Formal Modelling of Ontologies : An Event-B based Approach Using the Rodin Platform*

Yamine AIT AMEUR
IRIT/INPT-ENSEEIHT
Toulouse, France
yamine@enseeiht.fr

Idir AIT SADOUNE
LRI/CentraleSupelec
Gif Sur Yvette, France
idir.aitsadoune@centralesupelec.fr

Kahina HACID
IRIT/INPT-ENSEEIHT
Toulouse, France
kahina.hacid@enseeiht.fr

Linda MOHAND OUSSAID
LRI/CentraleSupelec
Gif Sur Yvette, France
linda.mohandoussaid@centralesupelec.fr

This paper reports on the results of the French ANR IMPEX research project dealing with making explicit domain knowledge in design models. Ontologies are formalised as theories with sets, axioms, theorems and reasoning rules. They are integrated to design models through an annotation mechanism. Event-B has been chosen as the ground formal modelling technique for all our developments. In this paper, we particularly describe how ontologies are formalised as Event-B theories.

1 Introduction

Nowadays, it is well accepted that formal ontologies are commonly used as support for the axiomatisation of the knowledge describing a domain of interest. In particular, for domains in the engineering area where concepts are well mastered by the different stakeholders, ontologies play a major role for knowledge exchange and heterogeneity reduction.

Meanwhile, we observe that defining a formal framework for integrating both ontologies represented by knowledge models and design models of particular systems did not draw the attention of many researchers in system engineering.

Approaches like those of [3][4][5][7][9][12] supporting the integration of both ontologies and design models contribute to strengthen these design models by offering the capability to design models to borrow knowledge from ontologies, using a particular annotation relationship. As a consequence, the design models are enriched and strengthened with axioms, theorems or invariants issued from the used ontologies.

This paper presents a summary of the work achieved in the context of the French ANR IMPEX research project. Ontologies are formalised as theories with axioms, theorems and reasoning rules. Event-B [1] has been chosen as the ground formal modelling technique for all our developments.

2 Event-B formal developments

The Event-B method [1] is a formal method based on first order logic and set theory. It relies on the notions of pre-conditions and post-conditions, weakest pre-condition and the calculus of substitution. An Event-B model is characterised by a set of variables, defined in the VARIABLES clause that evolve

*The work reported in this paper has been supported by the ANR project IMPEX ref : Projet-ANR-13-INSE-0001

R. Laleau, D. Méry, S. Nakajima, E. Troubitsyna (Eds): Joint Workshop on
Handling IMPlicit and EXplicit knowledge in formal system development
and Formal & Model-Driven Techniques for Developing Trustworthy Systems.
EPTCS 271, 2018, pp. 24–33, doi:10.4204/EPTCS.271.2

thanks to events defined in the EVENTS clause. It encodes a state transition system where the variables represent the state and the events represent the transitions from one state to another.

2.1 Event-B model

An Event-B model is made of several components of two kinds : Machines and Contexts. Machines contain the dynamic parts (states and transitions) of a model whereas Contexts contain the static parts (axiomatisation and theories) of a model. A Machine may be refined by another one, and a Context may be extended by another one. Moreover, a Machine sees one or several Contexts (figure 1).

