Strengthening MDE and Formal Design Models by References to Domain Ontologies. A Model Annotation Based Approach

Kahina Hacid, Yamine Aït-Ameur

Strengthening MDE and Formal Design Models by References to Domain Ontologies. A Model Annotation Based Approach

Kahina Hacid(✉) and Yamine Ait-Ameur

Université de Toulouse, INP, IRIT Institut de Recherche en Informatique de Toulouse, Toulouse, France
{kahina.hacid,yamine}@enseeiht.fr

Abstract. Critical systems are running in heterogeneous domains. This heterogeneity is rarely considered explicitly when describing and validating processes. Handling explicitly such domain knowledge increases design models robustness due to the expression and validation of new properties mined from the domain models. This paper proposes a stepwise approach to enrich design models describing complex information systems with domain knowledge. We use ontologies to model such domain knowledge. Design models are annotated by references to domain ontologies. The resulting annotated models are checked. It becomes possible to verify domain-related properties and obtain strengthened models. The approach is deployed for two design model development approaches: a Model Driven Engineering (MDE) approach and a correct by construction formal modeling one based on refinement and proof using Event-B method. A case study illustrates both approaches (This work is partially supported by the French ANR-IMPEX project.).

Keywords: Domain ontologies · Model annotation · Property verification · MDE · Proof and refinement · Event-B

1 Introduction

As part of the system engineering and complex system design, the models designed by engineers are placed at the center of the development process of the understudied system. Engineers use them to describe, reason, analyse and verify systems operating in different environments, domains and contexts. In addition, these models correspond to partial views of the studied system (e.g. functional, real-time, energy, mechanics, reliability, architecture, etc.). This leads to the production of several heterogeneous models corresponding to the same system which we qualify as *"design models"*.

In this context, the most important heterogeneity factors, in addition to the modeling languages, are those related to information, knowledge and assumptions of the underlying studied domain (environment and context of implementation and execution of designed systems). Domain knowledge information is usually not explicitly handled and therefore not included in the models associated to the

DOI: 10.1007/978-3-319-47166-2_24

systems under design that may be critical systems. In fact, although these models are developed in accordance with the standards and good practices, some knowledge, required for model interpretation and validation, remain implicit. As a consequence, a system may be considered as correct with respect to the initial requirements but, it can miss some of its relevant properties if the information related to its application domain are not black handled by the modeling activity. Therefore, the verification and validation activities are partially covered since domain requirements and constraints are themselves partially included in the designed models. Handling domain knowledge and properties requires the availability of (1) models for such knowledge and properties and (2) of a relationship to link both design models and domain knowledge models. Ontologies are good candidates for describing such domain knowledge. In particular, they are well suited for the characterization of engineering domains.

In order to handle domain knowledge in design models, our approach advocates (1) the use of conceptual ontologies to model and make explicit the domain knowledge and properties. These ontologies are designed from domain concepts. They represent the basic concepts of a domain together with their relationships and properties. Then, (2) we propose a reasoned, based on model annotation, approach to integrate knowledge mined from the studied application domain to the design models.

The objective of this paper is to increase the quality of engineering models by handling new hypotheses and properties entailed by making explicit the engineering knowledge.

Our proposal consists in annotating models by explicit references to ontologies. We propose a four-steps methodology. First, ontologies are used to clarify and formalize domain knowledge concepts, relationships and constraints. Then, the specific defined design models produced from given requirements and specifications are annotated by references to ontologies. These annotations link design models concepts with the corresponding ontology concepts. These ontology concepts offer an explicit semantics to the design model concepts. As a consequence, new design models overloaded by explicit references to ontologies are obtained. It becomes possible to express and verify new domain related properties on the obtained annotated design models.

This paper is structured as follows. Section 2 presents a didactic case study illustrating our approach. Section 3 gives a global definition of domain ontologies. Our approach for strengthening models through an annotation based method is presented in Sect. 4. Sections 5 and 6 give details of the implementation within MDE and a correct-by-construction approaches. Section 7 overviews different approaches promoted for annotation and semantic enrichment of models. A conclusion ends this paper and identifies some research directions.

2 A Case Study

In order to illustrate our proposal, we have chosen a didactic case study describing a simple information system. This information system results from requirements and is described through a set of concepts, actions and constraints as it is the case for applications in the engineering domain. The defined case study deals with the management of students diplomas and registration in the European higher education system. This system offers two kinds of curricula: first the Bachelor (Licence), Master and Phd, LMD for short, and second the Engineer curriculum. Each diploma of the LMD curricula corresponds to a given level: Bachelor/ Licence (high school degree + 6 semesters/180 ECTS credits), Master (Bachelor + 4 semesters/120 credits) and PhD (Master + 180 credits). Engineer curricula offers the engineer diploma five years after high school degree. Both Master and Engineer diplomas are obtained five years after high school degree.

2.1 Additional Requirements for Students Registration

In the studied information system, students register to prepare their next expected diploma. This registration action takes into account the last hold academic degree (or last diploma) as a pre-requisite to register for the next diploma. Constraints on the registration action require that the information system does not allow a student to register for a new diploma if he/she does not have the necessary qualifications. Therefore, the designed information system must check the logical sequence of obtained diplomas before allowing a student to register. For example, Phd degree registration is authorized only if the last obtained degree corresponds to a Master degree. The studied information system **prescribes** the necessary conditions for registering students for preparing diplomas.

2.2 The Domain Knowledge for Diplomas

Diplomas and their characteristics represent a central knowledge for the previously defined case study (but also for other possible applications). A knowledge model to **describe** the diploma knowledge through diplomas characteristics, rules and constraints can be defined as an ontology. This ontology shall model the whole concepts, properties and constraints associated to the **description** of diplomas. It shall cover the diplomas descriptions beyond their usage in the previously described information system, independently of any context of use. Several candidate ontologies are possible. We shall use a consensual ontology for this purpose. Reaching consensual agreements is out of scope of this paper. This activity is usually carried out by standards or users communities.

