Domain knowledge as Ontology-based Event-B Theories

I. Mendil[1], Y. Aït-Ameur[1], N. K. Singh[1], D. Méry[2], and P. Palanque[3]

[1]INPT-ENSEEIHT/IRIT, University of Toulouse, France
[2]Telecom Nancy, LORIA, Université de Lorraine, France
[3]IRIT, Université de Toulouse, France
`{ismail.mendil,yamine,nsingh}@enseeiht.fr, dominique.mery@loria.fr, palanque@irit.fr`

1 Context of the study

In general system engineering approaches, particularly formal methods, do not offer specific constructs allowing the designer to define formal models of domain knowledge, nor mechanisms allowing to import such existing models. However, there exist formal modelling languages and/or meta-models sometimes standardised [6] that support the formalisation of such domain knowledge.

In this paper, we show how Event-B theories [1,3,5] can be defined to formalise such domain knowledge and the Rodin Platform [2] is used to carry out the formal development and the verification process. We first give a generic theory defining an ontology modelling language and then show its instantiation in the case of the ARINC 661 standard describing interactive cockpits as an example of critical interactive systems.

This work has been achived in the context of the French national research agency (ANR) project FORMEDICIS [7][1] FORmal MEthods for the Development and the engIneering of Critical Interactive Systems

2 A theory for ontologies

Since we are interested in formalising the domain knowledge associated to critical interactive interfaces and use the domain properties in our Event-B models, we need a framework to express such knowledge.

In our case, domain knowledge is formalised using ontologies. Therefore, as a first step, we have developed a generic theory allowing to describe ontologies. An extract of this theory is given in Listing 1. Classes `C`, properties `P` and instances `I` are defined as type parameters and a set of other relevant operators is provided.

`OntologiesTheory` entails several useful theorems thanks to the definition of the operators. `thm1` is an example, of a theorem establishing the transitivity of the `isA` operator. Another example is `thm2` which is trivial but has a great benefit for discharging poof-obligations in Event-B models.

[1] `https://anr.fr/Projet-ANR-16-CE25-0007`

```
THEORY OntologiesTheory
TYPE PARAMETERS C,P,I
DATA TYPES Ontology(C,P,I)
CONSTRUCTORS consOntology(classes:ℙ(C),properties:ℙ(P),instances:ℙ(I),
 classProperties:ℙ(C×P),classInstances:ℙ(C×I),classAssociations:ℙ(C×P×C)
     , instanceAssociations:ℙ(I×P×I))
OPERATORS
isWDInstancesAssociations <predicate> (o: Ontology(C, P, I) ...
getInstanceAssociations <expression> (o: Ontology(C, P, I))
well−definedness isWDInstancesAssociations(o) ...
isWDOntology <predicate> (o: Ontology(C, P, I))
direct definition isWDClassProperites(o) ∧ isWDClassInstances(o) ∧
    isWDClassAssociations(o) ∧ isWDInstancesAssociations(o)
isA <predicate>(o: Ontology(C, P, I),c1: C,c2: C)
well−definedness isWDOntology(o),ontologyContainsClasses(o, {c1, c2})
direct definition getInstancesOfaClass(o,c1)⊆getInstancesOfaClass(o,c2)
addInstancesToAClass <expression> (o: Ontology(C, P, I),c: C,ii: ℙ(I))
well−definedness isWDOntology(o),ontologyContainsClasses(o, {c}),
    ontologyContainsInstances(o, ii),¬ classContainsInstances(o, c, ii)
direct definition  consOntology(getClasses(o), getProperties(o),
getInstances(o),getClassProperties(o),getClassInstances(o) ∪ ({c} × ii),
     getClassAssociations(o), getInstanceAssociations(o))
isVariableOfOntology <predicate> (o:Ontology(C,P,I),ipvs:ℙ(I×P×I))
well−definedness isWDOntology(o)
direct definition ipvs ⊆ { i1 ↦ p ↦ i2 | i1 ∈ I ∧ p ∈ P ∧ i2 ∈ I ∧
i1 ↦ p ↦ i2 ∈ instances(o) × properties(o) × instances(o) ∧
    (∃c1, c2 · c1 ∈ C ∧ c2 ∈ C ∧ {c1, c2} ⊆ getClasses(o) ⇒
     (c1↦p↦c2∈getClassAssociations(o)∧p∈getClassProperties(o)[{c1}] ∧
        i1∈getClassInstances(o)[{c1}]∧i2∈getClassInstances(o)[{c2}]))}
THEOREMS
thm1: ∀o, c1, c2, c3· o ∈ Ontology(C, P, I) ∧ isWDOntology(o) ∧ c1 ∈ C ∧
     c2 ∈ C ∧ c3 ∈ C ∧ ontologyContainsClasses(o, {c1, c2, c3})
         ⇒ (isA(o, c1, c2) ∧ isA(o, c2, c3) ⇒ isA(o, c1, c3))
thm2: ∀o, cs1, cs2 · o ∈ Ontology(C, P, I) ∧ isWDOntology(o) ∧ cs1 ⊆ C ∧
     cs2 ⊆ C ∧ cs1 ≠ ∅ ∧ cs2 ≠ ∅ ∧ ontologyContainsClasses(o, cs1) ∧
    ontologyContainsClasses(o,cs2)
         ⇒(ontologyContainsClasses(o,cs1∪cs2))
...
END
```

