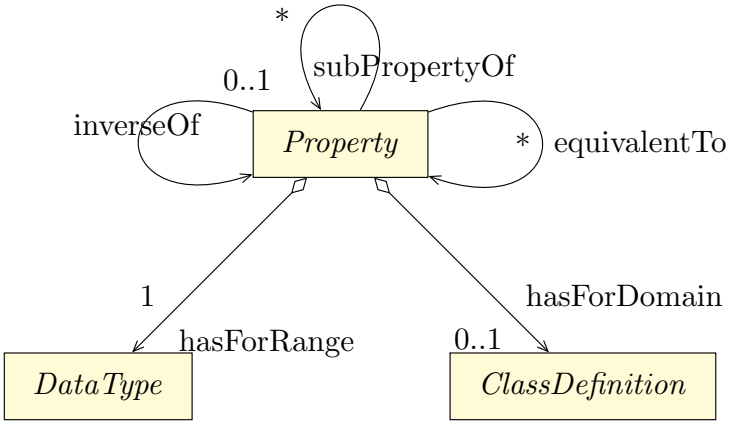

Figure 2.5: The *Property* concept defined in the *PropertyModel* model.

The *PropertyModel* defines three categories of properties (Figure 2.6): *NonDependentProperty* for defining independent properties, *DependentProperty* for defining dependent properties and *ConditionProperty* for defining properties on which *DependentProperty* depend.

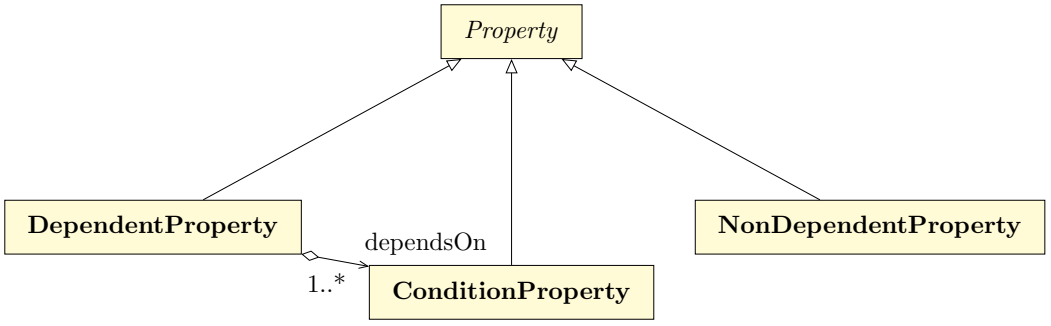

Figure 2.6: Differents types of *properties* defined by the *PropertyModel* model.

The range of the *Property* concept is formalized by the *DataType* class defined in the *DataTypeModel* part of the *pivot model* (Figure 2.7). The *DataTypeModel* supports the definition of *PrimitveType* (it contains *NotNumericType*, *NumericType*,

CurrencyType and *MeasureType*), the definition of *CollectionType* (bag, set, list, array, ...), the definition of *EnumeratedType* that represents a set of *SingleValue*, and the definition of *ClassInstanceType* that takes its values in the *ClassDefinition* class defined in the *ClassModel* model.

