Correct By Construction Standard Compliance/Conformance [1]

Yamine Aït Ameur*, **

* IRIT, INPT-ENSEEIHT - CNRS, University of Toulouse, France

22nd Software Certification Consortium
SCC'2024

May 9-10, 2024
Annapolis, Maryland, USA

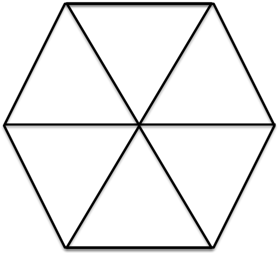
[1] Joint work with **G. Dupont, I. Mendil, D. Méry, M. Pantel P. Rivière** and **N. Singh**

This work is supported by ANR (Agence Nationale de la Recherche) grant ANR-19-CE25-0010-01

Outline

To start ...

Is this a **cube** or a **Polygon** ?



- Compute the `Volume` of this cube $\Longrightarrow$ Need of 3-D geometry

- Compute the `Surface` or `Volume` of this polygon $\Longrightarrow$ Need of 2-D geometry, but `Volume` has no meaning
- However, `Surface` and `Volume` of the cube are computable using 2-D geometry

Context

System models

- Different approaches used to model systems
  - stateful e.g. state-transition systems
  - stateless e.g synchronous languages
- **Prescriptive** models

Modelling languages

- Supported by different modelling languages
- Main objective $\Longrightarrow$ **reason** on system models to **establish properties** reflecting the modelled requirements.

Context

How rich is a modelling language from different perspectives ?

- expressivity
- semantics
- verification and validation capabilities
- ...

How modelling languages can be enriched ?

- Ad'hoc modelling languages, DSLs
- extension
- transformation
- composition
- ...

Encountered problems

Modelling requires to handle

- heterogenity
- domain knowledge and application domain
- standards and regulations
- ...

Context

State based formal methods

Capability of formal state-based methods

- to **model** complex systems
- **reason** about them to **establish properties** reflecting the modelled requirements.

In particular,

- ensuring system safety through the verification of invariant properties
- each reachable state of the modelled system fulfills these invariants, i.e. the **system state is always in a safe region and never leaves it**
- verification is based on an induction principle over traces of transition systems

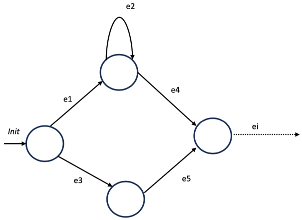


Figure: State-transition systems

Context

State based formal methods

State variables are modified by **actions** relying on

- the generalised assignment operation based on the *"becomes such that"* BAP
- noted $St :| BAP(St, St')$
- defining a state transition
- ASM rules, substitutions or events in B and Event-B, Hoare triples, Guarded Commands (GCL), operations in RSL and VDM , actions in TLA$^+$, schemas in Z, . . .

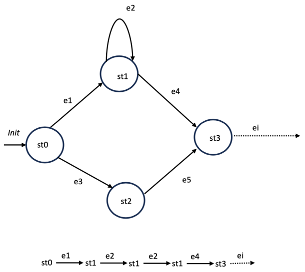


Figure: State-transition systems - Trace-based semantics

Context

Domain knowledge

Modelling of complex systems in system engineering relies on domain knowledge

- **shared** and **reused** in system models
- definitions as well as domain-specific properties.
- **descriptive** models

Two different types of Domains

- **Once and for all** formalised domains, **stable** and **reusable**
  - mathematics: diff. eq., control theory, probabilities, etc.
  - physics, flight dynamics, units, etc.
  - more generally, external theories related to designed systems
- **System dependent** formalised domains
  - describe system concepts
  - "instantiations", "specialisations" of above theories with additional specific constraints
  - Examples: valves, tanks, wheels, etc.

Standards, regulations, ontologies, ...

- defined **independently** of any **specific** system model
- designed asynchronously

Context

Domain knowledge

Modelling of complex systems in system engineering relies on domain knowledge

- formalised as **algebraic theories** with **data types, operators, axioms**
- Operators record **allowed transformations**
- and **theorems** proved *independently* of the designed system models.