3 Domain Ontologies as Models for Domain Knowledge

Gruber defines an ontology as *an explicit specification of a conceptualization* [1]. In our work, a domain ontology is considered as a *formal and consensual dictionary of categories and properties of entities of a domain and the relationships that hold among them* [2]. In this definition, entity represents any concept belonging to the studied domain. The term dictionary emphasizes that any entity and any kind of domain relationship described in the domain ontology may be referenced directly by a symbol (URI i.e. unique resource identifier). This referencing mechanism is the ground model for the annotation process. An ontology modeling language is required to describe such ontologies. Several ontology modeling languages have been developed so far. OWL[1] [3], PLIB [4,5], RDFS [6] are some examples of such languages. They describe ontology entities using different modeling artifacts like class hierarchies, properties, relationships, instances and individuals, constraints, etc. According to [2], a domain ontology is a domain conceptualization that shall obey to the three fundamental criteria: being formal, consensual and offering references capabilities.

1. Formal. An ontology is a conceptualization based on a formal theory which allows to check consistency properties and to perform some automatic reasoning over the ontology-defined concepts and individuals.
2. Consensual. An ontology is a conceptualization agreed upon by a community larger than the members involved in one particular application development (one design model). Ontology standards are good supports for such agreements.
3. Capability to be Referenced. Each ontological concept is associated with an identifier or URI. References to this concept becomes possible, using this identifier, from any environment, independently of the particular ontology where this concept was defined.

One other important characteristic is related to the design process. In the case of the engineering domain, ontologies are built from canonical (primitive) concepts, then non canonical (derived) concepts are defined from canonical ones by composition of derivation operators (restriction, union, intersection, algebraic operators, etc.) available in the ontology modeling language. Note that terms are associated to each concept. In this paper, we do not address the ontology design process, we suppose that ontologies already exist.

This section is voluntarily made concise. The literature related to ontology engineering is full of definitions, approaches, work, tools, applications etc. In this section we just reviewed some basic definitions and characteristics of ontologies that are relevant to set up our proposal described in the remaining sections.

4 Strengthening Design Models Using Domain Models: An Annotation Based Approach

The work presented in this paper addresses the case of design models described in an engineering context, where structured models are designed within specific design languages being either semi-formal or formal modeling languages. By strengthening design models, we mean enriching these models with relevant properties mined from domain knowledge models expressed by ontologies. Annotation based techniques are set up in order to link design models to ontologies. A four-steps methodology is proposed for this purpose.

4.1 A Stepwise Methodology

Our approach advocates the exploitation of domain knowledge, carried out by conceptual ontologies, in design models. This approach is stepwise, it is made of different steps. Figure 1 shows the overall schema of the approach.

[1] http://www.w3.org/2001/sw/wiki/OWL.

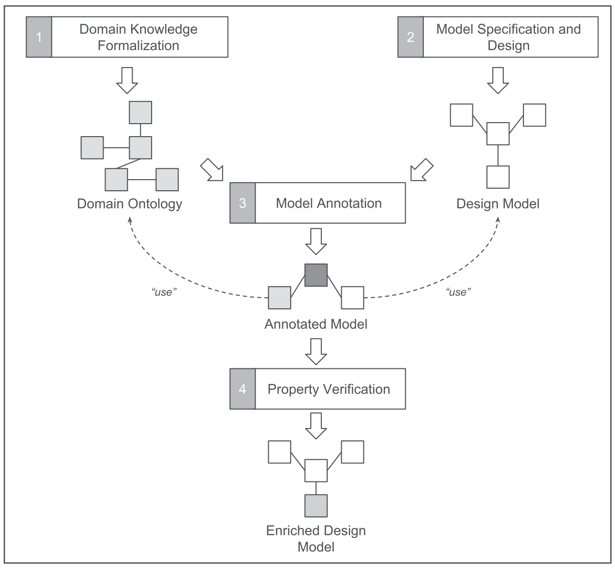

Fig. 1. A four steps methodology for handling domain knowledge in models.

1. Domain Knowledge Formalization. This step consists in making explicit the domain knowledge with a formalized knowledge model. So, information of the domain (concepts, links between these concepts, properties or these concepts and rules and constraints) are explicitly described in a knowledge model.

Formal ontologies are used for this purpose. The choice of this modelling language depends on the kind of reasoning to be performed. Note that this ontology shall be described independently of any context of use. It may also be built from existing ontologies (e.g. standard ontologies).

2. Model Specification and Design. Specific design models corresponding to a given specification are defined. They are formalized within a specific modelling language supporting different analysis, classically performed at the design modelling level.

3. Model Annotation. In this step, the relationships between design model entities and the corresponding knowledge concepts are made explicit. They correspond to model annotation and these relationships are themselves described with a modelling language.

4. Properties Verification. The annotated model obtained at the previous step is enriched by domain properties borrowed from the ontology. The annotated model is analysed to determine whether, on the one hand, the properties and/or the constraints expressed on the annotated model are still valid and, on the other, new properties entailed by the annotation are valid.

Finally, a new design model enriched with new domain information is obtained. Verification and validation of this model (step 4) are required to check if the former properties and/or domain ones, resulting from annotation still hold.

4.2 Some Remarks

The languages used to model ontologies, design models and annotation relationships may differ, semantic alignment between these modeling languages may be required. This topic is out of the scope of this paper, we consider that these languages have the same ground semantics. A single and shared modeling language for the description of both ontologies, design models and annotations is used in this work. Furthermore, the engineering application domain uses modeling languages with classical semantics using closed world assumption (CWA) [7].

The annotation step (step 3) described above requires the definition of annotation mechanisms. Different kinds of annotation mechanisms can be set up (inheritance, partial inheritance and algebraic relationships) [8,9]. The details and choice of the right mechanism are also out of the scope of this paper.

