Annotation of Engineering Models by References to Domain Ontologies

Kahina Hacid, Yamine Aït-Ameur

Annotation of Engineering Models by References to Domain Ontologies

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. Complex engineering systems execute within different contexts and domains. The heterogeneity induced by these contexts is usually implicitly handled in the development cycle of such systems. We claim that reducing this heterogeneity can be achieved by handling explicitly the knowledge mined from these domains and contexts. Verification and validation activities are improved due to the expression and verification of new constraints and properties directly extracted from the context and domains associated to the models. In this paper, we advocate the use of domain ontologies to express both domain and context knowledge. We propose to enrich design models that describe complex information systems, with domain knowledge, expressed by ontologies, provided by their context of use. This enrichment is achieved by annotation of the design models by references to ontologies. Three annotation mechanisms are proposed. The resulting annotated models are checked to validate the new minded domain properties. We have experimented this approach in a model driven engineering (MDE) development setting.

Keywords: Design models · Ontologies · Annotation · Model engineering

1 Introduction

In general, during system development, the knowledge provided by the engineering domain is not explicitly taken into account in the different models that result from this development. The system development process leads to the production of several heterogeneous models corresponding to different views or analyses of the same system. In this context, the most important heterogeneity factor, in addition to the one due to the use of different modelling languages, is related to information, knowledge and assumptions of the domain (the environment and context of execution of the systems) that are not explicitly formalised and therefore not used in the models of these systems. One of the reasons is the absence of such domain knowledge in the modelling language. The developer has to handle this information in the development process. It may happen that the assumptions made by the developers are contradictory due to an implicit consideration of domain knowledge. Indeed, although systems are developed according

DOI: 10.1007/978-3-319-45547-1_19

to development standards and good practices, a large part of the knowledge required to the interpretation and validation of these models of systems remain implicit. This situation may raise several insufficiencies and drawbacks during system verification and validation activities. More precisely, a system assumed to be sound after verification and validation may loose some of its properties if information related to its domain, context and environment are integrated to the model. Indeed, general knowledge information expressed as properties may be no longer valid in the developed model. For example, the addition of two variables X and Y, occurring in a design model, may not be valid if domain information states that X is measured in *meters* and Y in *miles*, although the modelling language allows such addition. The objective of our work is to propose a sound and operationalised approach to strengthen design models with domain knowledge resources carried out by the engineering domain associated to the designed models. We consider that on the one hand, ontologies are good candidates for describing and making explicit such knowledge [1], and on the other, annotation of model resources by references to ontologies makes it possible to handle domain knowledge in design models. More precisely, to reach this objective, we propose a solution involving in a first step the use of ontologies to clarify and formalise the domain knowledge. As a second step, annotation mechanisms are defined and set up to link both design models and ontologies. It becomes possible to express and verify new properties of the enriched design models. This paper is structured as follows. Section 2 recalls basic definitions of domain ontologies. Section 3 presents the NoseGear case study [2] illustrating the work developed in this paper. Our approach for strengthening models through an annotation based method, the developed annotation mechanisms and details of the implementation on the basis of a model driven engineering (MDE) approach are presented in Sect. 4. Application of the proposed approach to the case study is given in Sect. 5. Finally, Sect. 6 overviews different approaches promoting annotation and semantic enrichment of models. A conclusion ends this paper and identifies some research directions.

2 Domain Ontologies as Models for Domain Knowledge

Gruber defines an ontology as *an explicit specification of a conceptualisation* [3]. 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* [4]. In this definition, entity represents any concept belonging to the studied domain. The term dictionary emphasises that any entity and any kind of domain relationship described in the domain ontology may be referenced directly, for any purpose and from any context, independently of other entities or relationships, by a symbol (URI i.e. unique resource identifier). Ontology design requires to express a set of basic concepts related to structure (class, relationships, etc.), description (properties, attributes, etc.) and behaviour (derivation expression, labelled transitions systems, etc.). An ontology modelling language is required to describe such ontologies. Several ontology

modelling languages have been developed so far. OWL [5], PLIB [6,7], RDFS [8] are some examples of such languages. These languages describe ontology entities using different modelling artefacts like hierarchies, properties, relationships, instances and individuals, constraints, etc. According to [4], a domain ontology is a domain conceptualisation that obeys to the three fundamental criteria: being formal, consensual and offering references capabilities.