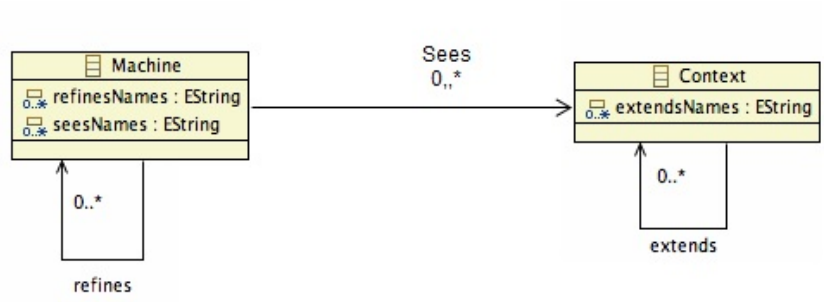

Figure 1: MACHINE and CONTEXT relationships

CONTEXT $context\_identifier_1$ **EXTENDS** $context\_identifier_2$ **SETS** s **CONSTANTS** c **AXIOMS** axm : $A(s,c)$ **THEOREMS** thm : $T(s,c)$ **END**	**MACHINE** $machine\_identifier_1$ **REFINES** $machine\_identifier_2$ **SEES** $context\_identifier_1$ **VARIABLES** v **INVARIANTS** inv : $I(s,c,v)$ **THEOREMS** thm : $T(s,c,v)$ **VARIANT** $V(s,c,v)$ **EVENTS** $< event\_list >$ **END**

Figure 2: The structure of an Event-B development

A Context is defined by a set of clauses (figure 2) as follows.

- SETS describes a set of abstract and enumerated types.
- CONSTANTS represents the constants used by a model.
- AXIOMS describes, in first order logic expressions, the properties of the attributes defined in the CONSTANTS clause. Types and constraints are described in this clause as well.
- THEOREMS are logical expressions that can be deduced from the axioms.

Similarly to Contexts, a Machine is defined by a set of clauses (figure 2). Briefly, the clauses mean.

- VARIABLES represent the state variables of the model of the specification. Refinement may introduce new variables in order to enrich the described system.
- INVARIANTS describe, by first order logic expressions, the properties of the variables defined in the VARIABLES clause. Typing information, functional and safety properties are usually described in this clause. These properties shall remain true in the whole model. Invariants need to be preserved by events (by induction). It also expresses the gluing invariant required by each refinement for property preservation.
- THEOREMS defines a set of logical expressions that can be deduced from the invariants. They do not need to be proved for each event like for the invariant.
- VARIANT introduces a decreasing natural number to ensure termination of *"convergent"* events.
- EVENTS defines all the events (transitions) that occur in a given model. Each event is characterized by its guard and by the actions performed when the guard is true. Each Machine must contain an "*Initialisation*" event. The events occurring in an Event-B model affect the state described in VARIABLES clause.

2.2 Proof obligation rules

Proof obligations (PO) are associated to any Event-B model. They are automatically generated. *The proof obligation generator plugin* in the Rodin platform [2] is in charge of generating them. These PO need to be proved in order to ensure the correctness of developments and refinements. The obtained PO can be proved automatically or interactively by *the prover plugin* in the Rodin platform. The rules for generating proof obligations to prove the correctness of an Event-B development are given in [1].

3 Need to embed ontologies in formal developments

When design models are produced, designers use domain knowledge in order to formalise the concepts and components of the system to be designed. Usually, this knowledge is not made explicit and is used in an empirical manner. There is no complete formalisation for the reasoning that can be associated to this knowledge

Embedding ontologies in design models in a modular way makes it possible to use ontology concepts and associated reasoning rules in the design models. The interest is to strengthen the models as shown in our previous work [3][4][5][7][9][12]. The Event-B method [1] has been set up to show how our approach works.

When integrating ontologies and design models, the main difficulty consists in defining a sound integration operation in order to overcome the difficulties resulting from possible semantic gaps that may occur due to the use of ontologies in formal development models. For example, we have adopted the closed world assumption that fits with the studied systems.

