Deliverable T1.2_D1.2 Event B encoding formal theories for ontologies

June 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

This report summarizes the work done in the context of the IMPEX project. It describes formal approaches for modeling ontologies using set theory and predicate logic supported by the Even-B method. In particular, we show the central role played by Event-B contexts to encode ontologies as theories.

In the context of IMPEX, two modeling approaches are defined. The first one is a shallow approach. It defines a formal approach to encode ontologies directly as Event-B contexts. The second approach uses deep modeling in which first ontology models are defined and then ontologies are defined as specific instances of these generic models.

This report is organized as follows. Chapter 2 describes the methodology used to describe formal models for ontologies. Chapters 3 and 4 are devoted to the description of the two identified approaches for modeling. Shallow modeling is presented in chapter 3, while deep modeling is addressed in chapter 4. Finally, a conclusion together with the description of ongoing work are available in Chapter 5.

Chapter 2

Ontologies as theories

This chapter describes the stepwise methodology we have defined in order to allow model designers to handle explicitly domain knowledge. The proposed approach consists in associating a domain ontology (concepts and associated constraints) to a design model. This association, is handled by an annotation mechanism we have defined. This annotation mechanism entails a loose coupling of the ontology and the annotated models. Indeed, neither modification nor evolution of the design models is required. Moreover, ontologies and models may evolve asynchronously.

2.1 Ontologies as domain theories

In this section, we recall some basic characteristics of ontologies and discuss how ontologies can be seen as theories.

2.1.1 Our approach

Our approach advocates the exploitation of domain knowledge, carried out by ontologies, in design models. We propose a stepwise methodology, composed of four steps, to establish a formal link between these two models. The approach is based on the definition of an annotation mechanism that represents this link. The definition of this mechanism depends on the used modeling languages for both ontologies and design models. Fig. 2.1 shows the overall schema of the approach. Four steps have been identified.

1. **Formalization of domain knowledge.** Domain information are formalized in an ontology modeling language. Concepts, entities, relationships, constraints, rules, etc. are explicitly defined. The result is a formal ontology expressed in the chosen ontology modeling language. The semantics of this language and the associated verification techniques are used to establish properties of the ontology. The expressive power of this language has an impact on the defined ontologies (e.g. different constraint description languages may be used). Finally, the ontology shall be defined independently of any context of use. It may also be built from already existing ontologies (e.g. standard ontologies).

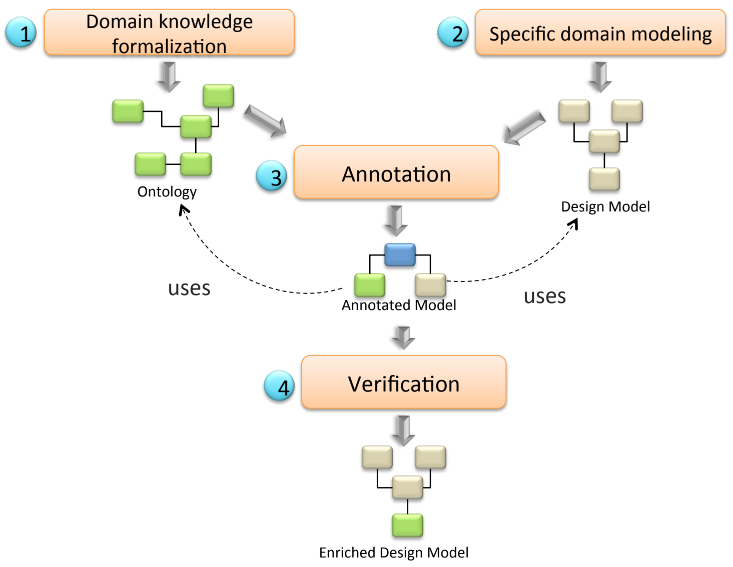

Figure 2.1: A four steps methodology for integrating domain knowledge and design models.

2. **Definition of design models.** Any formal modeling language is used to describe design models. Within this formal modeling language, users define and formalize specific design models corresponding to a given specification. Different analyses allowed by the modeling language and its associated verification technique may be performed on the designed model.

3. **Annotation of design models by references to ontologies.** Using specific mechanisms available in the formal modeling language, annotation of design models are explicitly described. Annotation consists of defining specific relationships between design models entities and ontology concepts. Different relationships are available, they have their specific properties. For example, the *Is_a* relationship can be used to assert that a given design model entity *is* an ontology concept (annotation by *subsumption*). Annotation is made explicit in the design models thanks to the use of these relationships.

4. **Expression and verification of properties.** Once design models are annotated by domain ontologies, the proof context of the design models is enriched by the domain properties expressed in the ontology. It becomes possible to check, on the one hand, if the design models already established before annotation (they may be no longer correct after annotation) are still consistent and, on the other hand, the other properties that emerged after annotation

(issued from the axiomatization of the domain).

At the end of this process, a new design model enriched with new information of the domain knowledge is obtained. This model makes an explicit representation of domain concepts and properties borrowed from the ontology thanks to the annotation.

2.1.2 Some Comments

1. It is important that the specified and used ontologies are defined in a consensual manner by the stakeholders involved in the system under design. Moreover, they should have relationships with the domain of the design model.

2. Steps 1 and 2 of the previous methodology, introduced in section 2.1.1, are independent. They may be run in parallel. Two situations may occur. Ontologies may be defined prior to the design model or they may preexist.

3. In the semantic web area, lot of efforts are devoted to the definition of automatic annotation mechanisms [3, 5, 6, 7, 8, 4]. There, the annotated models are documents in general and ontologies usually exploit terms rather than concepts. The definition of the annotations may be realized either by manual, semi-automatic or automatic processes[11, 12, 2, 14, 1]. In this report, we are concerned with formal design models targeting system design. Therefore, our approach relies on an interactive annotation performed by the designer.

4. The situation where a design model is an extension (by specialization of the ontology or the design model is subsumed by the ontology) of the defined ontology may occur. This situation preserves the ontological reasoning thanks to subsumption. It occurs when the design model is an extension of the domain ontology. In this case, reasoning by subsumption is possible. However, other situations may occur. For example in case the specific design model is cross to several domain ontologies (i.e. multi-domain model) and uses its own concepts, such a specialization is not striaghtforward. More complex mappings (e.g. algebraic expressions, or other structural relationships) are required. The reasoning capabilities offered by subsumption may be lost and more complex reasoning mechanims may be needed.