Listing 1: Ontology Modelling Language Data Type

3 The case of Arinc 661

ARINC 661 [4] defines a standard Cockpit Display System (CDS) interface intended for all types of aircraft installations. The primary objective is to minimize the cost to the airlines, directly or indirectly. It normalises the definition of cockpit display system (CDS) interface and the communication protocol with user applications. In particular, its objective is to

– minimize the cost of acquiring new avionic systems to the extent it is driven by the cost of CDS development;
– minimize the cost of adding new display function to the cockpit during the life of an aircraft;
– minimize the cost of managing hardware obsolescence in an area of rapidly evolving technology;
– introduce interactivity to the cockpit, thus providing a basis for airframe manufacturers to standardize the Human Machine Interface (HMI) in the cockpit.

he standard defines two external interfaces between the CDS and the aircraft systems. The first is the interface between the avionics equipment (user systems) and the display system graphics generators. The second is a means by which symbology and its related behavior are defined.

3.1 ARINC 661 Concepts Declaration

We have considered the ARINC 661 specification document aiming to describe specific case studies —weather radar system. We have identified a set of relevant concepts, after a thorough analysis of the specification document. The formalisation of the ARINC 661 proceeds by instantiating `OntologiesTheory`, it yields the Event-B theory `ARINC661Theory` in Listings 2, 3 and 4. Retained concepts are often ARINC 661 widgets like `Label`, `CheckButton`, etc. However other elements are introduced for organisation purposes where the widgets may be used like `wellBuiltClassProperties` and `wellBuiltTtypesElements`.

```
THEORY ARINC661Theory
IMPORT THEORY PROJECTS OntologiesTheory
AXIOMATIC DEFINITIONS  ARINC661Axiomatisation:
TYPES ARINC661Classes, ARINC66Properties, ARINC661Instances
OPERATORS
ARINC661_BOOL <expression> () : ARINC661Classes
A661_TRUE <expression> () : ARINC661Instances
A661_FALSE <expression> () : ARINC661Instances
CheckButtonStateClass <expression> () : ARINC661Classes
Label <expression> () : ARINC661Classes
A661_EDIT_BOX_NUMERIC_VALUES_CLASS <expression> () :
     ARINC661Classes
RadioBox <expression> () : ARINC661Classes
CheckButton <expression> () : ARINC661Classes
EditBoxNumeric <expression> () : ARINC661Classes
hasChildrenForRadioBox <expression> () : ARINC66Properties
hasCheckButtonState <expression> () : ARINC66Properties
hasValue <expression> () : ARINC66Properties
SELECTED <expression> () : ARINC661Instances
UNSELECTED <expression> () : ARINC661Instances
wellBuiltClassProperties<expression>():ℙ(ARINC661Classes×ARINC66Properties)
wellBuiltClassAssociations <expression> () : ℙ(ARINC661Classes ×
     ARINC66Properties × ARINC661Classes)
wellbuiltTypesElements<expression>():ℙ(ARINC661Classes×ARINC661Instances)
isWDRadioBox <predicate> (o: Ontology(ARINC661Classes,
     ARINC66Properties, ARINC661Instances) ) :
well−definedness isWDOntology(o)
isWDARINC661Ontology <predicate> (o: Ontology(ARINC661Classes,
     ARINC66Properties, ARINC661Instances) ) :
consARINC661Ontology <expression> (ii: ℙ(ARINC661Instances),
cii: ℙ(ARINC661Classes×ARINC661Instances),
ipvs:ℙ(ARINC661Instances ×ARINC66Properties×ARINC661Instances)):
Ontology(ARINC661Classes, ARINC66Properties, ARINC661Instances)
well−definedness isWDARINC661Ontology(consOntology(ARINC661Classes,
ARINC66Properties,ii,wellBuiltClassProperties,wellbuiltTypesElements∪cii
,wellBuiltClassAssociations, ipvs))
isVariableOfARINC661Ontology <predicate> (o: Ontology(ARINC661Classes,
ARINC66Properties, ARINC661Instances),
ui: ℙ(ARINC661Instances × ARINC66Properties × ARINC661Instances)) :
well−definedness isWDOntology(o)
...
```