Partial definitions play a key role

This idea is not new !

Data Type
T

Constructors
Constr_1 (...) : T
...
Constr_n (...) : T

Operators
Op_1 (x, x' : T, args : .....)
...
Op_n (x, x' : T, args : .....)

Figure: Algebraic data-types definitions

Model Annotation

Composing system models and domain knowledge - **Model Annotation**

How ?

- By **borrowing** domain specific theories in system design formal models. It brings
  - partial operators associated to **well-definedness** conditions
  - Hypotheses, Theorems and proof rules
- By **annotating** models with references to domain knowledge models

Transitions are seen as partially defined operations

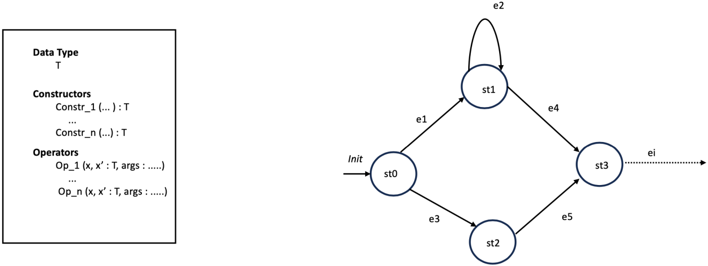


Figure: States and transitions linked to types and operators

The case of state-based formal methods with Event-B

Main features

Support for

- expressive data-types
- partially defined operators
- Well-definedness (WD) conditions
- Automatic proof obligation generation

The proposed framework

- Event-B can be extended to handle domain theories using its Theory component
- *Transfer and Reuse*, in the system design models, the proofs achieved on the theory side
- Keep using the Event-B invariant preservation mechanism while referring to externally defined data-types

Event-B Structure and Proofs

Event-B: Models and Machines

- a state based formal method with proof and refinement

Context	Machine
CONTEXT Ctx **SETS** s **CONSTANTS** c **AXIOMS** A **THEOREMS** T_{ctx} **END**	**MACHINE** M^A **SEES** Ctx **VARIABLES** x^A **INVARIANTS** $I^A(x^A)$ **THEOREMS** $T_{mch}(x^A)$ **VARIANT** $V(x^A)$ **EVENTS** **EVENT** evt^A **ANY** α^A **WHERE** $G^A(x^A, \alpha^A)$ **THEN** $x^A :\mid BAP^A($ $\alpha^A, x^A, x^{A'})$ **END** ... **END**

- set theory, basic types (integers, booleans) and their associated operators
- first order logic
- **explicit state** formalised as a set of state variables
- **initialisation event** and **guarded events** to record state changes based on **BAP** (Before-After Predicates)
- inductive reasoning on event traces
- invariant preservation and variant decreasing for reachability
- Rodin open source IDE

Event-B Structure and Proofs

Event-B: Proof Obligations

- a state based formal method with proof and refinement

(1)	Theorems (THMCtx)	$A \Rightarrow T_{ctx} A$
(2)	(THMMch)	$I^A(x^A) \Rightarrow T_{mch}(x^A)$
(3)	Initialisation (INIT)	$A \wedge G_A(\alpha^A) \wedge BAP^A(\alpha^A, x^{A\prime}) \Rightarrow I^A(x^{A\prime})$
(4)	Invariant preservation (INV)	$A \wedge I_A(x^A) \wedge G_A(x^A, \alpha^A) \wedge BAP^A(x^A, \alpha^A, x^{A\prime}) \Rightarrow I^A(x^{A\prime})$
(5)	Event feasibility (FIS)	$A \wedge I_A(x^A) \wedge G^A(x^A, \alpha^A) \Rightarrow \exists x^{A\prime} \cdot BAP^A(x^A, \alpha^A, x^{A\prime})$
(6)	Variant progress (VAR)	$A \wedge I^A(x^A) \wedge G^A(x^A, \alpha^A) \wedge BAP^A(x^A, \alpha^A, x^{A\prime}) \Rightarrow V(x^{A\prime}) < V(x^A)$

- Automatic generation of proof obligations.
- Rodin is equiped with automatic/interactive provers, SMT solvers, Model checkers, animators, etc.