2.1.3 Associated theory

According to the defined stepwise methodology, defined in previous sections, to make explicit domain knowledge expressed by ontologies in design models, we propose a general formal setting in which such a methodology can be deployed for specific formal methods.

As we do not address heterogeneous semantics, the rest of this report considers that the same satisfaction $\models$ and entailment $\vdash$ relationships are used for both of the ontology modeling language and for the design modeling language.

1. **Formalization of Domain Knowledge.** An ontology is described with an ontology modeling language. It defines axioms $A_{O_1}, \ldots, A_{O_m}$ and proof deduction rules from which properties i.e. theorems $T_{O_1}, \ldots, T_{O_n}$ may be deduced.

   - Ontologies shall be sound (healthiness of the ontology). This means that there exists a model M_O that satisfies the axioms of the ontology. We write $M_O \models A_{O_i}$ for $1 \leq i \leq m$
   - Each theorem can be deduced from the axioms and the other theorems. We write $A_{O_1}, \ldots, A_{O_m} \vdash T_{O_1}$ and $A_{O_1}, \ldots, A_{O_m}, T_{O_1}, \ldots T_{O_{i-1}} \vdash T_{O_i}$ for all $2 \leq i \leq n$

2. **Definition of design models.** The studied systems are described in the chosen modeling language. If the modeling language supports properties verification, then properties may be expressed and checked. Axioms $A_1, \ldots A_k$ and theorems $T_1, \ldots T_l$ describing the model properties are defined.

   - Described system models shall be sound (healthiness of the design model). This means that there exists a model M_D that satisfies the axioms defined for the system model. We write $M_D \models A_i$ for $1 \leq i \leq k$
   - Each theorem expressing properties on the design model can be deduced from the axioms and the other demonstrated theorems. We write $A_1, \ldots, A_k \vdash T_1$ and $A_1, \ldots, A_m, T_1, \ldots T_{i-1} \vdash T_i$ for all $2 \leq i \leq l$

3. **Annotation of design model by references to ontologies.** Annotation consists in integrating domain knowledge expressed by ontologies in the design model.

   - Integrated axioms define a sound annotation (healthiness of the annotated model). There exists a model M satisfying the axioms of both the ontology and the design model. We write $M \models A_1 \wedge \ldots \wedge A_k \wedge A_{O_1} \wedge \ldots \wedge A_{O_m}$.

4. **Expression and verification of properties.** The properties $T_1, \ldots, T_l$ shall be re-proved again once the model has been enriched by ontologies. Moreover, new emerging properties $P_1, \ldots, P_t$ may be inferred from the annotated model.

   - The properties of the design model before annotation need to be re-proved again. Indeed, the ontology may have brought relevant information that falsify 0 or more properties. We write $A_1, \ldots, A_k, A_{O_1}, \ldots, A_{O_m} \vdash T_1$ and $A_1, \ldots, A_k, A_{O_1}, \ldots, A_{O_m}, T_{O_1}, \ldots T_{O_n}, T_1, \ldots T_{i-1} \vdash T_i$ for all $2 \leq i \leq l$
   - When domain knowledge described by ontologies is embedded in the design models, new properties $P_1, \ldots P_t$ may arise. They should be proved. We write: $A_1, \ldots, A_m, A_{O_1}, \ldots, A_{O_m}, \vdash P_i$ for all $0 \leq i \leq t$

Remark. We have assumed that the same deduction logic (with $\vdash$ and $\models$) is associated to both the ontology and the design models. If this is not the case, alignment of the semantics of the ontologies and of the models should be performed. This is out of the scope of this report.

2.2 Formalizing ontologies: Two approaches

In the context of the IMPEX project, we have identified two mechanisms to define ontologies as formal theories. These two approaches use two different modeling approaches: shallow and deep modeling.

2.2.1 Shallow modeling: Ontologies as contexts

The approach that uses shallow modeling consists in defining the ontology concepts directly in the target modeling language. Traces of the structure of the ontology modeling language concepts are no longer available in the obtained model of the ontologies.

Here, ontologies are defined as theories, more precisely as Event-B contexts. The approach consists in encoding the ontology concepts directly in an Event-B context. Chapter 3 of this report gives the details of this approach.

2.2.2 Deep modeling: Ontologies as instances of ontology models

The approach that uses deep modeling consists in defining the ontology concepts together with the concepts of the modeling language that were used to define the ontology concepts.

Here, ontologies are defined as instances of ontology models. Two steps are required. First, an ontology model is formalized and then ontologies are defined as specific models corresponding to the defined ontology model. This approach is described in chapter 4.

Chapter 3

Encoding ontologies as Event-B contexts

In this chapter, we address the formal modeling of ontologies into a formal method consisting of the Event-B method. This formal modeling is based on existing ontologies models expressed into languages such as OWL, which are transformed into Event-B specification (context) by means of transformations rules. The transformation rules describe the corresponding sets, constants, axioms to each ontological construct.

3.1 An ontology as a context

The integration of such ontologies into a formal development process requires describing these ontologies into a formal language supporting the expression of ontology semantics and allowing the validation of significant properties. One way to integrate these ontologies into a specific formal method development process is to express the ontologies languages constructs into the target formal language by means of transformation rules.

In this chapter, we present the transformation rules developed for the Event-B method based on the OWL ontology language. These transformation rules develop for each OWL ontology the corresponding Event-B context expressing ontology concepts and validating reasoning rules. These transformation rules are described for the ontological canonical and non-canonical concepts identified in [9].

3.1.1 Canonical concepts

The canonical concepts : class, property, type and instance are the primitive concepts for the description of an ontology. They are combined to the Is-a subsumption relation in order to define inheritance hierarchies among concepts. The Ontology canonical concepts are described into an Event-B context using : constants, sets and axioms.

Classes

A class description consists of defining its name. Each class is implicitly a subclass of the root class Thing.

- XML-OWL description of a class.

  The OWL description of a class is given by the XML description below.

```
<owl:Class rdf:ID="ClassName">
 </owl:Class>
```

- Event-B formalization of a class.