Formal. An ontology is a conceptualization based on a formal theory to check consistency properties and to perform some automatic reasoning over concepts.

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.

Capability to be referenced. Each ontological concept is associated with an URI. References to this concept become possible, using this identifier, from any environment, independently of the ontology where this concept was defined. 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, etc.

3 The NoseGear Case Study

The NoseGear [2] is a sub-component of the landing gear of an airplane. The objective of the case study addressed in this paper is to estimate the speed of a grounded airplane. Speed is estimated by measuring the time taken by the NoseGear wheel to achieve a turn. An interruption is triggered each time a round is completed. This interruption increments two counters: a counter which calculates the number of turns the wheel made, and another recording the current time value. Then, a function operates to calculate the speed of the plane from the recorded values of both counters. The complete description of the NoseGear case study is given in [2]. This case study involves several independent views of the same system (physical, computing, etc.). We assume that multiple ontologies are used to express knowledge and properties associated to each view. Since these views relate to the same engineering area, *implicit* relationships may exist between the different views and therefore between the related ontologies. Through this case study, our goal is, first, to identify and to formalise these implicit relations existing between ontologies. Then, we use them to link multiple system components of the NoseGear design model. *Constraints* are used to express invariants defining properties mined from different ontologies.

4 Our Approach

4.1 Methodology

We propose a stepwise methodology to establish a formal explicit link between these two models. Figure 1 shows the overall schema of the approach involving four steps. Concepts, properties and constraints of the studied domain are

represented and formalised within a knowledge model (domain ontologies) at step 1. Specific design models are defined at step 2. At step 3, relationships between design model entities and the corresponding knowledge concepts are identified. Three different kinds of relationships can be set up, they are discussed in Sect. 4.2. Finally, at step 4, the annotated model is analysed to determine whether the constraints associated to the knowledge domain, carried out by the annotations, can be expressed in the new enriched design model. More details about the developed approach can be found in [9].

4.2 Model Annotation: Three Cases

Relations, formalized as model annotations, are established between design model entities and ontology concepts. Three annotation mechanisms are identified.

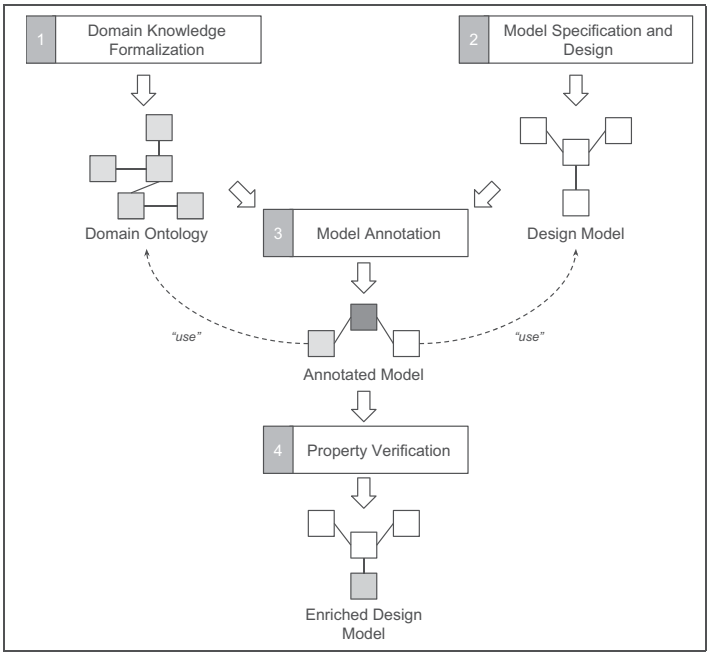
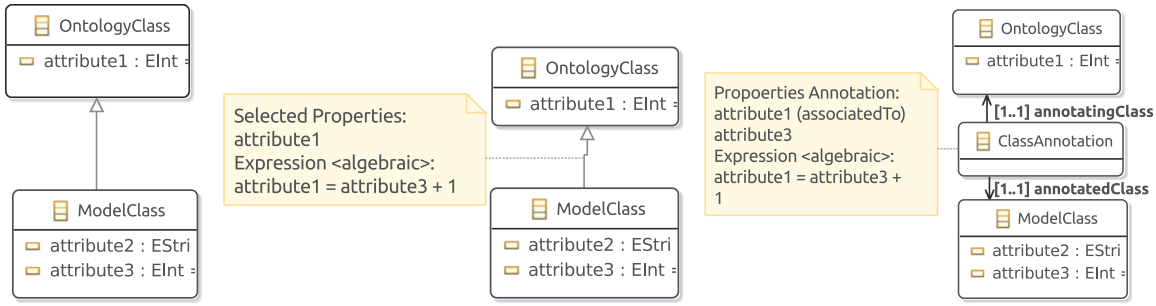