Proof obligations

- A set of well-definedness POs are associated to Event-B constructs

Event-B theories

Core Event-B is not equiped with

- rich data-types and associated operators
- in particular, there is no
  - **reals NOR continuous features**
  - **capability to introduce new data types**
  - **possibility to generate new proof obligations**

Event-B theories as a support for Event-B extensions

- introduced in 2010's by JR. Abrial, M. Butler, I. Maamria, ...
- support for defining new data types
- constructive or axiomatic
- tool supported as a PlugIn of the Rodin platform

Event-B theories

THEORY Th **IMPORT** Th1, ...

TYPE PARAMETERS E, F, ...

DATATYPES
T1$(E, \ldots) \equiv$ **cstr1**$(p_1 : T_1, \ldots)|\ldots$

OPERATORS
Op1 **nature** $(p_1 : T_1, \ldots)$
well—definedness
$WD(p_1, \ldots)$
direct definition
$Expr_1$
...

AXIOMATIC DEFINITIONS OPERATORS
AOp2 **nature** $(p_1 : T_1, \ldots) : T_r$
well—definedness $WD(p_1, \ldots)$
$WD(p_1, \ldots)$
...

AXIOMS
Ax_1, ...

THEOREMS
Th_1, ...

END

- **Algebraic Theories** as extensions for Event-B basic language
- **Data types**, **operators** with **WD conditions**
- Constructive definitions and axiomatic definitions
- **Relevant Theorems**
- proof rules: Inference and rewrite rules
- *Theory Plug-in* development environment and associated proof environement
- Proof rules can be included in the Rodin proof tactics
- Library for mathematical and domain-specific theories (i.e., Reals, differential equations etc.)

Well-Definedness (WD)

Event-B: Well-definedness Proof obligations

According to J.R. Abrial, Well-Definedness describes the

circumstances under which it is possible to introduce new term symbols by means of conditional definitions in a formal theory as if the definitions in question were unconditional, . . . It avoids describing ill-defined operators, formulas, axioms, theorems, and invariants.

- **Avoidance of ill-defined operators**, formulas, axioms, theorems, and invariants.
- **Each formula is associated to well-definedness POs** that ensure that the formula is well-defined and that *two-valued logic can be used* (M. Leuschell - IFM'2020).

- An inductively defined WD predicate $WD(f)$ is associated with each formula f

- **Example.** Let a and b be two integers, $f \in D \rightarrowtail R$, then
  - $WD(a \div b) \equiv WD(a) \wedge WD(b) \wedge b \neq 0$
  - $WD(f(a)) \equiv WD(a) \wedge f \in D \rightarrowtail R \wedge a \in \mathrm{dom}(f)$

Well-Definedness (WD)

Event-B Theories: Well-definedness Proof obligations

- Each **defined operator is associated to a (WD) condition** ensuring its correct definition.
- When it is **applied** (in the theory or in an Event-B machine or context), this **WD condition generates a PO** requiring to establish that this condition holds

- The theory **designer defines these WD conditions for the partially defined operators**.
- They are then added to the native Event-B WD POs
- Once the WD POs are proved, they are added as hypotheses in the proofs of the other POs

New proof obligations

- **Use of the WD mechanism**
- **When an operator is defined/applied, its WD PO is automatically generated**

Principle

State changes and Invariants

- **State change (transition)** is viewed as a **partial function**

  $Trans : State \nrightarrow \mathbb{P}(State)$ or $Trans : State \nrightarrow State$

- An **invariant restricts state changes** to **safe** states
  - A *well-defined* partial function, on the set of safe states $Safe_{St}$ as

    $$Trans_{Inv} : Safe_{St} \nrightarrow \mathbb{P}(State)$$

  - To **preserve the invariant**, one has to establish that:

    $$\mathbf{ran}(Trans_{Inv}) \subseteq \mathbb{P}(Safe_{St})$$

.