  In Event-B, we have modeled a class as a subset of the set Thing (carrier set corresponding to the root class Thing).

SETS
$Thing$
CONSTANTS
$ClassName$
AXIOMS
$axm1 : ClassName \subseteq Thing$

Types

A type consists of a set of literals used in datatype properties range definition.

- XML-OWL description of a type.

  The OWL description of a type is given by the XML description below.

```
<owl:Datatype rdf:ID="TypeName"> </owl:Datatype>
```

- Event-B formalization of a type.

  In Event-B, a type is modeled as an abstract carrier set populated with the literals defined as members of the type.

SETS
$TypeName$

Object and datatype properties

A property is defined as a relation from the property domain (class) to the property range (class/type). If the property range is a class it is called object property and if it is a datatype it is called datatype property.

- XML-OWL description of an object property.

  The OWL description of an object property is given by the XML description below.

```
<owl:ObjectProperty rdf:about="PropertyName">
       <rdfs:domain rdf:resource="ClassName1"/>
       <rdfs:range rdf:resource="ClassName2"/>
   </owl:ObjectProperty>
```

- Event-B formalization of an object property.

  In Event-B, an object property is modeled as a relation between the two classes defining the property (domain and range).

```
CONSTANTS
ClassName1
ClassName2
PropertyName
AXIOMS
axm1 : PropertyName ∈ ClassName1 ↔ ClassName2
...
```

- XML-OWL description of a datatype property.

  The OWL description of a property is given by the XML description below.

```
<owl:DatatypeProperty rdf:about="PropertyName">
       <rdfs:domain rdf:resource="ClassName"/>
       <rdfs:range rdf:resource="TypeName"/>
   </owl:DatatypeProperty>
```

- Event-B formalization of a datatype property.

  In Event-B, a property is modeled as a relation between property domain (classe) and property range (type).

```
SETS
TypeName
CONSTANTS
ClassName
PropertyName
AXIOMS
axm1 : PropertyName ∈ ClassName ↔ TypeName
...
```

Instances

An instance consists of a member of a class.

- XML-OWL description of a instance.

  The OWL description of a instance is given by the XML description below.

```
<ClassAssertion>
       <Class rdf:about="ClassName"/>
       <NamedIndividual rdf:about="InstanceName"/>
   </ClassAssertion>
```

- Event-B formalization of a instance.

  In Event-B, a class instance is defined as a member of the set defining the class.

```
CONSTANTS
ClassName
InstanceName
AXIOMS
axm1 : InstanceName ∈ ClassName
...
```

The Is_a relationship

The definition of an Is_a relationship between two classes states that the first class (subclass) is a subclass of the second one (mother class). The Is_a relationship is a fundamental relationship that construct a class inheritance hierarchy.

- XML-OWL description of the Is_A relationship.

  The OWL description of the Is_a relationship is given by the XML description below.

```
<owl:Class rdf:ID="SubClass">
    <rdfs:subClassOf rdf:resource="MotherClass" />
</owl:Class>
```

- Event-B formalization of the Is_a relationship.

  In Event-B, the Is_a relationship is defined as a set inclusion relationship between the corresponding sets to the subclass and the mother class.

```
CONSTANTS
SubClass
MotherClass
AXIOMS
axm1 : SubClass ⊆ MotherClass
...
```

3.1.2 Non-canonical concepts

In addition to canonical concepts, ontology languages offer other concepts in order to express more elaborate ontological constructs. These concepts derived from canonical ones are called non-canonical concepts, they consists of: equivalence, restriction and union.

Equivalence

The equivalence relationship between two classes states that the two classes have precisely the same instances.

- **XML-OWL description of the Equivalence relationship.**

  The following XML description gives the OWL definition of the equivalence relationship between two classes.

```
<owl:Class rdf:about="Class">
   <equivalentClass rdf:resource="EquivalentClass"/>
</owl:Class>
```

- **Event-B formalization of the Equivalence relationship.**

  The equivalence relationship is defined in Event-B using the set equality relationship between the corresponding sets to the equivalent classes.

```
CONSTANTS
subClass
EquivalentClass
AXIOMS
axm1 : subClass ∈ Thing
axm2 : EquivalentClass ∈ Thing
axm3 : subClass = EquivalentClass
```

Restriction

A restriction is a subclass of a mother class. It contains all instances that satisfy a given condition on an object or datatype property, defined on the mother class.

- **XML-OWL description of the restriction operator.**

  The restriction operator considers a class and property and defines a restricted population for a class. The following XML description gives the OWL definition of an allValuesFrom restriction.

```
<owl:Restriction>
<owl:onProperty rdf:resource="Property" />
<owl:allValuesFrom rdf:resource="Class"  />
</owl:Restriction>
```

  The allValuesFrom restriction operator defines a subclass of the property domain whose members have their relational image exclusively in Class.

- **Event-B formalization of the restriction operator.**

  The allValuesFrom restriction operator is defined in Event-B by set difference between the relational antecedent of the property on Class set (members of the property domain whose members have their relational image in Class set) and the relational antecedent of the property on the complement of Class set (members of the property domain whose members have not their relational image in Class set). We define the relational antecedent of Class set using the relational image of the inverse relation of Property.

CONSTANTS
$Class$
$Property$
$Restriction$
AXIOMS
$axm1 : Restriction = Property \sim [Class] \setminus Property \sim [ran(Property) \setminus Class]$
...

Union

The union concept is a binary combination operator, it builds from two classes a union class containing all individuals that are instances of at least one of the two classes.

- **XML-OWL description of the union operator.**

  The XML-OWL definition for the union is defined on two classes as follows.

```
<owl:Class rdf:ID="UnionClass">
  <owl:unionOf rdf:parseType="Collection">
    <owl:Class rdf:about="Class1" />
    <owl:Class rdf:about="Class2" />
  </owl:unionOf>
</owl:Class>
```

- **Event-B formalization of the union operator.**

  The union combination of two classes is modeled in Event-B as the set union of the two sets corresponding to the two classes.

```
CONSTANTS
Class1
Class2
UnionClass
AXIOMS
axm1 : UnionClass ⊆ Thing
axm2 : UnionClass = Class1 ∪ Class2
...
```

3.2 An example of ontologies

The section below presents generic transformation rules that maps each ontological construct to the Event-B code that models it. Applying these transformation rules, allows to generate the corresponding Event-B context to each specific ontology.