To illustrate the approaches proposed in the context of the IMPEX project, we use an extract of the ontology of diplomas described using OWL formalism [11] (figure 3). It defines classes for diplomas (*Diplom*). Other classes subsumed by the diploma class are defined: *Bachelor*, *Master*, *Engineer* and *Phd*. This Ontology states that *Master* and *Engineer* diplomas are equivalent diplomas, and the concept *Diplomas_For_Phd* is defined as the union of the students that hold an engineer or a master diploma.

```
<Ontology>
        ...
        <Class ID="Diplom"/>

        <Class ID="Bachelor">
                <subClassOf resource="Diplom"/>
        <Class>

        <Class ID="Master">
                <subClassOf resource="Diplom"/>
        </Class>

        <Class ID="Engineer">
                <subClassOf resource="Diplom"/>
                <equivalentClass resource="Master"/>
        </Class>

        <Class ID="Phd">
                <subClassOf resource="Diplom"/>
        </Class>

        <Class ID="Diplomas_For_Phd">
                <unionOf parseType="Collection">
                        <Class about="Master"/>
                        <Class about="Engineer"/>
                </unionOf>
        </Class>
        ...
</Ontology>
```

Figure 3: Extract of the diplomas Owl ontology

4 Ontologies as theories

As mentioned above, ontologies are formalised as theories integrated to formal system modelling languages. In the context of the IMPEX project, we have identified two approaches to define ontologies as formal theories. These two approaches use two different modelling processes: shallow [9] and deep modelling [6, 7].

4.1 Shallow modelling: Ontologies as contexts

The approach that uses shallow modelling consists in modelling the ontology concepts directly in the target modelling language without keeping trace of the structure of the ontology modelling language concepts [9]. One way to integrate the ontology concepts 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 our case, a shallow modelling approach consists in encoding the ontology concepts (classes, properties, ...) directly in an Event-B context by using abstract sets, constants and axioms.

For example, each class is implicitly a subclass of the root class defined by the *Thing* abstract class, both modelled as sets. The *subclass relationship* is defined as a set inclusion relationship (encoding a subsumption relationship) between the corresponding sets to the subclass and the mother class, and the equivalence relationship is defined in Event-B using the set equality relationship between the corre-

sponding sets to the equivalent classes. The union combination of two classes is modeled in Event-B as the set union of the two sets corresponding to the two classes. To get all formalisation rules defined for the shallow modelling process, the reader may refer to this reference [9].

By applying some of this formalisation rules to the diplomas ontology described in section 3, we get the following Event-B context (figure 4).

CONTEXT *Ontology*
SETS
Thing
CONSTANTS
Phd Master Engineer Diplom Bachelor Diplomas_For_Phd
AXIOMS
axm1 : $Diplom \subseteq Thing$
axm2 : $Bachelor \subseteq Diplom$
axm3 : $Master \subseteq Diplom$
axm4 : $Engineer \subseteq Diplom$
axm5 : $Engineer = Master$
axm6 : $Phd \subseteq Diplom$
axm7 : $Diplomas\_For\_Phd = (Engineer \cup Master)$
END

Figure 4: Event-B context for diplomas : shallow modelling

4.2 Deep modelling: Ontologies as instances of ontology models

The approach that uses deep modelling consists in modelling the ontology concepts together with the concepts of the modelling language that were used to define the ontology concepts [6, 7]. Here, ontologies are defined as instances of ontology models. Two steps are required. First, an ontology model is formalised and then ontologies are defined as specific models corresponding to the defined ontology model. In our approach, we consider that both ontology modelling concepts and ontologies are explicitly modelled.

We have used the Event-B method to formalise these concepts. More precisely, as we consider ontologies as theories, we have used Event-B contexts to formalise such concepts. Classes, properties, instances and values are defined by the $CLASS$, $PROPERTY$, $INSTANCE$ and $VALUE$ carrier sets. These sets are abstractly defined, they are populated when defining specific ontologies.

Several relationships available in ontology modelling languages have been formalised. We have modelled *subclass* as a relation between classes. A set IS_A gathers the possible *subclass* relations between classes. A second part of this definition describes the constraints associated to inheritance i.e. inclusion of sets of instances. Indeed, in *axm*2 of figure 5, 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.

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. Second, the axiom *axm*3 states that the defined relation is reflexive, symmetric and transitive.

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 (axiom *axm*4 of figure 5).