An alternative approach to prove invariants of Event-B system models

- A data type T for $State$ + operators $\Rightarrow$ well-defined partial functions
- Each operator $\boldsymbol{Op}(x_1 : \boldsymbol{T}_1, x_2 : \boldsymbol{T}_2, \ldots, x_n : \boldsymbol{T}_n)$ of type T is associated to a WD$(x_1, x_2, \ldots, x_n)$ stating that $x_1, x_2, \ldots, x_n \in \mathbf{dom}(Op)$
- Safe state changes are defined according to a given property *independently* of any model

Principle

Step 1. Definition of an algebraic data-type

Formalise domain knowledge

- a data-type corresponding to the type of a state variable
- a set of operators associated to well-Defined (WD) conditions
- relevant theorems guaranteeing properties of the data-type
- $\Longrightarrow$ The theorems are proved once and for all.

Step 2. Formalise a system model

Annotate states and transitions of system models

- State variables are **typed** using the defined data-type $\Longrightarrow$ State variables annotation
- In Events, state variables are manipulated using the operators associated to the data-type $\Longrightarrow$ Events and transitions annotation
- **WD POs** associated to the operators are **automatically generated**
- Theorems of the theory hold at the machine level (for free)

A generic algebraic theory for the manipulated type

An algebraic theory for (parameterised) data-type T

- A set of operators manipulating data-type $T(ArgsType)$ elements
- Specific properties associated to the data-type defined as a predicate

```
THEORY  Theo
TYPE PARAMETERS   ArgsTypes
DATATYPES
    T(ArgsTypes) ≡ Cons(args : ArgsTypes) | . . .
OPERATORS
    . . .
    Op_i   predicate  (el : T(ArgsType), args : ArgsTypes)
        well−definedness condition  WD_Op_i(args)
        direct definition  Op_Exp_i(el, args)
    . . .
    WD_Op_i  predicate  (args : ArgsTypes)
        direct definition  WD_Op_Exp_n(el, args)
    . . .
   Properties  predicate  (el : T(ArgsTypes))
        direct definition  properties_Exp(el)
  . . .
THEOREMS
    ThmTheoOp_1 :
        ∀x, args · x ∈ T(ArgsTypes) ∧ args ∈ ArgsTypes∧
          WD_Op_1(args) ∧ Op_1(x, args) ⇒ Properties(x)
        . . .
    ThmTheoOp_n :
        ∀x, args · x ∈ T(ArgsTypes) ∧ args ∈ ArgsTypes∧
          WD_Op_n(args) ∧ Op_n(x, args) ⇒ Properties(x)
```

Theorems

- guarantee that an operation does not move to a situation that does not satsify the *Properties* predicate

Proofs

- Theorems must be proved for all the operators that preserve the properties
- Proofs are made using the provers offered by the framework (may be external ones)

An instantiation context of the generic theory

An Event-B context with

- A data-type $T(s)$
- Instantiated theorems

CONTEXT Ctx
SETS s
CONSTANTS c
AXIOMS ...
THEOREMS
ThmTheoOp$_1$Inst : $\forall x, args \cdot x \in T(s) \land args \in s \land$
$\quad$ WD_Op$_1(args) \land$ Op$_1(x, args) \Rightarrow$ Properties(x)
...
ThmTheoOp$_n$Inst : $\forall x, args \cdot x \in T(s) \land args \in s \land$
$\quad$ WD_Op$_n(args) \land$ Op$_n(x, args) \Rightarrow$ Properties(x)

Proofs

- Theorems are trivially proved (type checking)

An Event-B model manipulating data-type elements

An Event-B Machine with

- a state variable x of type $T(s)$ and specific invariants
  - a typing invariant
  - by *AllowedOper*, **only** defined operators manipulate state x provided their WD hold

```
MACHINE      Machine  SEES   Ctx
VARIABLES    x

INVARIANTS
 TypingInv  : x ∈ T(s)
 AllowedOper: ∃args · args ∈ s ∧ (WD_Op1(args) ∧ Op1(x, args)) ∨ ... ∨ (WD_Opn(args) ∧ Opn(x, args))

THEOREMS
 SafThm     : Properties(x)
EVENTS
   Evt1 ≙                     ...        Evtn  ≙
      ANY  α                                ANY  α
      WHEN                                  WHEN
         grd1 : α ∈ s ∧ WD_Op1(α)              grd1 : α ∈ s ∧ WD_Opn(α)
      THEN                                  THEN
         act1 : x :| Op1(x′, α)                act1 : x :| Opn(x′, α)
      END                                   END
END
```