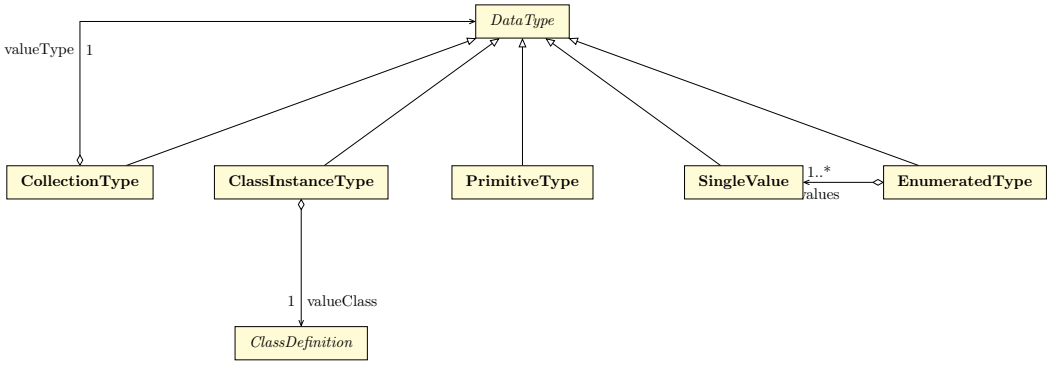

Figure 2.7: The *DataType* concept defined in the *DataTypeModel* model.

Following the receipt of different ontological concepts derived from the Input Model component (OWL classes, OWL data type properties and OWL object properties in the case of OWL models), the *Pivot Model* component translates them into generic concepts (classes, properties and data types). After this first translation step, the obtained generic concepts are ready to be treated by the next process handled by the Output Model component.

2.2.3 Output Model component

The Output Model component receives the generic concepts computed by the Pivot Model component and translates them into Event-B Context elements (sets, constants and axioms). This process uses transformation rules, proposed in the IMPEX project, that formalise each ontological concept by an Event-B definition. Two approaches have been developed (shallow and deep) and the user of the *OntoEventB* plug-in can choose one of them. The Shallow approach encodes ontologies concepts directly as Event-B Context elements, and the Deep approach uses deep modeling. In this case, ontologies generic concepts are formalised in a first Event-B Context, and the ontologies specific concepts are defined in a second Event-B Context as specific instances of the generic Contexts [3].

The use of this architecture allows us to extend the *OntoEventB* plug-in by taking into account new input ontology description languages without redefining the Event-B formalisation rules between Pivot Model component and Output Model component. Indeed, as soon as the new concepts defined by these new languages are translated into generic concepts of the Pivot model, they will be directly formalized in the Event-B Context elements without redefining new transformation rules.

Chapter 3

The OntoEventB plug-in

3.1 Introduction

The *OntoEventB* tool is developed as an Eclipse plug-in to be integrated into a Rodin platform [2] which is an Eclipse product and an IDE (Integrated Development Environment) supporting Event-B developments. In the following sections, we describe the *OntoEventB plug-in* installation and using processes.