Next two sections show the deployment of this methodology on two different modeling techniques. The first one is based on model driven approaches where constraint checking is performed and the second one is based on a refinement and proof formal modeling technique with the Event-B method.

5 First Deployment: Integration in a Model Based Development

We show below how the proposed stepwise methodology can be deployed in an MDE setting.

5.1 Model Driven Engineering (MDE) Based Developments

MDE brought several significant improvements to the development cycle of complex systems allowing system developers to focus on more abstract levels than classical programming level. MDE is a form of generative engineering [10] in which all or part of an information system is generated from models. A system can be described by several models corresponding to several views or abstraction levels. These models are often described using either graphical or textual notations supported by semi-formal modeling languages. These languages support the description and representation of both structural, descriptive and behavioral aspects of a system. The capability to define constraints that limit the interpretation domain of models is offered using constraints definition languages. In this context, UML [11], the MOF [12] and OCL [13] play the role of standard, they are widely and commonly used by the MDE community.

Moreover, MDE handles models at different development stages of a system development life-cycle. MDE offers several techniques to automate different development steps. Indeed, model operationalization for code generation, documentation and testing, validation, verification, implementation, model analysis are available. These techniques use the capability to transform source models either to other target models in order to get benefits from the available analysis techniques offered by the target modeling technique or to source code in a given language. Transformations are defined by means of transformation rules describing the correspondence between the entities in the source model and those of the target model. This transformation process is automated as much as possible by means of processing programs, which are in most cases developed in general purpose languages (e.g. Java) or in dedicated transformation modeling languages (e.g. ATL[2], Kermeta[3], QVT [14]). In this work, MDE techniques are set up. Meta-models of each manipulated models (ontologies, design models and annotations) are defined in order to build an annotation model in an uniform setting, and ease the prototyping. We have implemented this approach using model

[2] ATLAS Transformation Language: http://www.eclipse.org/atl/.
[3] Kermeta: http://www.kermeta.org/.

driven engineering techniques with the Eclipse modeling Framework (EMF)[4]. The Ecore meta-model being the modeling language.

5.2 Step 1. Domain Knowledge Formalization

The deployment of our methodology requires, in its first step, the availability of an ontology formalizing the domain knowledge. The model of the ontology is designed to integrate all the relevant properties of the domain, including its constraints.

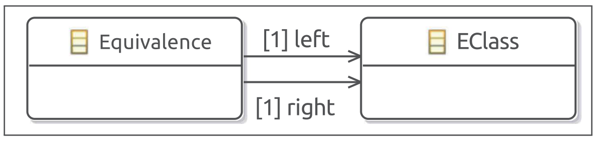

Fig. 2. The *Equivalence* Relationship.

Concepts and properties are modeled as classes and attributes of the ontology and the ontological constraints are added as OCL constraints. The whole ontological relationships like Equivalence, restriction, etc. are also expressed. As illustration, Fig. 2 gives the definition of the equivalence relationship as a class at the meta-modeling level.

$\forall$ x,y,z $\mid$ x$\mapsto$y $\in$ Eq $\wedge$ y$\mapsto$z $\in$ Eq $\Rightarrow$ x$\mapsto$z $\in$ Eq	`EQ_transitivity: Equivalence.allInstances()-> forAll(eq1, eq2| eq1.right = eq2.left implies Equivalence.allInstances()-> exists(eq3|eq2.right = eq3.right and eq1.left = eq3.left));`

Fig. 3. Equivalence relationship: transitivity property expressed OCL.

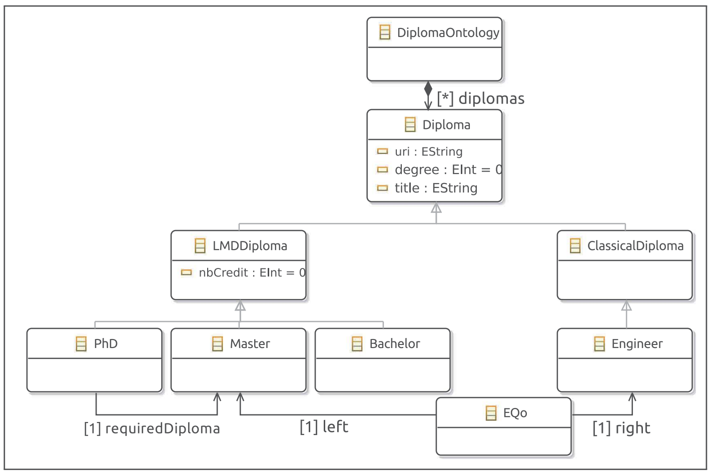

Fig. 4. The *Diplomas* ontology.

The properties related to *symmetry, reflexivity and transitivity* of the equivalence relationship are formalized as OCL constraints. For example, the formalization of transitivity property is given in Fig. 3.

The defined ontology for diplomas is depicted on Fig. 4. Diplomas and their characteristics represent a central knowledge for the previously defined case study (but for other possible applications as well). A model to *describe* the diploma knowledge through diplomas characteristics, rules and constraints defines an ontology. It represents a shared knowledge model that can be used beyond the described application. The defined ontology contains a set of inter-related classes and relevant properties as follows.

[4] Eclipse modeling framework: https://www.eclipse.org/modeling/emf/.

- A subsumption relationship (represented by the *is_a* relationship on Fig. 4) is used to define hierarchies between categories of diplomas. *LMDDiploma* and *ClassicalDiploma* describe respectively the *Bachelor, Master* and *PhD* diplomas and other diplomas (e.g. *Engineer*).
- Several descriptive properties, like *title, degree, uri* of the *Diploma* class describe the name, the uri and the level of a given diploma and *nbCredit* defines the credit number required for each diploma.
- An ontological constraint on the model states that *Master* is equivalent to *Engineer*. It is written in the ontology modeling language as $EQ\_o(Master, Engineer)$ where EQ_o is an instance of the $Equivalent\_Class$ of the ontology meta-model.