Listing 2: ARINC 661 theory declarations

3.2 ARINC 661 Concepts Definition

Since ontology standards do not define explicitly operators (they rely on ad'hoc APIs on their XML representation) to manipulate the concepts they describe, we have defined a set of operators allowing to manipulate the concepts of the `ARINC661Theory`. Moreover, the definition of the concepts are given in the shape of axioms. In particular, the effective type parameters for ontology instantiation are defined in *ARINC661ClassesDef*, *ARINC661PropertiesDef* and *ARINC661InstancesDef*. Also, `consARINC661Ontology` is provided for a valid construction of operator under the condition —formalised as well-definedness condition—that the arguments are valid. In addition, `isWDARINC661Ontology` allows the checking of the validity of a given ARINC 661 ontology.

```
AXIOMS
ARINC661ClassesDef: partition(ARINC661Classes, {ARINC661_BOOL},
    {ARINC661_STRING_CLASS}, {Label},{RadioBox}, {CheckButton}, ...)

ARINC66PropertiesDef: partition(ARINC66Properties,  {hasVisible},
    {hasEnable},{hasAnonymous},{hasChildrenForRadioBox}, ...)
ARINC661InstancesDef: partition(ARINC661Instances, {A661_TRUE},
    {A661_FALSE},{A661_TRUE_WITH_VALIDATION}, LabelInstances, ...)
wellBuiltClassProperties:
wellBuiltClassProperties = ({Label} × {hasVisible,...})    ∪
    ({RadioBox} × {hasWidgetType, hasParentIdent, hasVisible, ...}) ∪
    ({CheckButton}×{hasWidgetType,hasVisible,hasEnable, ...})∪ ...)
consARINC661Ontology: ∀ii, cii, ipvs · ii ∈ ℙ(ARINC661Instances) ∧
    cii ∈ ℙ(ARINC661Classes × ARINC661Instances) ∧
    ipvs ∈ ℙ(ARINC661Instances×ARINC66Properties × ARINC661Instances) ∧
    wellbuiltTypesElements ∩ cii = ∅ ∧ ii ⊆ WidgetsInstances
    ⇒ consARINC661Ontology(ii, cii, ipvs) =
            consOntology(ARINC661Classes, ARINC66Properties, ii,
                wellBuiltClassProperties, wellbuiltTypesElements ∪ cii,
                wellBuiltClassAssociations, ipvs))
isWDEditBoxNumeric:
    ∀o·o∈ Ontology(ARINC661Classes,ARINC66Properties,ARINC661Instances)⇒
    (isWDRadioBox(o)⇔((∀ed,v·ed↦hasValue↦v∈getInstanceAssociations(o)
    ⇒v ∈ A661_EDIT_BOX_NUMERIC_ADMISSIBLE_VALUES)     )
isWDARINC661Ontology:
∀o· o ∈ Ontology(ARINC661Classes, ARINC66Properties, ARINC661Instances)
    ⇒(isWDOntology(o)∧isWDRadioBox(o)∧isWDEditBoxNumeric(o)⇒
        isWDARINC661Ontology(o))
...
```