3.2.1 Ontology for diplomas: *is_a* and equivalence

Below is presented an extract of the diplomas ontology formalized using the developed transformation rules into an Event-B context. This extract illustrates the use of the subsumption (Is-a) and equivalence relationships.

```
CONTEXT Ontology
SETS
Thing
CONSTANTS
Phd
Master
Engineer
Diploms
Bachelor
AXIOMS
axm1 : Diploms ⊆ Thing
axm2 : Phd ⊆ Diploms
axm3 : Master ⊆ Diploms
axm4 : Engineer ⊆ Diploms
axm5 : Bachelor ⊆ Diploms
axm6 : Engineer = Master
...
END
```

The above extract describes diplomas ontology, it defines the global Diplomas class as subclass of the root ontology class Thing, and the classes Phd, Master, Engineer and Bachelor as subclasses of the class Diplomas in the axioms: axm1, axm2, axm3 and axm4. The equivalence relation between Engineer and Master diplomas is defined by means of an equality relationship between the sets Engineer and Master in axm6.

3.2.2 Ontology for diplomas: use of the union operator

The second illustration of our transformation rules is shown in the extract below. In addition to the diplomas class hierarchy defined in the first example, it models the non canonical union concept.

```
CONTEXT Ontology
SETS
Thing
CONSTANTS
Diploms
Phd
Master
Engineer
Diplomas_For_Phd
Bachelor
Master_UPS
Engineer_N7
AXIOMS
axm1 : Diploms ⊆ Thing
axm2 : Phd ⊆ Diploms
axm3 : Master ⊆ Diploms
axm4 : Engineer ⊆ Diploms
axm5 : Bachelor ⊆ Diploms
axm6 : Diplomas_For_Phd ⊆ Diploms
axm7 : Master_UPS ∈ Master
axm8 : Engineer_N7 ∈ Engineer
axm9 : Diplomas_For_Phd = (Engineer ∪ Master)
...
END
```

The union operator is used to model the Diplomas_For_Phd class which gathers diplomas allowing to prepare a PhD diploma: Master and Engineer. The Diplomas_For_Phd class is modeled using a set Diplomas_For_Phd defined by the union set of the Master and Engineer sets (see axm9).

The Master and the Engineer classes contain respectively the Master_UPS and the Engineer_N7 diplomas, this is modeled respectively in the axioms (axm7 et axm8). The resulting Diplomas_For_Phd set definition contains the two instances Master_UPS and Engineer_N7, which is in adequacy with the Diplomas_For_Phd class definition.

3.2.3 Ontology for diplomas: use of the restriction operator

The third illustration of our transformation rules is presented in the extract below on diplomas ontology. It concerns the restriction operator.

CONTEXT *Ontology*
SETS
Thing
CONSTANTS
Phd
Master
Engineer
Diploms
Bachelor
SI
ENSEEIHT
Level
hasLevel
*RestrictionEngineer*5
AXIOMS
$axm1 : Diploms \subseteq Thing$
$axm2 : Phd \subseteq Diploms$
$axm3 : Master \subseteq Diploms$
$axm4 : Engineer \subseteq Diploms$
$axm5 : Bachelor \subseteq Diploms$
$axm6 : SI \in Master$
$axm7 : ENSEEIHT \in Engineer$
$axm8 : Level \subseteq \mathbb{N}\nVdash$
$axm9 : 5 \in Level$
$axm10 : hasLevel \in Diploms \leftrightarrow Level$
$axm13 : SI \mapsto 5 \in hasLevel$
$axm12 : ENSEEIHT \mapsto 5 \in hasLevel$
$axm11 : RestrictionEngineer5 = hasLevel \rhd 5$
...
END

The restriction is introduced in the diplomas ontology in order to define engineer students, whose diploma's level equals to 5. The datatype property hasLevel is defined from the Diploms class into Level: a subset of integers set defining the level of a diploma (see axm8). It is modeled by means of a partial function from Diploms to the Level sets (see axm10). Two instances of the hasLevel property are defined for SI and ENSEEIHT. Finally the restriction is modeled by the RestrictionEngineer5 set, it defines a range restriction of the hasLevel function on the value 5 (see axm11).

3.3 Deduction rules and theorems

Commonly, defined ontologies are submitted to reasoners in order to detect ontology inconsistencies, answer queries over ontology classes and instances and define classification by the inference of a subsumption hierarchy for the classes described in the ontology. These reasoners infer logical consequences from a set of asserted facts or axioms using inferring (deduction) rules. They use first-order predicate logic to perform reasoning; inference commonly proceeds by forward and backward chaining.

Accordingly, the correctness of the reasoners inferring rules is crucial for the correctness of the obtained ontology after performing reasoning.

One advantage of using formal methods to model ontology is to validate within this formal framework these inferring rules. Indeed, inferring rules can be modeled by theorems to be proven to guarantee their correctness.

We give below an example of such inferring rules, a subsumption deduction rule used for classification purposes.

$$\boxed{(C1 : Restriction(P, E1) \wedge C2 : Restriction(P, E2) \wedge E1 \Rightarrow E2) \Rightarrow C1\ Is\_A\ C2}$$

The deduction rule is related to property restrictions, it means that for two P property restrictions C1 and C2 on two classes E1 and E2 with an inclusion relation between E1 and E2, there is a subsumption relation between C1 and C2.

The above deduction rule is modeled by a theorem to integrate and to prove in the Event-B context modeling the ontology. The extract below show the axioms and the theorem describing the restriction deduction rule.

AXIOMS
$axm1 : E1 \subseteq E$
$axm2 : E2 \subseteq E$
$axm3 : E1 \subseteq E2$
$axm4 : P \in C \leftrightarrow E$
$axm5 : C1 = P \rhd E1$
$axm6 : C2 = P \rhd E2$
$thm1 : C1 \subseteq C2$
$\dots$

The P property is defined as an object property from C to E using relation definition (axm4), the inclusion relation between the sets E1 and E2 is described in axm3, the restrictions on E1 and E2 are defined respectively in axm5 and axm6 by means of range restrictions. Finally, the deduction rule is defined by thm1 that express the Is-A relationship between C1 et C2 using inclusion sets.