In the ontology, this constraint is represented by an *equivalent* class linking the *left: Master* and *right: Engineer* classes of the same ontology. Another constraint defined as *thesisRequirement* carried by the *requiredDiploma* relationship *requiredDiplom_i* property) is added to assert that any master (or any equivalent diploma) is required to prepare a PhD.

5.3 Step 2. Model Specification and Design

The design models are defined by the designer according to a given specification. Several design models corresponding to particular designs for a problem requirement may be produced. The designed models include specific design constraints expressed using OCL [13].

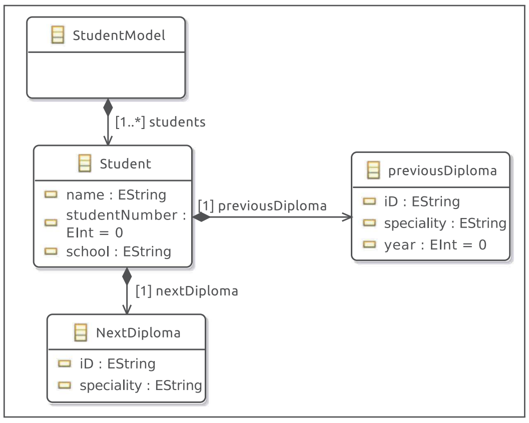

Fig. 5. Engineering student model.

Figure 5 depicts one possible UML class diagram describing a part of the information system related to the management of students. Obviously, other models can be defined as a solution for the problem requirements. In this model, a student holds a diploma (degree) represented by the last graduation diploma he obtained (*previousDiploma* relationship). A student, modeled by the *Student* class with the properties *name*,*studentNumber* and *school*, representing his name, his student number and his school. The *PreviousDiploma* class describes the last diploma hold by a student, with *speciality*, *year* and *iD* properties for the chosen speciality, a year of graduation and the type of diploma (m, e, p and b for master, engineer, Phd and Bachelor respectively). Last, the *NextDiploma* class describes the next diploma a student intends to prepare. Moreover, a constraint named *phdInscription* on the student *nextDiploma* is defined. It asserts that a student registering for a PhD diploma needs to hold a master diploma to be allowed to register for a PhD. It represents a model invariant and it is defined by the OCL constraint of Fig. 6.

Student.nextDiploma.iD = ”p” ⇒ Student.previousDiploma.iD = ”m”	`phdInscritpion: self.NextDiploma.iD = ’p’ implies self.PreviousDiploma.iD = ’m’`

Fig. 6. Formalization of *phdInscritpion* constraint.

5.4 Step 3. Model Annotation

In step 3, relations, defining the defined annotation model, are set up between the design model entities and the ontology concepts. Figure 7 shows how the *PreviousDiploma* class of the students design model is annotated by the *Master* class of the Diplomas ontology.

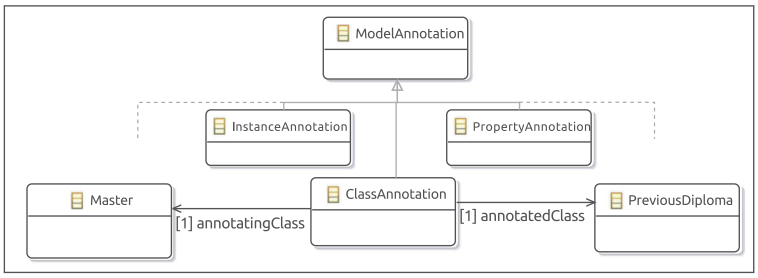

Fig. 7. Annotation of Student model.

Similarly, *NextDiploma* is annotated by *PhD* of the Diplomas ontology. The non-structural property *Equivalent_Class* and the *thesisRequirement* constraint can now be accessed and exploited. So, the equivalence between Master and Engineer classes is expressed and made explicit within the design model.

5.5 Step 4. Properties Verification

The last step analyses the obtained annotated design models through formally established links with the ontology. This annotation process leads to the enrichment of the original design model with new relations, properties, constraints and rules. Ontological properties and classes are considered to be available in the enriched model if they have been selected or linked to model properties during the annotation process (third step of the approach). The new enriched model is validated by (re-)checking all the constraints (the existing and the new added ones) on the model and all its instances. The *equivalence* property and the *thesisRequirement* constraint are now explicit on the annotated student model. The verification process ends with integrating the *equivalence* domain constraint into the enriched design model since all the properties it is relating to are available. At this level, it becomes possible to conclude that a student can apply for preparing a Phd thesis if he holds an engineer diploma. Thus, the *phdInscritpion* constraint is modified to integrate the result of annotation. Its formalization is given in Fig. 8. This property became explicit after handling domain knowledge (by annotation) expressed in the ontology.

Student.nextDiploma.iD="P"
 $\Rightarrow$
 annotation(Student.previousDiploma)
 $\in$ eq(Master)
	`phdInscritpion: self.nextDiploma.iD = 'p' implies`
	 `  let c: ecore::EClass = ClassAnnotation.allInstances()->`
	 `    select(inst|inst.annotatedClass =`
	 `    self.previousDiploma)-> at(0).annotatingClass`
	 `  in  Equivalence.allInstances()-> exists(eq|`
	 `   eq.left.uri = 'Master_uri' and eq.right = c);`

Fig. 8. The OCL constraint *phdInscritpion* after annotation.

6 Second Deployment: Integration in the Event-B Formal Method

In this section, we show how the proposed stepwise methodology can be deployed in the case of formal modeling. The refinement and proof Event-B[5][15] formal method has been chosen for this purpose. It applies the four defined steps (see Sect. 4) and gives the root Event-B models for the case study.

6.1 Event-B: A Refinement and Proof Based Formal Method

The Event-B method [15] is a stepwise formal correct-by-construction development method. It is based on the refinement of an initial model, a machine by gradually adding design decisions. A set of proof obligations (PO), based on the weakest precondition calculus [16], is associated to each machine. Development correctness is guaranteed by proving these PO.