Fig. 1. A four steps methodology for handling domain knowledge in design models.

Annotation by inherice. (Figure 2(a)) is defined by the *Is_a* relationship (subsumption relationship [10]). In this case, a concept of the ontology subsumes an entity of the design model. The mapping relationship is the subsumption (is_a). Properties, attributes, rules and constraints that apply to the ontological concept are also applicable to the design model entity. This annotation maintains the ontological reasoning and preserves it at the design model level. This relationship is usually set up in an a priori setting where the ontology is designed before the design models are defined.

Annotation by partial inheritance. (Figure 2(b)) is defined by the *Is_case_of* relationship. It is also a subsumption relationship. It defines a partial inheritance [10]. This relation behaves like the *Is_a* relationship, except that it does not require the inheritance of all the ontological properties. In fact, only some of the relevant properties and constraints of the ontology class are imported. The annotation mechanism is in charge of selecting which properties and constraints are imported. The main advantage of this approach is flexibility, it can be set up in any situation (a priori and a posteriori).



(a) Inheritance (*Is_a*). (b) Partial inheritance (*Is_case_of*). (c) Association.

Fig. 2. Annotations mechanisms.

Annotation by association. (Figure 2(c)) *Is_a* and *Is_case_of* relationships are based on relationships that preserve subsumption reasoning. It may happen that an annotation needs specific relationships defined by the users. These relationships are themselves described in ontologies. This annotation enables the connection of ontological classes with model classes by association. In this case, subsumption reasoning contained in the ontology is not preserved at the annotated design model level. But, the properties borrowed from the association to the design model can be used to express properties.

Annotation meta-models. The annotation mechanisms described above need to be described in the modelling language in order to get a uniform modelling setting (here UML). A consequence of the choice of UML, is that the *Is_a* relationship is built-in and does not need to be defined. The *Is_Case_of* and *Association* annotation relations need to be defined within the modelling language. Two meta-models (one for each type of annotation) describing these mechanisms are introduced (Fig. 3). They link design model entities and ontology concepts at the meta-model level.