The *thm*1 theorem is proved into the Rodin platform, this guarantee that the corresponding deduction rule is valid.

Chapter 4

Encoding ontologies as instances of Generic ontology models

In this chapter, we report the work achieved in the IMPEX project on formal modeling of ontologies using a deep modeling approach.

Here, we consider that both ontology modeling concepts and ontologies are explicitly modeled. We have used the Event-B method to formalize these concepts. More precisely, for defining ontologies as theories, we have used Event-B contexts as support for formalization.

4.1 An ontology model as a context

In the first report of the IMPEX consortium [9], we have identified that specific modeling languages support the definition of ontologies. The availability of such languages means that specific semantic constructs are associated to these languages. Therefore, the need to explicitly represent such semantic constructs when reasoning formally appears on ontologies. One way to model such ontology constructs is to use the constructs offered by Event-B language to explicitly model the ontology modeling language constructs.

In this chapter, we show how such constructs are modeled as Event-B contexts and then how specific ontologies can be described. We also show how reasoning rules available in ontology reasoners can be formally expressed and proved.

According to the characteristics of the modeling languages reported in [9], we have described formal Event-B contexts for canonical concepts and for non-canonical ones. We have collected the main constructs available in the OWL[13] and Plib[10].

4.1.1 Canonical concepts

Canonical concepts represent the core concepts of an ontology. Among these concepts, we find classes, properties, types and instances. The subsumption relationship offers the capability to define concepts hierarchies.

Formalization of basic concepts

The basic resources are described in an Event-B context. Sets and axioms are used to describe the following concepts

- classes, properties, instances and values are defined by the $CLASS$, $PROPERTY$, $INSTANCE$ and $VALUE$ carrier sets. These sets are abstractly defined, they will be populated when defining specific ontologies;
- to associate properties to classes, the $HAS\_PROPERTIES$ relationship associating classes and properties is defined;
- instances are attached to classes by the $HAS\_INSTANCES$ relationship that relates classes to sets of instances;
- within an instance, a property may have a value through the HAS_VALUES relationship which associates a value to a pair $(instance, property)$

```
Context Ontology_Model

Sets
   CLASS,
   PROPERTY,
   INSTANCE,
   VALUES,
   ...
Axioms
   HAS_PROPERTIES = CLASS ↔ PROPERTY

   HAS_INSTANCES = CLASS ↔ INSTANCE

   HAS_VALUES = INSTANCE × PROPERTY ↔ VALUES

    ...
```

Observe that the previous context defines through relations the whole basic and canonical concepts available in an ontology modeling language. Sets represent the unique carrier to describe such concepts. Finally, the set of axioms of the $AXIOMS$ clause is completed by the relevant properties and definitions associated to the defined sets and relations.

Other concepts are defined in this context, we did not give the whole definition of this context.

The Is_a relationship

The Is_a relationship is a fundamental relation that links the classes together in a class hierarchy. More precisely, it defines the subsumption relationship provided by the inheritance relation classically defined in object oriented languages.

- XML-OWL description of the Is_a relationship.

  The OWL description of the Is_a relationship is given by the XML description below.

```
<owl:Class rdf:ID="SubClass">
     <rdfs:subClassOf rdf:resource="MotherClass" />
</owl:Class>
```

- Event-B formalization of the Is_a relationship.

  In Event-B, we have modeled Is_a as a relation between classes. A set IS_A gathers the possible Is_a relations between classses. A second part of this definition describes the constraints associated to inheritance i.e. inclusion of sets of instances. Indeed, in $axm1$ it is explicitly stated that the set of instances of a class x such that $x\ Is\_a\ y$ is included in the set of instances of class y.

```
Axioms

axm1: IS_A ={IsA | IsA ∈ CLASS ↔ CLASS ∧

        (∀ x, y·(x ∈ CLASS ∧ y ∈ CLASS ∧ x↦y ∈ IsA
        ⇔
        union({r· r ∈ HAS_INSTANCES| ran({x}◁r)})
        ⊆ union({r· r ∈ HAS_INSTANCES|
        ran({y}◁r)}) ))
        }
```

  Note that this definition of the subsumption defines the subsumption of classical inheritance. Another subsumption relationship corresponding to the Is_Case_Of relationship borrowed for the Plib ontology model is also defined. It considers that not all the propoerties of a given class are valued when a class instance is defined.

4.1.2 Non-canonical concepts

Once the canonical concepts are defined, ontology modeling languages available in the literature like OWL or Plib offer the possibility to define other ontological concepts from the canonical ones by derivation. These concepts are defined as non-canonical concepts.

To define such non canonical concepts, ontology languages offer various derivation operators. Below, we review the main operators studied in the context of the IMPEX project.

Equivalence

- **XML-OWL description of the Equivalence relationship.**

  The following XML description gives the OWL definition of the equivalence relationship between two classes.

```
<owl:Class rdf:about="Class">
   <equivalentClass rdf:resource="EquivalentClass"/>
</owl:Class>
```

  The semantic of this relationship needs to be precised.

- **Event-B formalization of the Equivalence relationship.**

  To model the equivalence relationship, we proceed in the same manner as for the Is_a relationship. First, the equivalence is a relation between classes $CLASS \leftrightarrow CLASS$. Second, a set of relevant axioms state that the defined relation is reflexive, symmetric and transitive.

```
Axioms
  axm1: EQUIVALENCE = { EQo| EQo ∈ CLASS ↔ CLASS
        ∧
        (∀ x, y, hasInstances· (x ∈ CLASS ∧ y ∈ CLASS ∧ hasInstances ∈ HAS_INSTANCES ∧ x↦ y ∈ EQo
            ⇒
        (hasInstances[{x}] = hasInstances[{y}])))
        ∧
        (∀ x · (x ∈ CLASS ⇒ x↦ x ∈ EQo))
        ∧
        (∀ x, y· (x ∈ CLASS ∧ y ∈ CLASS ∧ x↦ y ∈ EQo ⇒ y↦ x ∈ EQo))
        ∧
        (∀ x, y, z· (x ∈ CLASS ∧ y ∈ CLASS ∧ z ∈ CLASS ∧ x↦ y ∈ EQo ∧ y ↦ z ∈ EQo ⇒ x↦ z ∈ EQo))
        }
```

The definition of the equivalence relationship is given at the class level. Other axioms are given on the instances.