To obtain the definitions of all the formalisation rules defined for the deep modelling process, the reader may refer to these references [6, 7].

CONTEXT *Ontology_Model*
SETS
CLASS PROPERTY INSTANCE VALUES …
CONSTANTS
HAS_INSTANCES … IS_A … EQUIVALENCE … UNION_OF …
AXIOMS
axm1 : $HAS\_INSTANCES = CLASS \leftrightarrow INSTANCE$
axm2 : $IS\_A = \{IsA | IsA \in CLASS \leftrightarrow CLASS \wedge (\forall x,y \cdot (x \in CLASS \wedge y \in CLASS \wedge x \mapsto y \in IsA \Leftrightarrow union(\{r \cdot r \in HAS\_INSTANCES | ran(\{x\} \lhd r)\}) \subseteq union(\{r \cdot r \in HAS\_INSTANCES | ran(\{y\} \lhd r)\})))\}$
axm3 : $EQUIVALENCE = \{EQo | EQo \in CLASS \leftrightarrow CLASS \wedge (\forall x \cdot (x \in CLASS \Rightarrow x \mapsto x \in EQo)) \wedge (\forall x,y \cdot (x \in CLASS \wedge y \in CLASS \wedge x \mapsto y \in EQo \Rightarrow y \mapsto x \in EQo)) \wedge (\forall x,y,z \cdot (x \in CLASS \wedge y \in CLASS \wedge z \in CLASS \wedge x \mapsto y \in EQo \wedge y \mapsto z \in EQo \Rightarrow x \mapsto z \in EQo))\}$
axm4 : $UNION\_OF = \{unionOf | (unionOf \in (\mathbb{P}(CLASS) \times \mathbb{P}(CLASS) \leftrightarrow CLASS)) \wedge (\forall x,y,z \cdot (x \in \mathbb{P}(CLASS) \wedge y \in \mathbb{P}(CLASS) \wedge z \in CLASS \wedge x \mapsto y \mapsto z \in unionOf \Rightarrow \forall instance \cdot (instance \in INSTANCE \Rightarrow \exists hasInstance \cdot (hasInstance \in HAS\_INSTANCES \Rightarrow (\forall n,m \cdot (n \in x \wedge m \in y \wedge (n \mapsto instance \in hasInstance \vee m \mapsto instance \in hasInstance)) \Rightarrow z \mapsto instance \in hasInstance)))))\}$
axm_i : …
END

Figure 5: Event-B generic context for ontology : deep modelling

In figure 6, we give an extract of the ontology of diplomas we have formalised as instances of the generic concepts previously introduced. The defined ontology illustrates the $subClassOf$, $Equivalence$ and $UnionOf$ relationships.

CONTEXT *Diplomas_Ontology*
EXTENDS *Ontology_Model*
CONSTANTS
Diplom Bachelor Master Engineer Phd Diplomas_For_Phd
isA eQ unionOf
AXIOMS
axm1 : $partition(CLASS, \{Diplom\}, \{Bachelor\}, \{Master\}, \{Engineer\}, \{Phd\}, \{Diplomas\_For\_Phd\})$
axm2 : $isA = \{Master \mapsto Diploms, Bachelor \mapsto Diploms, Engineer \mapsto Diploms, Phd \mapsto Diploms\}$
axm3 : $eQ = \{Bachelor \mapsto Bachelor, Master \mapsto Master, Engineer \mapsto Engineer, Phd \mapsto Phd, Master \mapsto Engineer, Engineer \mapsto Master\}$
axm4 : $unionOf = \{\{Master\} \mapsto \{Engineer\} \mapsto Diplomas\_For\_Phd\}$
axm_i : …
THEOREMS
thm1 : $isA \in IS\_A$
thm2 : $eQ \in EQUIVALENCE$
thm3 : $unionOf \in UNION\_OF$
END

Figure 6: Event-B context for diplomas : deep modelling

5 The OntoEventB plug In

The OntoEventB plug-in [10] has been developed to automatically support the translation of ontologies models, described using ontology description languages such as OWL [11] or PLIB [8], into Event-B Contexts [1]. It takes as input an ontology description file and generates, according to the selected approach (shallow or deep), the corresponding Event-B contexts. The OntoEventB plug-in is developed according to an architecture composed of three components: Input, Pivot and Output Models (Figure 7).