An Event-B model [15] (see Fig. 9(a)) is defined by a*MACHINE*. It encodes a state transitions system which consists of: the variables declared in the

CONTEXT	**MACHINE**
ctxt_id_2	*machine_id_2*
EXTENDS	**REFINES**
ctxt_id_1	*machine_id_1*
SETS	**SEES**
s	*ctxt_id_2*
CONSTANTS	**VARIABLES**
c	v
AXIOMS	**INVARIANTS**
$A(s, c)$	$I(s, cv)$
THEOREMS	**THEOREMS**
$T_c(s, c)$	$T_m(s, c, v)$
END	**VARIANT**
	$V(s, c, v)$
	EVENTS
	Event $evt =$
	any x
	where $G(s, c, v, x)$
	then
	$v :\mid BA(s, c, v, x, v')$
	end
	END

(a) Contexts and machines.

Theorems	$A(s, c) \Rightarrow T_c(s, c)$ $A(s, c) \wedge I(s, c, v) \Rightarrow T_m(s, c, v)$
Invariant preservation	$A(s, c) \wedge I(s, c, v) \wedge G(s, c, v, x)$ $\wedge BA(s, c, v, x, v') \Rightarrow I(s, c, v')$
Event feasibility	$A(s, c) \wedge I(s, c, v) \wedge G(s, c, v, x)$ $\Rightarrow \exists v'.BA(s, c, v, x, v')$
Variant progress	$A(s, c) \wedge I(s, c, v)$ $\wedge G(s, c, v, x) \wedge BA(s, c, v, x, v')$ $\Rightarrow V(s, c, v') < V(s, c, v)$

(b) Generated proof obligations for a model.

Fig. 9. Basic definitions: contexts, machines and proof obligations.

[5] http://www.event-b.org/.

VARIABLES clause to represent the state; and the events declared in the *EVENTS* clause to represent the transitions (defined by a Before-After predicate BA) from one state to another.

The model holds also *INVARIANTS* and *THEOREMS* to represent relevant properties of the defined model. Then a decreasing *VARIANT* may introduce convergence properties when needed. An Event-B machine is related, through the *SEES* clause to a *CONTEXT* which contains the relevant sets, constants axioms, andtheorems needed for defining an Event-B model. The refinement capability [17], introduced by the *REFINES* clause, decomposes a model (thus a transition system) into another transition system with more design decisions while moving from an abstract level to a less abstract one. New variables and new events may be introduced at the refinement level. In a refinement, the invariants shall link the variables of the refined machine with the ones of the refining machine. A gluing invariant is introduced for this purpose. It preserves the proved properties and supports the definition of new ones.

Once an Event-B machine is defined, a set of proof obligations is generated. They are submitted to the embedded prover in the RODIN [15] platform. Here the prime notation is used to denote the value of a variable after an event is triggered. More details on the Event-B method can be found in [15].

6.2 Step 1. Domain Knowledge Formalization

The different concepts describing the main features of an ontology language are defined in an Event-B context. All the basic ontological concepts and relationships are formally described within a general Event-B context. This context can be extended to be specialized for a specific ontology. Listing 1.1 is an extract of the Event-B *Ontology_Relations* context defining relevant finite sets for CLASS, PROPERTIES and INSTANCES. It also defines the basic ontological relationship EQUIVALENCE that may exist between two classes. Other concepts, relations and properties are defined to cover more ontological concepts. They are not given in this paper for space reasons. Note that this context can be extended to be specialized for a specific ontology. The context *Ontology_Relations* (Listing 1.1) is extended by *Diplomas_Ontology* (Listing 1.2). Diplomas are defined as Classes. The equivalences between the different classes are explicitly formalized using the specific equivalence relation *EQo* belonging to the set *EQUIVALENCE* of equivalence relationships.

Listing 1.1. Ontological relationship formalization context.

```
Context Ontology_Relations Sets CLASS, PROPERTIES,
INSATANCES Constants EQUIVALENCE Axioms
  axm1: EQUIVALENCE = { Eq| Eq ∈ CLASS ↔ CLASS ∧
     (∀ x· (x ∈ CLASS ⇒ x↦ x ∈ Eq)) ∧
     (∀ x, y· (x ∈ CLASS ∧ y ∈ CLASS ∧ x↦ y ∈ Eq ⇒ y↦ x ∈ Eq)) ∧
     (∀ x, y, z· (x ∈ CLASS ∧ y ∈ CLASS ∧ z ∈ CLASS ∧ x↦ y ∈ Eq ∧ y ↦ z ∈ Eq ⇒ x↦ z ∈ Eq)) }
  ...
End
```

The equivalences between the different classes are explicitly formalized using the specific equivalence relation *EQo* belonging to the set *EQUIVALENCE* of equivalence relationships. The correct definition of *EQo* relationship is guaranteed by proving the theorem in *thm3*. This proof requirement entailed by the use of formal methods guarantees that used specification relationships like *EQo* formally fulfill the equivalence relationship properties.

Listing 1.2. Diplomas ontology.

```
Context Diplomas_Ontology
Constants Master, Engineer, Bachelor, PhD
Axioms
axm1: partition(CLASS, {Master}, {PhD}, {Bachelor}, {Engineer})
axm2: EQo = {Bachelor↦Bachelor, Engineer↦Engineer, PhD↦PhD,
             Master↦Master, Master↦Engineer, Engineer↦Master}
thm3: EQo ∈ EQUIVALENCE     Theorem
End
```

In addition, the *Diplomas_Ontology* context describes the set *Master, Bachelor, PhD, Engineer* as specific diplomas. It also states that an *Engineer* diploma is equivalent to a *Master* diploma.