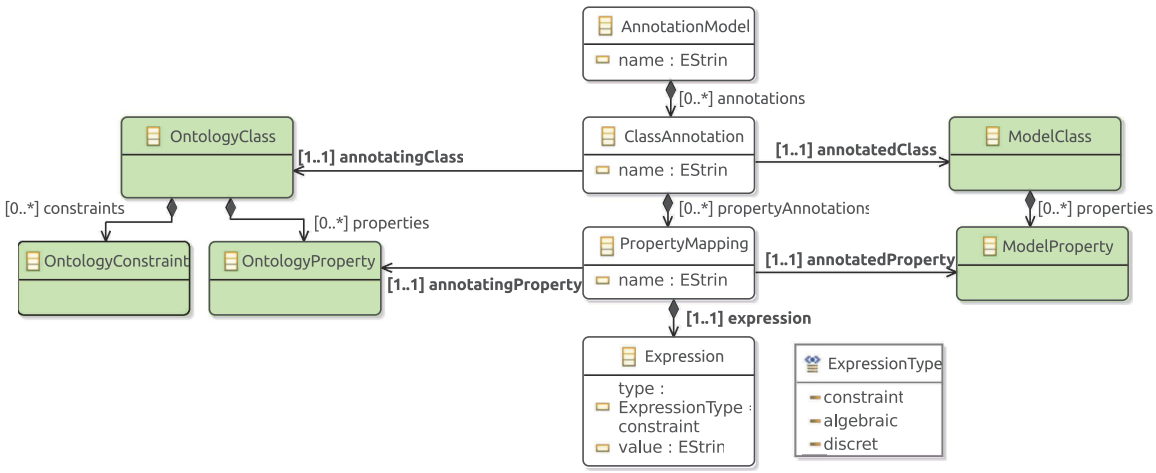


Fig. 3. Annotation by *Association* meta-model.

AnnotationModel: the entry point of the annotation models.
ClassAnnotation: an association of ontology concepts and design model classes.
PropertyMapping: relations between properties of design model and of ontology.
Expression: algebraic expressions to compose properties (constraint, derivation).

4.3 Properties Expression and Verification

The last step of the approach analyses the obtained annotated design models. The 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 (or expressible) in the enriched model if they have been explicitly selected or linked to model properties during the annotation process (third step of Fig. 1). It may happen that these relations, properties and constraints could not be expressed at the design model level and thus not valuable at instantiation level due to the absence of attributes to express them or of the values of these attributes (instances). These constraints become meaningless. At this level, an analysis of the obtained annotated model is necessary after an annotation by *Is_Case_Of* or by *association* because these two types of annotation offer the possibility of having only some ontological properties in the enriched design model. The annotation by *Is_a* does not suffer from this drawback since all ontological constraints in the design model can be expressed (all the properties of the annotating ontological classes are inherited in the design model). The proposed analysis procedure is depicted on Fig. 4. The process begins by selecting an annotated class in the model and analyse it to retrieve the ontological class that annotates it. Each constraint of the annotating class is then analysed to decide if it is expressible in the model. The expressible constraints are integrated into the model, the other ones are discarded.

```
BEGIN
   For (an annotated model)
   begin
      Select a new annotated class;
      Select the corresponding onology class;
      For (all ontology class constraints)
      begin
         Select a new constraint;
         if (constraint is expressible in the design model) then
            Integrate to the domain model;
         else
            Add an error message;
         endif;
      end;
   end;
END;
```

Fig. 4. Algorithm of the verification process.

5 Application to the NoseGear Case Study

5.1 Step 1. a Domain Ontolgy

Figure 5 depicts the ontology used to annotate the NoseGear design model. It is composed of two parts defining specific modelled domain knowledge: an avionic ontology *PlaneOntology* and a devices ontology *DevicesOntology* composed of classes and properties. *Constraints* are defined on the ontology model *Ontology* (Fig. 5).

Constraints. We present two *constraints*: N_{max} and F_{CPU}. They express implicit relations that may exist between different views of the NoseGear models (e.g. computation and physical views). Formalized within ontologies, these *constraints* describe implicit links between domain ontologies. They link the components of the design model after making explicit the knowledge in the model.

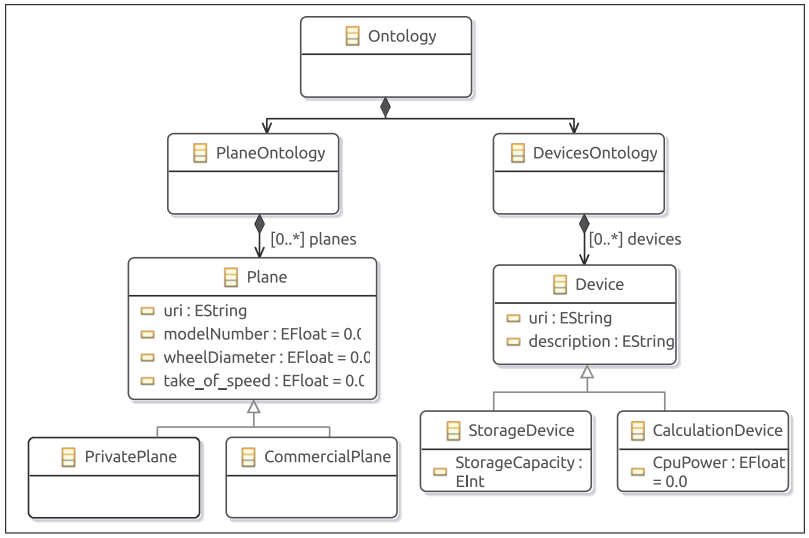


Fig. 5. Overview of the ontology of the NoseGear.