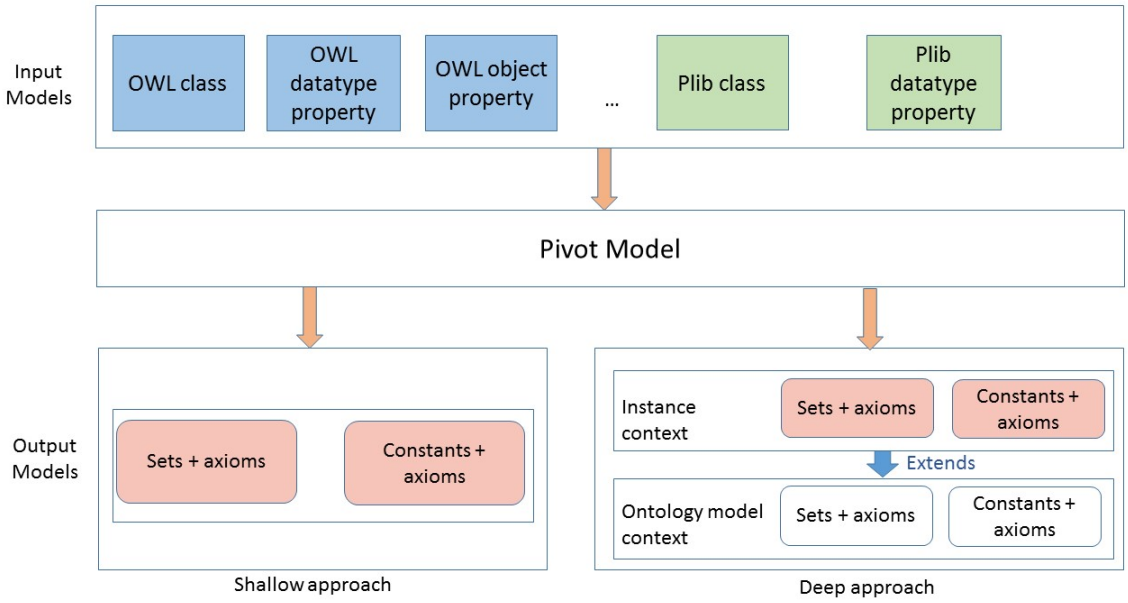

Figure 7: The OntoEventB internal architecture.

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

The Pivot Model component. This component is an intermediate operational model, which summarizes the common relevant concepts used by ontology description languages (classes, properties and data types). It defines generic concepts that integrate all specific concepts that can be received from the Input Model component. The Pivot Model can be extended to integrate other generic concepts that can be identified if a new language is added as input model in the Input Models component.

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

The Output Model component. This component has as input the generic concepts computed by the Pivot Model component and translates them into Event-B Context elements (sets, constants and axioms). This process uses transformation rules that formalise each ontological concept by an Event-B definition following the two approaches proposed and described in section 4 (Shallow and Deep modelling approaches). The user of the OntoEventB plug-in can choose one of them.