Restriction

The restriction operator is applied on a class or on a class expression. It defines a filter on the population (instances) of a class using a logical property.

- **XML-OWL description of the restriction operator.**

  The restriction operator considers a class and a property, and it defines a restricted population for a class.

```
<owl:Restriction>
<owl:onProperty rdf:resource="..." />
<owl:allValuesFrom rdf:resource="..."  />
</owl:Restriction>
```

- **Event-B formalization of the restriction operator.**

  The restriction operator is defined as a filter on the set of instances. The filtering property is defined as a functional parameter on the instances using the $Property\_Verification$ function on instances.

```
Axioms
  axm1: Property_Verification = INSTANCE ⟶ BOOL
  axm2: RESTRICTION = {rest| rest ⊆ INSTANCE ∧
          (∃ p. (p ∈ Property_Verification ∧ rest= p⁻¹[{TRUE}]))
          }
```

  The previous definition states that the obtained restriction is defined as a set of instances whose value, by the function $Property\_Verification$, is equal to $True$. The inverse function is used for this purpose.

Union

The second shown non-canonical operator is union. It defines the binary union class collecting the instances of two classes.

- **XML-OWL description of the union operator.**

  The XML-OWL definition for the union operation is defined on two classes as follows.

```
<owl:Class rdf:ID="UnionClass">
  <owl:unionOf rdf:parseType="Collection">
    <owl:Class rdf:about="Class1" />
    <owl:Class rdf:about="Class2" />
  </owl:unionOf>
</owl:Class>
```

- **Event-B formalization of the union operator.**

  Similar to the restriction operator, the union operator builds explicitly the set of instances as the union of the sets of instances of the two classes.

```
Axioms
  axm1: UNION_OF = {unionOf|
        (unionOf ∈ (ℙ(CLASS) × ℙ(CLASS)↔ CLASS))
        ∧
        (∀ x, y, z·(x ∈ ℙ(CLASS) ∧ y ∈ ℙ(CLASS) ∧ z ∈ CLASS ∧ x↦y↦z ∈ unionOf
        ⇒
        ∀ instance· (instance ∈ INSTANCE
          ⇒
          ∃ hasInstance· (hasInstance ∈ HAS_INSTANCES ⇒
          (∀ n, m· (n ∈ x ∧ m ∈ y ∧ (n↦instance ∈ hasInstance ∨ m↦instance ∈ hasInstance))
          ⇒
          z↦instance ∈ hasInstance))))
        ) }
```

  The $UnionOf$ operator is defined as a relation between sets of classes. The defined logical property states that if an instance belongs to a class x or an instance belongs to a class y then it belongs to the class z belonging to the $UnionOf$ relation.

4.2 An example of ontologies

Once the ontology concepts are defined at a generic level, it is possible to define specific ontologies as instances of the generic concepts. The Event-B context defining the specific ontology is defined as an extension of the previous context defining the ontology modeling concepts.

4.2.1 Ontology for diplomas: *Is_a* and equivalence

Below, we give an extract of the ontology of diplomas. We have formalized this ontology as instances of the generic concepts previously introduced. The defined ontology illustrates the Is_a and equivalence relationships. Two parts compose this definition.

- an axiomatization part defined by the axioms clauses. It defines the extension of the sets corresponding to instances of the concepts defined in the generic context;
- a theorem part, which represents properties to be proved. The theorems guarantee that the defined relations at the instance level conform to their definition at the generic level.

```
Context Diplomas_Ontology
Extends Ontology_MODEL
Axioms
          axm1: partition(CLASS,
                    {Diplom},
                    {Bachelor},
                    {Master},
                    {Engineer},
                    {Phd}
                    )
          axm2: partition(INSTANCE,
                    {SI},
                    {SRLC},
                    {ENSEEIHT},
                    {ISAE}
                    )
          axm3: hasInstances = {
                    Master↦SI, Master↦SRLC,
                    Master↦ENSEEIHT, Master↦ISAE,
                    Engineer↦ENSEEIHT, Engineer↦ISAE,
                    Engineer↦SI, Engineer↦SRLC,
                    Diplom↦ENSEEIHT, Diplom↦SI,
                    Diplom↦SRLC, Diplom↦ISAE
                              }
          axm4: isA = {
                    Bachelor↦Diplom,
                    PhD↦Diplom,
                    Master↦Diplom,
                    Engineer↦Diplom
                              }
          axm5: EQ ={
                    Bachelor↦Bachelor, Master↦Master,
                    Engineer↦Engineer, Phd↦Phd,
                    Master↦Engineer, Engineer↦Master
                              }

// Relevant Theorems
          thm1: hasInstances ∈ HAS_INSTANCES
          thm2: isA ∈ IS_A
          thm3: EQ ∈ EQUIVALENCE
```

The above ontology defines classes for diplomas. Other classes subsumed by the diploma class are defined: Bachelor, Master, Phd. The subsumption is materialized by the definition of the isA set, where $isA \in$ IS_A and an equivalent relation EQ (stating that $Master$ and $Engineer$ diplomas are equivalent) is also defined.

The proof of the theorems guarantees that the defined instances correspond to the generic definitions of sections 4.1.1 and 4.1.2.

4.2.2 Ontology for diplomas: use of the union operator

Similar to the previous example, we show how operators are applied to define non canonical concepts using the same case study of diplomas. Again, we find two parts, one for axioms defining the instances of the concepts, and one for the theorems to guarantee the conformity of the definition.

Note that for this example, we have introduced a set of instances that was not required in the previous example since the defined relations (Is_a and equivalence) do not require the definition of the set of instances.