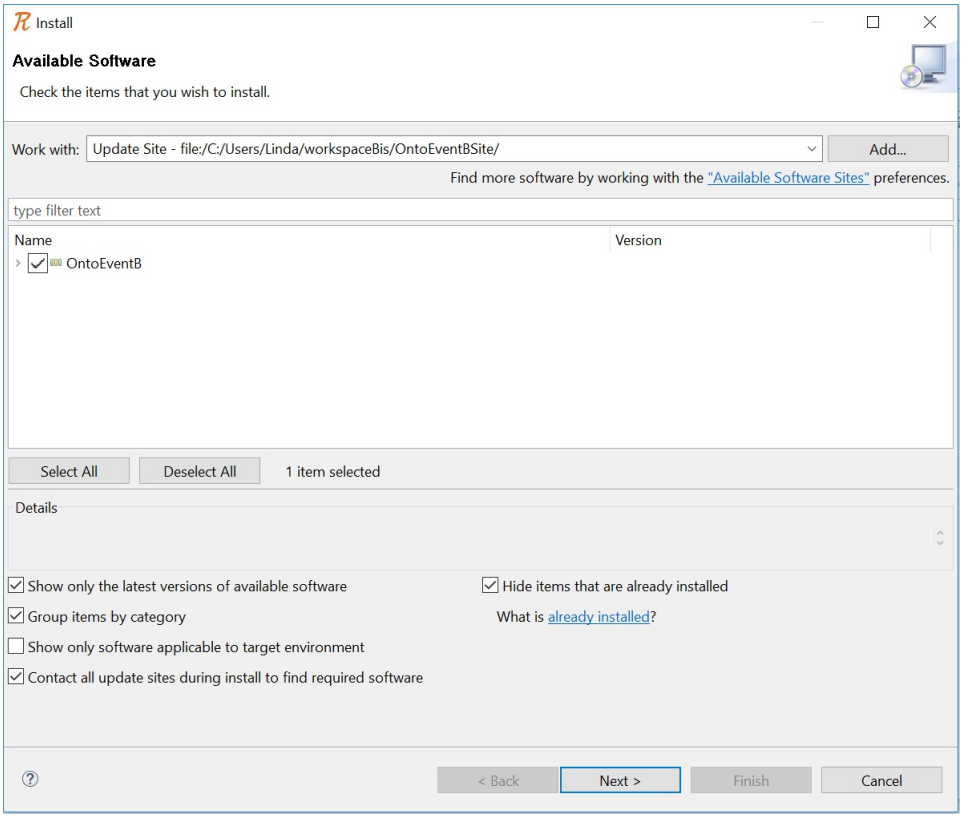

Figure 3.1: The Install New Software wizard of Rodin platform.

3.2 Installing and Using OntoEventB plug-in

3.2.1 Installing OntoEventB plug-in

To use *OntoEventB* plug-in in your Rodin platform instance, you must install a plug-in by using the *Install New Software* menu item accessible from the *Help* menu (Help $--> $ Install New Software). The use of this menu allows to launch the Install New Software wizard that offers you the possibility to add new software to your platform (see Figure 3.1). You click on the *Add* button and you may type the location of the update site[1] of the *OntoEventB* plug-in for downloading and installing the plug-in automatically.

3.2.2 Using the OntoEventB plug-in

After installing the *OntoEventB* plug-in in your Rodin platform instance, the *convert to Event-B* sub-menu becomes available by right clicking on a file with a *.owl* extension in the project explorer as shown in Figure 3.2. It proposes two functions: Deep method and Shallow method, corresponding to the two proposed approaches described in [3] with more detailed examples.