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

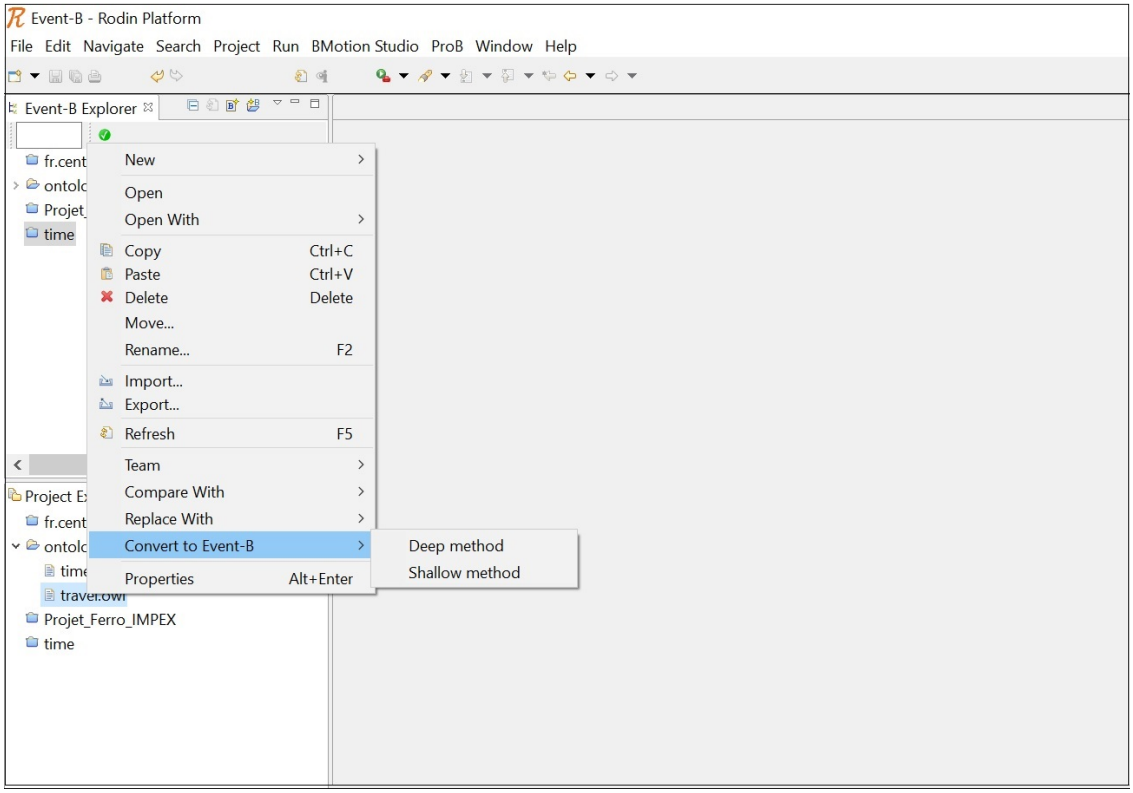

Figure 8: The OntoEventB submenu.

Installing and Using OntoEventB plug-in. The OntoEventB tool is developed as an Eclipse plug-in to integrate it into a Rodin platform [2], an IDE (Integrated Development Environment) supporting Event-B developments. To use OntoEventB plug-in in your Rodin platform instance, you must install the plug-in by using the Install New Software menu item[1] for downloading and installing the plug-in automatically.

After installing the OntoEventB plug-in in a Rodin platform instance, the convert to Event-B submenu becomes available by right clicking on an owl file (with an `.owl` extension) in the project explorer

[1] OntoEventB update site : `http://wdi.supelec.fr/OntoEventB-update-site/`

as shown in Figure 8. It proposes to se up the two modelling techniques: deep and shallow corresponding to the two proposed approaches we introduced section 4.

6 Conclusion

This paper reports on some of the results of the French ANR IMPEX research project. We have discussed the interest of making explicit domain knowledge in design models in order to strengthen them. We also proposed a straightforward approach formalizing ontologies as theories encoded within Event-B contexts. This approach led to the development of Plug-In that produces automatically Event-B contexts from ontologies expressed in different ontology models.

Moreover, the previous work achieved in this project showed the interests of the approach to strengthen models in different areas. We have applied the developed approach to case studies issued from avionic systems, medical devices and electronic voting systems.