The definition of the instances of the concepts introduced are given by the axioms. The following Event-B context represents an extract of the definition of the ontology.

```
Context Ontology
Extends Ontology_Relations

Constants
          Bachelor,
          Master,
          Engineer,
          Phd,
          Master_UPS,
          Diplomas_For_Phd,
          Master_UPS,
          Engineer_N7,
          unionOf,
          hasInstances
Axioms
  axm1: partition(CLASS,
                    {Bachelor},
                    {Master},
                    {Engineer},
                    {Phd},
                    {Diplomas_For_Phd}
                    )
  axm2: partition(INSTANCE,
                    {Master_UPS},
                    {Engineer_N7}
                    )
  axm3: hasInstances = {
                    Master↦Master_UPS,
                    Engineer↦Engineer_N7,
                    Diplomas_For_Phd↦Engineer_N7,
                    Diplomas_For_Phd↦Master_UPS
                    }

  axm4: unionOf = {
                    {Master}↦{Engineer} ↦ Diplomas_For_Phd
                    }

// Relevant Theorems

  thm1: hasInstances ∈ HAS_INSTANCES
  thm2: unionOf ∈ UNION_OF
  ....
End
```

The diplomas are modeled as classes. Here canonical concepts are the same as those introduced in the previous example. We have added a non-canonical concept *Diplomas_For_Phd* defined as the union of the students that hold an *engineer* or a *master* diploma.

Two instances are defined in the set of instances: $Master\_UPS$ and $Engineer\_N7$ by axiom 2. Axiom 3 precises that these two instances are instances of the $Master$ and $Engineer$ classes respectively. Moreover, the same axiom states that both $Master\_UPS$ and $Engineer\_N7$ are instances of the non-canonical class *Diplomas_For_Phd*. Finally, axiom4 states that *Diplomas_For_Phd* is the union of the classes $Master$ and $Engineer$.

Proving that *Diplomas_For_Phd* is really the right union at the instance level, requires to prove theorem 2 which guarantees that the instances are really obtained as the union set. In other words, the definition given by axiom 2 is compatible with the definition of the union.

4.2.3 Ontology for diplomas: use of the restriction operator

To illustrate how the restriction operator is applied for a specific ontology, we also use the case of diplomas. We define students that are engineers as being a group of students whose level equals 5. A restriction is used for this purpose.

```
Context Ontology
Extends Ontology_Relations
Constants
        Bachelor, Master, Engineer, PhD,
        Diplom, LMDDiplom, ClassicalDiplom,   // Classes
        SI, SRLC, ENSEEIHT, ISAE, Person
        BachelorStudent, MasterStudent,      // Instances
        level, age, name,               // Properties

        hasInstances, hasValues, LevelEng, RestrictionEngineer5          // Instances of Generic concepts

Axioms

axm1: partition(CLASS,
                {Bachelor},
                {PhD},
                {Master},
                {Engineer},
                {Diplom},
                {LMDDiplom},
                {ClassicalDiplom},
                {Person}
        )
axm2: partition(INSTANCE,
                {SI},
                {SRLC},
                {ENSEEIHT},
                {ISAE},
                {BachelorStudent},
                {MasterStudent}
        )
axm3: partition(PROPERTY,
                {level},
                {age},
                {name}
        )
axm4: hasInstances = {
                Master↦SI,
                Master↦SRLC,
                Engineer↦ENSEEIHT,
                Engineer↦ISAE,
                Diplom↦ENSEEIHT,
                Person↦BachelorStudent,
                Person↦MasterStudent
        }
axm5: hasProperties = {
                Person↦name, Person↦age,
                Master↦level, Engineer↦level,
                PhD↦level, Bachelor↦level
axm6: hasValues = {
                SI↦level↦5,
                ENSEEIHT↦level↦5
        }
axm7: LevelEng = (
                λi. i ∈ INSTANCE ∧ hasInstances⁻¹[{i}]={Engineer} |
                                        bool(hasValues(i↦level)=5))
axm8: RestrictionEngineer5 = LevelEng⁻¹[{TRUE}]

\\Relevant theorems

thm1: hasInstances ∈ HAS_INSTANCES
thm2: hasProperties ∈ HAS_PROPERTIES
```

```
thm3: hasValues ∈ HAS_VALUES
thm4: LevelEng ∈ Property_Verification
thm5: RestrictionEngineer5 ∈ RESTRICTION
...
```

The previous context extends the generic context and defines the instances for classes, properties and instances. It also defines the restriction predicate as a lambda function returning a boolean when the level equals to 5 ($axm6$). This lambda function is instantiated ($axm8$). Then a theorem asserts that this instantiation produces an instance of the $Property\_verification$ parameter of the restriction operator defined at the generic level ($thm4$). Finally, a theorem ($thm5$) states that we have defined a restriction corresponding to the definition of the generic level.

4.3 Deduction rules and theorems

In practice, when ontologies are defined, users set up reasoners in order to infer knowledge from the canonical and non-canonical reasoning. These reasoners apply deduction rules in order to infer these facts or to define classification trees. The application of these rules is performed within reasoning engines.

The definition of these deduction rules corresponds to theorems. The validation of the correctness of these rules is mandatory to guarantee the correctness of the performed reasoning and thus of the reasoners.

The work we have achieved in the context of the IMPEX project by formalizing deduction rules as theorems led us to the capability to prove, in the defined Event-B context generic theorems corresponding to deduction rules used by reasoners.

As example, we consider the theorem used by reasoners to build class hierarchy in order to make explicit the subsumption relationship (placement in the subsumption hierarchy) for non-canonical classes. The following theorem deals with the restriction operation on classes.

$$\boxed{(C1 : \mathit{Restriction}(C, P1) \wedge C2 : \mathit{Restriction}(C, P2) \wedge P1 \Rightarrow P2) \Rightarrow C1\ Is\_a\ C2}$$

The following Event-B theorem is added to the generic ontology concepts context previously defined.

```
thm1: ∀ x, p1, p2· (x ∈ INSTANCE ∧ p1 ∈ Property ∧ p2 ∈ Property
        ∧ (p1(x)=TRUE ⇒ p2(x)=TRUE)
                ⇒
           p1⁻¹[{TRUE}] ⊆ p2⁻¹[{TRUE}]))

thm2: ∀ x, p1, p2· (x ∈ INSTANCE ∧ p1 ∈ Property ∧ p2 ∈ Property ∧ (p1(x) ⇒ p2(x))
                ⇒
(∀ y, z, hasInstances, isA· (y ∈ CLASS ∧ z ∈ CLASS ∧ hasInstances ∈ HAS_INSTANCES
∧ isA ∈ IS_A ∧ hasInstances[{y}] = p1⁻¹[{TRUE}] ∧ hasInstances[{z}] = p2⁻¹[{TRUE}]
                ⇒
          y↦z ∈ isA )))
```

The given theorem is written in two steps. Lemma $thm1$ states that if a two sets of instances satisfy properties $P1$ and $P2$ respectively such that $P1 \Rightarrow P2$ then the former set of instances in included in the later. Theorem $thm2$ defines the suited theorem. Lemma $thm1$ is used to prove it.