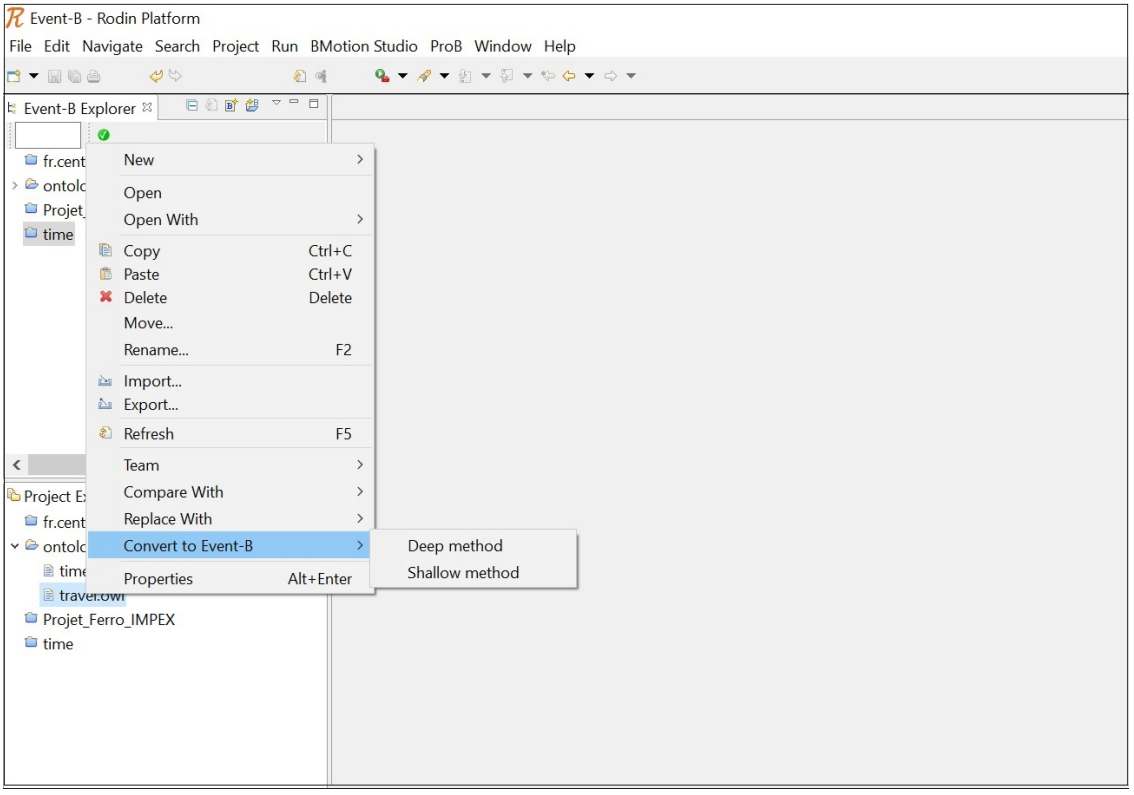

Figure 3.2: The OntoEventB submenu.

[1] OntoEventB update site:
`http://downloads.sourceforge.net/project/ontoeventb/update-site`

We illustrate the use of the two functions offered by the OntoEventB plug-in on an OWL ontology example relative to a travel domain. An extract of this ontology is presented below. It contains *four classes* (Accommodation, Destination, Activity and Contact), *one Object Property* (hasAccommodation) and *one Data type Property* (hasCity).

travel.owl

```
<owl:Ontology>
        ...
        <owl:Class rdf:ID="Accommodation"></owl:Class>
        <owl:Class rdf:ID="Destination"></owl:Class>
        <owl:Class rdf:ID="Activity"></owl:Class>
        <owl:Class rdf:ID="Contact"></owl:Class>

        <owl:ObjectProperty rdf:ID="hasAccommodation">
                <rdfs:range rdf:resource="#Accommodation"/>
                <rdfs:domain rdf:resource="#Destination"/>
        </owl:ObjectProperty>

        <owl:DatatypeProperty rdf:ID="hasCity">
                <rdfs:domain rdf:resource="#Contact"/>
                <rdf:type rdf:resource="FunctionalProperty"/>
                <rdfs:range rdf:resource="string"/>
        </owl:DatatypeProperty>
        ...
</owl:Ontology>
```

OntoEventB shallow function

The use of the shallow function with an OWL ontology leads to the creation of a new Event-B project containing a single Context C0 describing the ontology model by a set of sets, constants and axioms according to the shallow formalisation approach. The application of shallow function to the travel ontology generates the Event-B Context presented below.

Context C0

```
CONTEXT C0

SETS
        Thing String ...

CONSTANTS
        Accommodation, Destination, Activity, Contact,
```

hasAccommodation, hasCity, ...

AXIOMS

$@axm1 : Accommodation \subseteq Thing$

$@axm2 : Destination \subseteq Thing$

$@axm3 : Activity \subseteq Thing$

$@axm4 : Contact \subseteq Thing$

$@axm5 : hasAccommodation \in Destination \leftrightarrow Accommodation$

$@axm6 : ...$

$@axm7 : hasCity \in Contact \nrightarrow String$

...

END

The classes *Accomodation*, *Destination*, *Activity* and *Contact* are respectively defined in the axioms $axm1$, $axm2$, $axm3$ and $axm4$ as abstract sets. The *hasAccommodation* Object Property is defined by the axiom $axm5$ as a relation between *Destination* and *Accommodation* sets that represent the domain and the range of the *hasAccommodation* property. Finally, the axiom $axm7$ formalize the functional characteristic of the *hasCity* property. More details of the shallow transformation approach are given in [3].