N_{max} of the wheel. This *constraint* determines the optimal memory size *StorageCapacity*) of the landing gear's lap counter (*StorageDevice* in the ontology). We calculate the maximum number of laps (N_{max}) that can be made by the wheel on the take off track. N_{max} is obtained by dividing the maximum distance that can be travelled (i.e. length of the take off track) by the distance travelled in one turn (circumference of the wheel). $N_{max} = D_{max}/C_{wheel}$ is obtained.

The maximum memory size of the laps counter can then be deduced by bounding N_{max}: $2^{k-1} \leqslant N_{max} \leqslant 2^k$, k being the number of bits needed to represent N_{max}. As a consequence, we have been able to exploit the topology knowledge to determine the optimal size of the register encoding the N_{max} value.

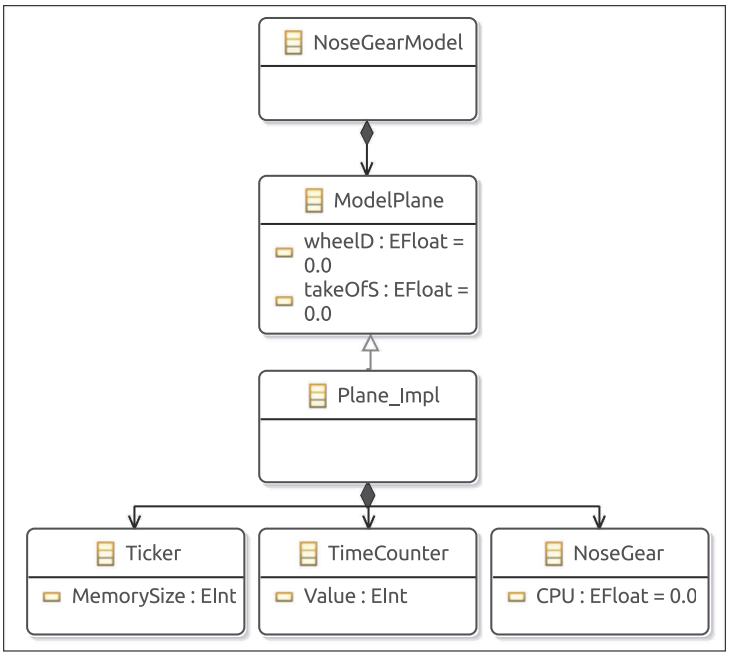


Fig. 6. Overview of the NoseGear design model.

$\mathbf{F}_{CPU}$. The second *constraint* determines which kind of CPU processor (the CPU frequency F_{CPU}) is needed to support calculations related to the number of laps of the NoseGear wheel. To be responsive, the CPU frequency must be adjusted to be able to detect at best every lap of the wheel. It involves having at least a rising edge clock whenever a complete lap is done. The following relationship: $F_{CPU} = take\_of\_speed/C_{wheel}$ is obtained. Here, cinematic theory specifies the required frequency of a calculator.

These two *constraints* are defined in the presented ontologies. They help to explicit existing relations between different knowledge domains of the NoseGear model.

5.2 Step 2. Design Models

The design model of the *NoseGear* describes a simple architecture of an aircraft. An overview of the Ecore model is given in Fig. 6. Note that the NoseGear architecture is usually represented by several models, each one describing a specific view. These models are not given here due to space limitation but may be obtained from [11]). The NoseGear model is defined by the abstract system *ModelPlane*. *Plane_Impl* implements the Plane system. It is composed of the *Nosegear* (calculator to detect and calculate the number of laps the wheel makes), the *Ticker* (the counter to store the value of the number of laps) and *TimeCounter* (the time counter).