Moreover, the proof of this theorem can be used to justify or assert that the rule implemented by the reasoners is valid.

Chapter 5

Conclusion

5.1 Conclusion

The work reported in this document concerns the formalization of domain ontology using set theory and predicate logic. We have chosen the Event-B method as a support of this formalization.

Two modeling approaches have been identified and illustrated through examples.

The first one is based on a shallow modeling where the concepts of ontologies are directly encoded in the target theory. In this case, the information related to the modeling language are not explicitly defined in the target formal modeling language. Reasoning on ontologies is performed using the underlying proof system of Event-B.

The second approach consists in formalizing ontology models within an Event-B context and then ontologies are defined as specific instances of the generic ontology model. The interest of the approach is that the ontology model is explicitly described and reasoning on ontologies is performed using the proof system described in the ontology model and the underlying proof system of Event-B.

In this work, we have show an application to the validation of ontology reasoners through the definition and proof of theorems that correspond to reasoning rules in ontology reasoners.

5.2 Ongoing work

Several extensions, in the area of ontology modeling, arose from the presented work. We are currently working on

1. Studying ontologies as theories. This work consists in defining ontologies and ontology models as Event-B theories. Such a definition will allow ontology designer to define directly ontology modeling concepts as sets or "types" that can be attached to the ontology concepts. Moreover, the theorems can be turned into rules that will be used in the proof process.

2. Designing and implementing a tool for formalizing ontologies. The tool under development consists in producing the Event-B contexts of theories from the ontology definitions provided either as OWL ontologies or PliB ontologies.

3. Formalizing model annotation. This work consists in defining an annotation relationship between ontology concepts and design models concepts. The objective through this annotation is to strengthen design models by borrowing properties from ontologies to the design models.

4. Finally, development of ontologies for industrial domains. We are developing an ontology representative of an industrial domain: the railway domain.

Bibliography

[1] Y. Aït-Ameur, J. P. Gibson, and D. Méry. On implicit and explicit semantics: Integration issues in proof-based development of systems - version to read. In *Leveraging Applications of Formal Methods, Verification and Validation. Specialized Techniques and Applications - 6th International Symposium, ISoLA 2014, Imperial, Corfu, Greece, October 8-11, 2014, Proceedings, Part II*, volume 8803, pages 604–618. Springer Verlag, 2014.

[2] N. Belaid, S. Jean, Y. Aït-Ameur, and J. Rainaud. An ontology and indexation based management of services and workflows application to geological modeling. *IJEBM*, 9(4):296–309, 2011.

[3] K. Bontcheva, V. Tablan, D. Maynard, and H. Cunningham. Evolving GATE to Meet New Challenges in Language Engineering. *Natural Language Engineering*, 10(3/4):349–373, 2004.

[4] A. Chebotko, Y. Deng, S. Lu, F. Fotouhi, and A. Aristar. An ontology-based multimedia annotator for the semantic web of language engineering. *Int. J. Semantic Web Inf. Syst.*, 1(1):50–67, 2005.

[5] H. Cunningham, D. Maynard, K. Bontcheva, V. Tablan, N. Aswani, I. Roberts, G. Gorrell, A. Funk, A. Roberts, D. Damljanovic, T. Heitz, M. A. Greenwood, H. Saggion, J. Petrak, Y. Li, and W. Peters. *Text Processing with GATE (Version 6).* 2011.

[6] S. Desprès and S. Szulman. Terminae method and integration process for legal ontology building. In M. Ali and R. Dapoigny, editors, *Advances in Applied Artificial Intelligence, 19th International Conference on Industrial, Engineering and Other Applications of Applied Intelligent Systems, IEA/AIE 2006, Annecy, France, June 27-30, 2006, Proceedings*, volume 4031 of *Lecture Notes in Computer Science*, pages 1014–1023. Springer, 2006.

[7] S. Handschuh and S. Staab. CREAM: creating metadata for the semantic web. *Computer Networks*, 42(5):579–598, 2003.

[8] S. Handschuh, R. Volz, and S. Staab. Annotation for the deep web. *IEEE Intelligent Systems*, 18(5):42–48, 2003.

[9] IMPEX Consortium. Formal models for ontologies, 2015.

[10] ISO. Industrial automation systems and integration. parts library. part 42: Description methodology: Methodology for structuring parts families. ISO ISO13584-42, International Organization for Standardization, Geneva, Switzerland, 1998.

[11] Y. Lu, H. Panetto, Y. Ni, and X. Gu. Ontology alignment for networked enterprise information system interoperability in supply chain environment. *Int. J. Computer Integrated Manufacturing*, 26(1-2):140–151, 2013.

[12] L. S. Mastella, Y. Aït-Ameur, S. Jean, M. Perrin, and J. Rainaud. Semantic exploitation of engineering models: An application to oilfield models. In A. P. Sexton, editor, *Dataspace: The Final Frontier, 26th British National Conference on Databases, BNCOD 26, Birmingham, UK, July 7-9, 2009. Proceedings*, volume 5588 of *Lecture Notes in Computer Science*, pages 203–207. Springer, 2009.

[13] 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/`.

[14] D. S. Zayas, A. Monceaux, and Y. Aït-Ameur. Knowledge models to reduce the gap between heterogeneous models: Application to aircraft systems engineering. In R. Calinescu, R. F. Paige, and M. Z. Kwiatkowska, editors, *15th IEEE International Conference on Engineering of Complex Computer Systems, ICECCS 2010, Oxford, United Kingdom, 22-26 March 2010*, pages 355–360. IEEE Computer Society, 2010.