OntoEventB deep function

The use of the deep function with an OWL ontology leads to the creation of a new Event-B project containing two Contexts: a first generic Context named *ontologyModel* defining generic ontology elements (this Context is generated for all ontologies), it defines the ontological generic concepts like CLASS, PROPERTY, INSTANCE, VALUES, ...

Context OntologyModel

CONTEXT OntologyModel

SETS

CLASS, PROPERTY, INSTANCE, VALUES

CONSTANTS

HAS_PROPERTIES, HAS_INSTANCES, HAS_VALUES, IS_A

AXIOMS

$@axm1 : HAS\_PROPERTIES = CLASS \leftrightarrow PROPERTY$

$@axm2 : HAS\_INSTANCES = CLASS \leftrightarrow INSTANCE$

$@axm3 : HAS\_VALUES = INSTANCE \times PROPERTY \leftrightarrow VALUES$

...

END

and a second Context named *C0* extending the *ontologyModel* Context. This second Event-B Context describes the ontology by instantiating the *ontologyModel* Context elements according to the deep transformation approach. The application of deep function to the travel ontology generates the Event-B Context presented below.

Context C0

CONTEXT C0 EXTENDS OntologyModel

CONSTANTS
Thing, Accommodation, Destination, Activity, Contact,
hasAccommodation, hasCity

AXIOMS
@$axm1 : partition(CLASS,$
$\{Thing\}, \{Accommodation\}, \{Destination\}, \{Activity\}, \{Contact\})$
@$axm2 : partition(PROPERTY, \{hasAccommodation\}, \{hasCity\})$
@$axm3 : hasProperties =$
$\{Destination \mapsto hasAccommodation, Contact \mapsto hasCity\}$
@$axm4 : hasProperties \in HAS\_PROPERTIES$
...
END

The axiom $axm1$ defines the set of classes *Accomodation*, *Destination*, *Activity* and *Contact*, the axiom $axm2$ defines the set of properties *hasAccommodation* and *hasCity*, and axioms $axm3$ and $axm4$ define properties relations to their domain definition. For more details about the deep approach refer to [3].

Chapter 4

Conclusion

4.1 Conclusion

This report presents the *OntoEventB plug-in* supporting the automatic generation of Event-B contexts from ontologies descriptions. It presents the internal architecture of the tool, its installation procedure and shows how Event-B contexts can be automatically generated according to the two proposed approaches (shallow and deep).

4.2 Ongoing work

The presented tool gave rise to several extensions. We are currently working on:

1. Completing the deep approach part with the remaining transformation rules. Actually, the OntoEventB deep function encodes only some ontology concepts (classes, properties, class inheritance ...).

2. Extending the tool to emphasis other input ontologies descriptions such as Plib ontologies.

3. Developing a more accomplished pivot model describing the ontologies in a conceptual level, this conceptual model constitute an independent representation of ontologies requiring a specific description language, an input editor and a transformation tool to Event-B contexts.

Bibliography

[1] J.-R. Abrial. *Modeling in Event-B: system and software engineering*. Cambridge University Press, 2010.

[2] J.-R. Abrial, M. J. Butler, S. Hallerstede, T. S. Hoang, F. Mehta, and L. Voisin. Rodin: an open toolset for modelling and reasoning in Event-B. *STTT*, 12(6):447–466, 2010.

[3] IMPEX Consortium. Formal models for ontologies, June 2016.

[4] W. OWL Working Group. *OWL 2 Web Ontology Language: Document Overview*. W3C Recommendation, 27 October 2009. Available at `http://www.w3.org/TR/owl2-overview/`.

[5] G. Pierra. Context-explication in conceptual ontologies: the plib approach. In *Proceedings of the 10th ISPE International Conference on Concurrent Engineering (CE 2003), Vol. Enhanced Interoperable Systems*, volume 26, page 2003, 2003.