5.3 Step 3. Annotation Process

In Fig. 7, *annotation by association* is established between *Plane* ontological class and *ModelPlane* of the model using *ClassAnnotation* (bullet 1). Annotations are also established between *StorageDevice* and *Ticker* and between *Calculation-Device* and *NoseGear*. A correspondence is established between *take_of_speed* of the ontology and *takeOfS* of the model using *PropertyAnnotation* (bullet 2). Correspondences are also established between: *wheelD* and *wheelDiameter*, *CpuPower* property of the ontological *CalculationDevice* and *CPU* property of *NoseGear*, and finally, between *StorageCapacity* property of *StorageDevice* and *memorySize* property of the *Ticker* model.

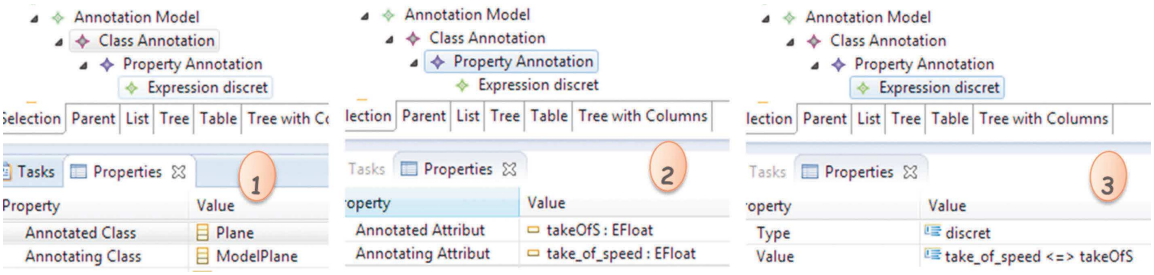


Fig. 7. Overview of the annotation process of the NoseGear design model.

A *discrete* typed expression states that *take_of_speed* is equivalent to *takeOfS* (bullet 3). Other expressions are defined for the other mapped properties.
***wheelD* and *wheelDiameter*.** Algebraic expression $wheelD = wheelDiameter * 100$ is defined. *wheelD* is measured in centimeters and *wheelDiameter* in meters.
***Cpu* and *CpuPower*.** A discrete expression $CpuPower \Leftrightarrow CPU$ is established.
***memorySize* and *StorageCapacity*.** $StorageCapacity \Leftrightarrow memorySize$ is set.

5.4 Step 4: Property Verification

The *verification* step consists in analysing the obtained annotated NoseGear design model. A constraint analysis is trigged according to the algorithm of Fig. 4. This analysis shows that the ontological*constraints* N_{max} and F_{CPU} linking different views of the system are expressible within the enriched model. Indeed, all the ontological properties they are related to are retrieved within the NoseGear model. Thus, they are included in the final NoseGear design model to ease the plane speed computation.

6 Related Work

Many researchers studied the issue of semantic enrichment of models. [12] proposed informal annotations for business models in an interoperability context. Annotations are classified according to their type (decoration, linking, instance identification etc.), their content and artefacts models. In [13], the authors propose an annotation method which promotes mapping UML class's attributes with domain ontology concepts. It shows the corresponding relations with a markup language and UML itself. [14], presented a semantic annotation method allowing the annotation of templates, process model fragments and modelling languages. General Process Ontology (GPO) is used as a reference in the ontology modelling process. [15] propose a semantic annotation framework for the management of heterogeneous process models according to four perspectives: basic description of process models (profile annotation), process's modelling languages (meta-model annotation), process model (model annotation) and purposes of process models (annotation goal). In [16], a reasoning phase is based on the output of the annotation phase. Reasoning rules produce inference results: (1) suggestion of semantic annotation, (2) detection of inconsistencies between semantic annotations and (3) conflict identification in annotated objects.

Compared to our work, the approaches cited above, propose informal and restrictive annotations to improve the common understanding of models and to address interoperability issues. They do not deal with the formal correctness of models with respect to domain properties and constraints.