Listing 3: ARINC 661 theory definitions

3.3 ARINC 661 theory Theorems

Last, the most important part concerns the properties embedded in the theory in the form of theorems. They are particularly useful to formalise standard requirements. Moreover, the validation of the fact that the ontology has the right structure is done through the theorems `thm1` and `thm2`. There are two important properties ensuring that the structure of the ontology is valid: the classes are related to properties already defined and similarly that the class associations component encompasses only the provided classes and properties. The theorem proofs are discharged thanks to the definition of `wellBuiltClassProperties`, `wellBuiltClassAssociations` and `wellBuiltTypesElements`

```
THEOREMS
thm1: ∀ii, cii, ipvs ·
ii ∈ ℙ(ARINC661Instances) ∧ cii ∈ ℙ(ARINC661Classes×ARINC661Instances)∧
ipvs ∈ ℙ(ARINC661Instances × ARINC66Properties × ARINC661Instances) ∧
wellbuiltTypesElements ∩ cii = ∅ ∧ ii ⊆ WidgetsInstances
    ⇒ isWDClassProperites(consARINC661Ontology(ii, cii, ipvs))
thm2: ∀ii, cii, ipvs ·
ii ∈ ℙ(ARINC661Instances) ∧ cii∈ ℙ(ARINC661Classes×ARINC661Instances) ∧
ipvs ∈ ℙ(ARINC661Instances× ARINC66Properties×ARINC661Instances) ∧
wellbuiltTypesElements ∩ cii = ∅ ∧ ii ⊆ WidgetsInstances
    ⇒ isWDClassAssociations(consARINC661Ontology(ii, cii, ipvs))
END
```

Listing 4: ARINC 661 theory theorems

4 Conclusion

This approach shows that axiomatising domain knowledge as ontologies expressed in Event-B theories is a suitable solution to handle standard requirements in system design. The defined theory for ARINC 661 standard specification has been used to develop Event-B models for several case studies like WXR user interface and TCAS application. We have used the defined data types to type state variables. Axioms and theorems have been used to prove specific properties on these case studies. The ontology description theory is presented as playing a the role of scaffolding for producing a domain-specific theories thanks to the Event-B theories featuring type genericity.

Due to the complexity of the theories developed for the aforementioned objective, we reported a serious bug to the development team of Plug-in Theory which was fixed and integrated in a future release. All Event-B developments are available and interested reader may contact the first author for a copy.

References

1. Abrial, J.R.: Modeling in Event-B: system and software engineering. Cambridge University Press (2010)
2. Abrial, J.R., Butler, M., Hallerstede, S., Hoang, T.S., Mehta, F., Voisin, L.: Rodin: An open toolset for modelling and reasoning in event-b **12**(6) (2010)
3. Abrial, J.R., Butler, M., Hallerstede, S., Leuschel, M., Schmalz, M., Voisin, L.: Proposals for mathematical extensions for Event-B. Tech. Rep. (2009)
4. ARINC: Arinc 661 specification: Cockpit display system interfaces to user systems, prepared by aeec, published by sae, 16701 melford blvd., suite 120, bowie, maryland 20715 usa (June 2019)
5. Butler, M., Maamria, I.: Mathematical extension in Event-B through the rodin theory component (2010)
6. Calegari, D., Mossakowski, T., Szasz, N.: Heterogeneous verification in the context of model driven engineering. Science of Computer Programming, Elsevier Journal. **126**, 3–30 (2016)
7. Formedicis, `https://anr.fr/Projet-ANR-16-CE25-0007`