This work is still an on-going work. We are currently investigating the possibility to formalise ontologies of behaviours (e.g. ontologies of services) and their use to annotate behavioural components of design models (e.g. events of an Event-B model). First results are already available on plastic interfaces [5].

Other investigations consider design system models re-factoring with the objective of handling explicitly domain knowledge in order to support the verification of new properties mined from domain models.

References

[1] Jean-Raymond Abrial (2010): *Modeling in Event-B - System and Software Engineering*. Cambridge University Press, doi:10.1017/CBO9781139195881.

[2] Jean-Raymond Abrial, Michael J. Butler, Stefan Hallerstede, Thai Son Hoang, Farhad Mehta & Laurent Voisin (2010): *Rodin: an open toolset for modelling and reasoning in Event-B*. *STTT* 12(6), pp. 447–466, doi:10.1007/s10009-010-0145-y.

[3] Yamine Aït Ameur, J. Paul Gibson & Dominique Méry (2014): *On Implicit and Explicit Semantics: Integration Issues in Proof-Based Development of Systems*. In: *Leveraging Applications of Formal Methods, Verification and Validation. Specialized Techniques and Applications - 6th International Symposium, ISoLA 2014, Imperial, Corfu, Greece, Proceedings, Part II*, doi:10.1007/978-3-662-45231-8_50.

[4] Yamine Aït Ameur & Dominique Méry (2016): *Making explicit domain knowledge in formal system development*. *Sci. Comput. Program.* 121, pp. 100–127, doi:10.1016/j.scico.2015.12.004.

[5] Abdelkrim Chebieb & Yamine Aït Ameur (2015): *Formal Verification of Plastic User Interfaces Exploiting Domain Ontologies*. In: *2015 International Symposium on Theoretical Aspects of Software Engineering, TASE 2015, Nanjing, China, September 12-14, 2015*, doi:10.1109/TASE.2015.25.

[6] Kahina Hacid & Yamine Aït Ameur (2016): *Annotation of Engineering Models by References to Domain Ontologies*. In: *Model and Data Engineering - 6th International Conference, MEDI 2016, Almería, Spain, September 21-23, 2016, Proceedings*, pp. 234–244, doi:10.1007/978-3-319-45547-1_19.

[7] Kahina Hacid & Yamine Aït Ameur (2016): *Strengthening MDE and Formal Design Models by References to Domain Ontologies. A Model Annotation Based Approach*. In Tiziana Margaria & Bernhard Steffen, editors: *Leveraging Applications of Formal Methods, Verification and Validation: Foundational Techniques - 7th International Symposium, ISoLA 2016, Imperial, Corfu, Greece, October 10-14, 2016, Proceedings, Part I*, *Lecture Notes in Computer Science* 9952, pp. 340–357, doi:10.1007/978-3-319-23781-7_8.

[8] ISO (1998): *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.

[9] Linda Mohand-Oussaïd & Idir Aït-Sadoune (2017): *Formal Modelling of Domain Constraints in Event-B*. In: *Model and Data Engineering - 7th International Conference, MEDI 2017, Barcelona, Spain, October 4-6, 2017, Proceedings*, pp. 153–166, doi:10.1007/978-3-319-66854-3_12.

[10] Linda Mohand Oussaïd & Idir Ait-Sadoune (2017): *OntoEventB : Un outil pour la modélisation des ontologies dans B Événementiel*. In: *AFADL 2017*, Montpellier, France, pp. 117–121.

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

[12] David Simon Zayas, Anne Monceaux & Yamine Aït Ameur (2010): *Knowledge Models to Reduce the Gap between Heterogeneous Models: Application to Aircraft Systems Engineering*. In Radu Calinescu, Richard F. Paige & Marta Z. Kwiatkowska, editors: *15th IEEE International Conference on Engineering of Complex Computer Systems, ICECCS 2010, Oxford, United Kingdom, 22-26 March 2010*, IEEE Computer Society, doi:10.1109/ICECCS.2010.35.