7 Conclusion and Future Work

The work achieved in this paper starts from the general observation that domain knowledge related properties are not handled nor formalised during the system development. It focuses on making explicit domain knowledge. It shows how the integration of domain knowledge in design models handles the expression and the verification of new properties and constraints that emerge from the explicit expression of domain specific knowledge. In order to allow such properties expression and verification, we proposed an incremental model oriented approach to enrich and strengthen design models thanks to references to domain knowledge resources. This stepwise approach is based on model engineering techniques and is composed of four steps. First, ontologies are set up in order to make explicit and formalise domain knowledge using concepts like classes, properties, constraints, relations etc. To get a uniform model for all the resources involved in our approach, we have characterised these ontologies using a meta-model. Then, as a second step, we have defined and used annotations to explicitly establish a link between the domain ontologies and the design models. Three types of annotations have been defined for this purpose: annotation *by inheritance Is_a*, annotation by *partial inheritance Case_of* and annotation by *association*. Finally, the last step checks if the ontological constraints can be expressed and interpreted within the annotated design model before they can be integrated to the final enriched design model obtained after annotation. A prototype implementing this approach has been built on top of the EMF Eclipse platform. This approach has been developed as part of the IMPEX-ANR project [17] and has been deployed within formal methods based on refinement and proof using the Event-B method. It has been applied to several case studies of the engineering domain [9].

Several other research directions to pursue our work can be envisaged. First, we are interested in promoting our approach to handle, during the annotation process of design models, instances of ontologies. Design models could be annotated by both classes and instances of an ontology. Then, the capability to annotate behavioural resources in design models (like state-transition systems, events, etc.) is another open issue. Finally, we are interested in moving forward towards the formalisation of an ontological language in an upper level within a formal context based on proof using Event-B [18] theories.

References

1. Aït Ameur, Y., Méry, D.: Making explicit domain knowledge in formal system development. Sci. Comput. Program. **121**, 100–127 (2016)
2. The NoseGear Case Study. http://www.cl.cam.ac.uk/mjcg/FMStandardsWorkshop/NoseGear.html
3. Grube: toward principles for the design of ontologies used for knowledge sharing (1993)
4. Jean, S., Pierra, G., Aït Ameur, Y.: Domain ontologies: a database-oriented analysis. In: Filipe, J., Cordeiro, J., Pedrosa, V. (eds.) WEBIST 2005/2006, LNBIP, vol. 1, pp. 238–257 (2006)
5. Ontology web language. http://www.w3.org/2001/sw/wiki/OWL
6. ISO: Parts library - part 42: Description methodology: Methodology for structuring parts families, ISO ISO13584-42 (1998)
7. ISO: Parts library - part 25: Logical resource: Logical model of supplier library with aggregate values and explicit content, ISO ISO13584-25 (2004)
8. RDF Schema. http://www.w3.org/TR/rdf-schema/
9. Hacid, K., Aït Ameur, Y.: Strengthening MDE and formal design models by references to domain ontologies, a model annotation based approach. In: ISOLA (2016, to appear)
10. Jean, S.: OntoQL, an exploitation language for ontology-based databases, Theses (2007)
11. The NoseGear Case Study. http://www.cl.cam.ac.uk/mjcg/FMStandardsWorkshop/sampleCode.pdf
12. Boudjlida, N., Panetto, H.: Annotation of enterprise models for interoperability purposes. In: IWAISE, April 2008
13. Wang, Y., Li, H.: Adding semantic annotation to UML class diagram. In: ICCASM (2010)
14. Lin, Y., Strasunskas, D.: Ontology-based semantic annotation of process templates for reuse. In: CAiSE (2005)
15. Lin, Y., Strasunskas, D., Hakkarainen, S., Krogstie, J., Solvberg, A.: Semantic annotation framework to manage semantic heterogeneity of process models. In: CAiSE (2006)
16. Carvalho, V.A., Almeida, J.P.A., Guizzardi, G.: Using reference domain ontologies to define the real-world semantics of domain-specific languages. In: CAiSE (2014)
17. Consortium: Formal models for ontologies, Technical report (2015)
18. Abrial, J.R.: Modeling in Event-B - System and Software Engineering. Cambridge University Press, New York (2010)