6.3 Step 2. Model Specification and Design

Design models are formalized within Event-B using contexts and machines. Static part (constants, types and data) of the design models is defined within contexts and the dynamic part is referred to as a machine which *sees* the defined contexts (static part). Listing 1.3 depicts a generic context defining the relevant set *CONCEPT* characterizing the concepts involved in the definition of a design model. This context is extended to define specific applicative contexts. The static part associated to the case study is given in the *student_Model* context (Listing 1.4). DIPLOMS and STUDENTS concepts are introduced.

Listing 1.3. Generic design model context.

```
Context Design_Model
  Set CONCEPT
End
```

Listing 1.4. Student design model context.

```
Context Student_Model Extends Design_Model
 Constants m, p, e, b, titi, toto, DIPLOMS, STUDENTS
 //Master, PhD, Engineer and Bachelor diplomas
 Axioms
  axm1 : CONCEPT = DIPLOMS ∪ STUDENTS
  axm2 : DIPLOMS ∩ STUDENT = ∅
  axm3 : partition(DIPLOMS, {e},{m},{p},{b})
  axm4 : partition(STUDENTS, {toto},{titi})
  axm5 : finite(CONCEPT)
End
```

Finally, once the different concepts are described, it becomes possible to describe the behavioral part of the model. Indeed, the model considers the case of a student willing to register for a PhD. For the case study, the dynamic part (Listing 1.5) defines the *Register* event within a machine. An invariant *inv1* ensures that a student can register for a *PhD* only if he/she holds a master degree.

Listing 1.5. Student design model machine.

```
Machine Student_Register
Invriants
inv1 : ∀ x (x ∈ STUDENTS ∧ x↦ p ∈ phd_register ⇒ previousDiplom[{x}] ⊆ {m})
Events
  Phd_Register ≜ Any Dip
                   Where    grd1: dip ∈ {m}
                            grd2: previousDiplom[{student}]={dip}
                    Then    act1: phd_register = phd_resgite ∪ {student ↦ p}
End
```

6.4 Step 3. Model annotation

An annotation model is defined within a general context as a generic relationship *ANNOTATION_CLASS* linking design models to ontologies (see Listing 1.6).

Listing 1.6. Annotation relationship formalization context.

```
Context Annotation_Relationship
Extends Ontology_Relations, Student_Model
Axioms
  axm1: ANNOTATION_CLASS =
                    CLASS ↔ CONCEPT
End
```

Listing 1.7. Annotation model context.

```
Context Annotation_Model
Extends Annotation_Relationship
Axioms
  axm1: annotation ∈ ANNOTATION_CLASS
  axm2: annotation = {Master↦m, Engineer↦e}
End
```

Other annotation relationships can be defined, for example to link properties, relationships or constraints etc. Moreover, additional properties may be defined for the annotation. They have not been given in this paper due to space limitations. Listing 1.7 defines annotations for the m and e concepts manipulated at the design model level. The defined annotation states that m and e are annotated by the $Master$ and $Engineer$ ontological concepts respectively. It becomes possible to reason on the equivalence of these concepts (or any other ontological relationship that may exist between $Master$ and $Engineer$ concepts).

Once the annotations are achieved, they can be integrated into the design model and enrich it. Annotations are exploited to access to the ontological concepts and properties in design models. When the annotation relation is established on the design model (Listing 1.5) the annotated student design model is obtained. Listing 1.8 depicts the *Student_Register* machine after the annotation process. The invariant *inv1* is rewritten to integrate the annotations. Indeed, it states that any student holding a previous diploma that is equivalent to the inverse annotation of m can be registered for preparing a PhD. Thus, the equivalence ontological relationship can now be exploited to enrich the model.

Listing 1.8. Annotated Student_Register machine.

```
Machine Student_Register
Invariants    inv1 : ∀ x (x ∈ STUDENTS ∧ x↦ p ∈ phd_register ⇒ ((previousDiplom[{x}] ⊆ {m})
∨ (previousDiplom[{x}] ⊆ annotation[EQo[annotation⁻¹[{m}]]]) ))
Events
  Phd_Register ≜ Any Dip
                 Where    grd1: dip ∈ {m,e}
                          grd2: previousDiplom[{student}]={dip}
                 Then     act1: phd_register = phd_resgite ∪ {student ↦ p}
End
```

6.5 Step 4. Properties Verification

The property verification step is achieved trough discharging all the PO. The capability to annotate design models concepts independently of their usage in design models allows developers to express new properties acting on ontology concepts just by annotating the design models with new properties. The new definition of invariant *inv*1 of Listing 1.8 uses *explicitly* the defined annotation relationship. As a result, the design models can be questioned, verified or checked with regards to new properties exploiting annotations that borrow ontology concepts and properties to the design models. Invariants similar to *inv*1 are defined using the annotation relationship. For our case study, the correctness of the new enriched model is proven. The new POs generated by the annotation are discharged by proving that invariant *inv1* still holds after the *Phd_Register* event is triggered. All the POs associated to the *Student_Register* machine have been proved for all values of *dip* that are equivalent to *Master*.

7 Related Work

Semantic enrichment of models has drawn the attention of several research communities. Different methods and techniques emerged with the aim to enrich the semantics of models using annotation mechanisms. We distinguish three main categories of semantic annotations.

In [18–21], the authors use ontologies for raw data annotation in an informal context. Web pages and documents are annotated with semantic information formalized within linguistic ontologies. Once annotations achieved, formal reasoning is performed. This category of annotation is out of the scope of this paper.

In the second category of approaches ontologies are used for the semantic enrichment of models in a semi-formal context. [22] propose a fully automated technique for integrating heterogeneous data sources called "ontology-based database". This approach assumes the existence of a shared ontology and guarantees the autonomy of each source by extending the shared ontology to define its local ontology. In [23–27] annotations are made in an interoperable context and aim to improve the reading, common understanding and re-usability of the models and thus enabling unambiguous exchange of models. In [28], a reasoning phase is performed based on the output of the annotation phase. The reasoning rules produce inference results: (1) Suggestion of semantic annotation, (2) Detection of inconsistencies between semantic annotations and (3) Conflict identification between annotated objects. These approaches addressing interoperability issues focused on improving the common understanding of models. They do not deal with the formal correctness of models with respect to domain properties and constraints.

The third category of approaches is related to the semantic enrichment of design models related to an application domain using formal annotations. Annotations are directly set up inside the models. Examples of such approaches are the classical pre and post-conditions of Hoare pre-conditions [29] or program annotation tools like Why3 [30]. In [31], the authors introduce real-world types to document the programs with relevant characteristics and constraints of the real-world entities. Real-world types are connected to entities of the programs (variables, functions, etc.). The reasoning and checking of the correctness of programs in regards to real-world types becomes possible by type checking. These approaches seem close to ours, but, to the best of our knowledge, they do not use explicitly modeled ontologies.

Always in the context of formal methods, other approaches use annotations with expressions that make explicit references to ontologies. Indeed, in [32–34], the authors argue that many problems in the development of correct systems could be better addressed through the separation of concerns. [32,33] advocate the re-definition of design models correctness as a ternary relation linking the requirements, the system and application domains. Domain concepts are then explicitly modeled as first-class objects as we did in our approach. Furthermore, similarly to our approach, they propose the formalization of ontologies by Event-B contexts. The formalized information can then be integrated incrementally and directly in the behavioral requirements using refinements. In [34] a DSL abstract syntax and references to domain ontologies are axiomatized into logic theories. These two models are related using a third logical theory. The authors use the Alloy formal method to check the consistency of the unified theory.

Compared to our approach, the approaches cited above use, through annotations, domain information and knowledge directly (i.e. as built in concepts) in the design model. Our approach improves these approaches. It suggests to first separate the ontology and the design model and second to make the annotation explicit using an annotation model. In this way, models are separated from the domain model and thus ontologies and models can evolve asynchronously.

8 Conclusion and Future Work

The integration of domain knowledge and information during the system specification and design phases allows the developers to handle axioms, hypotheses, theorems or properties mined from the application domain. This requirement is a major concern in system engineering where different standards provide system designers with relevant domain knowledge information not explicitly handled by the design models.

In this paper, we have proposed a stepwise methodology allowing system designers to explicitly handle domain knowledge in their design models. Ontologies have been chosen to express the knowledge models describing explicitly the domain knowledge. Depending on the chosen modelling language, these ontologies modelled through concepts, relationships between concepts and associated constraints on the one hand and domain axioms and theorems on the other hand. We have shown that both ontologies and design models can be integrated in a single modelling language. The interest of such integration is semantic alignment where both ontologies, annotations and design models are described in a common shared modelling language.

We have shown, using a toy case study, the deployment of this approach in the case of model driven engineering techniques and formal methods based on refinement and proof using the Event-B method. We have used the Eclipse modelling framework and the Rodin platform to operationalize our proposal. In both cases, the approach proved powerful enough to enrich design models with knowledge domain properties. We have noticed that when used in design models, by annotation, ontologies strengthen with axioms and theorems that were not explicitly defined in the design models. In our case, thanks to the enrichment provided by the annotation mechanisms we have been able to enrich the design models with an equivalence property attached to concepts of the design models.

The proposed approach has been developed as part of the IMPEX-ANR project [35] and has been applied to several case studies in the engineering domain. Indeed, experiments with MDE based techniques have been conducted on the oilfield engineering models [8,36] and avionic systems [37,38]. Formal methods have been applied in the case of human computer interaction models [39], avionic systems [32] and in system engineering [33,40,41].

The work presented in this paper opened several new research directions. The whole coverage of available ontology languages like OWL or Plib and the associated annotation mechanisms is currently under study. We are designing meta-models in the MDE setting and generic contexts in the Event-B setting. Then, the case of semantic mismatch, where ontologies and design models are not described in the same modeling language, needs to be addressed. Semantic alignment shall be studied. Finally, the integration of more than one design model addressing different aspects or view of a single system sharing a common ontology needs to be handled as well. Indeed, our idea is to offer to the designer the capability to use validated properties of a given model as hypotheses and axioms as hypotheses in another design model.

References

1. Gruber, T.R.: A translation approach to portable ontology specifications. Knowl. Acquis. **5**(2), 199–220 (1993)
2. Jean, S., Pierra, G., Aït Ameur, Y.: Domain ontologies: a database-oriented analysis. In: Filipe, J., Cordeiro, J., Pedrosa, V. (eds.) WEBIST 2006. LNBIP, pp. 238–254. Springer, Heidelberg (2006)
3. Bechhofer, S., Van Harmelen, F., Hendler, J., Horrocks, I., McGuinness, D., Patel-Schneider, P., Stein, L., et al.: Owl web ontology language reference. W3C Recommendation 10 (2004)
4. ISO: Industrial automation systems and integration - parts library - part42: description methodology: methodology for structuring parts families. ISO ISO13584-42, Geneva, Switzerland (1998)
5. ISO: Industrial automation systems and integration - parts library - part25: logical resource: logical model of supplier library with aggregate valuesand explicit content. ISO ISO13584-25, Geneva, Switzerland (2004)
6. Brickley, D., Guha, R.V.: RDF vocabulary description language 1.0: RDF schema. W3C Recommendation, W3C, February 2004

7. Aït Ameur, Y., Méry, D.: Making explicit domain knowledge in formal system development. Sci. Comput. Program. (2015, to appear)
8. Silveira Mastella, L., Aït-Ameur, Y., Jean, S., Perrin, M., Rainaud, J.-F.: Semantic exploitation of engineering models: an application to oilfield models. In: Sexton, A.P. (ed.) BNCOD 26. LNCS, vol. 5588, pp. 203–207. Springer, Heidelberg (2009)
9. Belaid, N., Jean, S., Aït Ameur, Y., Rainaud, J.F.: An ontology and indexation based management of services and workflows application to geological modeling. IJEBM **9**(4), 296–309 (2011)
10. Schmidt, D.C.: Model-driven engineering. IEEE Comput. Soc. **39**(2), 25 (2006)
11. OMG: OMG Unified Modeling Language (OMG UML), Superstructure, Version2.4.1 (2011)
12. OMG: Meta Object Facility (MOF) Core Specification Version 2.0 (2006)
13. OMG: OMG Object Constraint Language (OCL), Version 2.3.1, January 2012
14. OMG: Meta Object Facility (MOF) 2.0 Query/View/TransformationSpecification, Version 1.1, January 2011
15. Abrial, J.R.: Modeling in Event-B - System and Software Engineering. Cambridge University Press, Cambridge (2010)
16. Dijkstra, E.W.: A Discipline of Programming. Prentice Hall PTR, Upper Saddle River (1977)
17. Abrial, J.R., Hallerstede, S.: Refinement, decomposition, and instantiation of discrete models: application to event-b. Fundam. Inf. **77**(1–2), 1–28 (2007)
18. Bontcheva, K., Tablan, V., Maynard, D., Cunningham, H.: Evolving gate to meet new challenges in language engineering. NLE **10**(3–4), 349–373 (2004)
19. Cunningham, H., Maynard, D., Bontcheva, K.: Text Processing with Gate. Gateway Press, Murphys (2011)
20. Despres, S., Szulman, S.: Terminae method and integration process for legal ontology building. In: Ali, M., Dapoigny, R. (eds.) IEA/AIE 2006. LNCS (LNAI), vol. 4031, pp. 1014–1023. Springer, Heidelberg (2006)
21. Handschuh, S., Volz, R., Staab, S.: Annotation for the deep web. IEEE (5) (2003)
22. Bellatreche, L., Pierra, G., Xuan, D.N., Hondjack, D., Ameur, Y.A.: An a priori approach for automatic integration of heterogeneous and autonomous databases. In: Galindo, F., Takizawa, M., Traunmüller, R. (eds.) DEXA 2004. LNCS, vol. 3180, pp. 475–485. Springer, Heidelberg (2004)
23. Boudjlida, N., Panetto, H.: Annotation of enterprise models for interoperability purposes. In: CAISE, April 2008
24. Wang, Y., Li, H.: Adding semantic annotation to UML class diagram. In: ICCASM (2010)
25. Lin, Y., Strasunskas, D.: Ontology-based semantic annotation of process templates for reuse. In: Proceedings of the CAiSE, vol. 5. Citeseer (2005)
26. Lin, Y., Strasunskas, D., Hakkarainen, S.E., Krogstie, J., Solvberg, A.: Semantic annotation framework to manage semantic heterogeneity of process models. In: Martinez, F.H., Pohl, K. (eds.) CAiSE 2006. LNCS, vol. 4001, pp. 433–446. Springer, Heidelberg (2006)
27. Zouggar, N., Vallespir, B., Chen, D.: Semantic enrichment of enterprise models by ontologies-based semantic annotations. In: EDOC. IEEE (2008)
28. Liao, Y., Lezoche, M., Panetto, H., Boudjlida, N., Loures, E.R.: Formal semantic annotations for models interoperability in a PLM environment. arXiv (2014)
29. Hoare, C.A.R.: An axiomatic basis for computer programming. Commun. ACM **12**, 576–580 (1969)
30. Filliâtre, J.C., Paskevich, A.: Why3 – where programs meet provers. In: ESOP

31. Knight, J., Xiang, J., Sullivan, K.: A rigorous definition of cyber physical systems. In: Trustworthy Cyber Physical Systems Engineering (2016, to appear)
32. Ait-Ameur, Y., Gibson, J.P., Méry, D.: On implicit and explicit semantics: integration issues in proof-based development of systems. In: Margaria, T., Steffen, B. (eds.) ISoLA 2014, Part II. LNCS, vol. 8803, pp. 604–618. Springer, Heidelberg (2014)
33. Méry, D., Sawant, R., Tarasyuk, A.: Integrating domain-based features into event-b: a nose gear velocity case study. In: Bellatreche, L., Manolopoulos, Y., Zielinski, B., Liu, R. (eds.) MEDI 2015. LNCS, vol. 9344, pp. 89–102. Springer, Heidelberg (2015). doi:10.1007/978-3-319-23781-7_8
34. de Carvalho, V.A., Almeida, J.P.A., Guizzardi, G.: Using reference domain ontologies to define the real-world semantics of domain-specific languages. In: Jarke, M., Mylopoulos, J., Quix, C., Rolland, C., Manolopoulos, Y., Mouratidis, H., Horkoff, J. (eds.) CAiSE 2014. LNCS, vol. 8484, pp. 488–502. Springer, Heidelberg (2014)
35. IMPEX Consortium. Formal models for ontologies. Technical report (2015)
36. Mastella, L.S.: Semantic exploitation of engineering models: application to petroleum reservoir models. Ph.D. thesis, ENSMP (2010)
37. Aït Ameur, Y., Hacid, K.: Report ame corac-panda project. Technical report, Institut de Recherche en Informatique de Toulouse, Toulouse university (2015)
38. Hacid, K.: Explicit definition of prperties by model annotation. Technical report, Institut de Recherche en Informatique de Toulouse, Toulouse university (2014)
39. Chebieb, A., Aït Ameur, Y.: Formal verification of plastic user interfaces exploiting domain ontologies. In: TASE (2015)
40. Simon-Zayas, D.: A framework for the management of heterogeneous models in Systems Engineering. Theses, ISAE-ENSMA - Poitiers, June 2012
41. Zayas, D.S., Monceaux, A., Aït Ameur, Y.: Knowledge models to reduce the gap between heterogeneous models: application to aircraft systems engineering. In: ICECCS (2010)