- The **SafThm** theorem states that the properties **Properties** hold
- Its proof is straightforward: use of the instantiated theorems *ThmTheoOp$_i$Inst* (context) + the modus-ponens ($\Rightarrow$-elimination rule)

Domain models as ontologies

Design of Critical Interactive Systems

- Descriptive standards with rules and regulations for CIS
- Designers must conform to the standards
- ARINC[2] 661 standard describes Cockpit Display System (CDS) standard for communication protocols between interface objects and aircraft systems
  - Widespread use in the industry ⇒ i.e., Airbus A380 and Boeing B787.
  - 800 pages of definitions and requirements for the CDS and its graphical objects
  - In Our work ⇒ we focus on the widget aspects (chapter 3.0).

Several developed case studies: Weather Radar System (WXR), Traffic Collision Avoidance System (TCAS), ...

Modelling ARINC661 as an ontology

- Need to define ontologies in Event-B

  ⇒ Define an Event-B Theory
- CIS components
  - are instances of the ontology
  - manipulated by Event-B models

[2] Aeronautical Radio, Incorporated

Ontologies as Event-B theories

Ontologies Modelling Language (OML) - DATATYPE

THEORY *OntologiesTheory*
TYPE PARAMETERS
C ,
P ,
I
DATATYPES
$\text{Ontology}(C, P, I) \equiv$
consOntology(
$\quad$ *classes* : $\mathbb{P}(C)$,
$\quad$ *properties* : $\mathbb{P}(P)$,
$\quad$ *instances* : $\mathbb{P}(I)$,
$\quad$ *classProperties* : $\mathbb{P}(C \times P)$,
$\quad$ *classInstances* : $\mathbb{P}(C \times I)$,
$\quad$ *classAssociations* : $\mathbb{P}(C \times P \times C)$,
$\quad$ *instanceAssociations* : $\mathbb{P}(I \times P \times I)$
)

OPERATORS
...
END

- **Ontology(C, P, I)** : a generic data type for **classes**, **properties**, **instances**.
- Specifying class properties, class associations and classe intances
- Constrained instantiation: `instancePropertyValues` & `isWDInstancePropertyValues`.
- **37 operators**

Ontologies as Event-B theories

Ontologies Modelling Language (OML) - Operators

THEORY *OntologiesTheory*
. . .
OPERATORS
isWDInstancesAssociations **predicate** $(o : Ontology(C, P, I))$
well−definedness $isWDClassProperites(o) \land isWDClassInstances(o) \land isWDClassAssociations(o)$
direct definition
$instanceAssociations(o) \subseteq instances(o) \times properties(o) \times instances(o) \land$
$instanceAssociations(o) \subseteq \{i1 \mapsto p \mapsto i2 \mid i1 \in I \land p \in P \land i2 \in I \land$
$i1 \mapsto p \mapsto i2 \in instances(o) \times properties(o) \times instances(o) \land$
$(\exists c1, c2 \cdot c1 \in C \land c2 \in C \land \{c1, c2\} \subseteq getClasses(o)$
$\Rightarrow (c1 \mapsto p \mapsto c2 \in getClassAssociations(o) \land p \in getClassProperties(o)[\{c1\}] \land$
$i1 \in getClassInstances(o)[\{c1\}] \land i2 \in getClassInstances(o)[\{c2\}]))\}$
getInstanceAssociations **expression** $(o : Ontology(C, P, I))$
well−definedness $isWDInstancesAssociations(o)$
direct definition $instanceAssociations(o)$
isWDOntology **predicate** $(o : Ontology(C, P, I))$
direct definition
$isWDClassProperties(o) \land isWDClassInstances(o) \land$
$isWDClassAssociations(o) \land isWDInstancesAssociations(o)$
CheckOfSubsetOntologyInstances **predicate** $(o : Ontology(C, P, I), ipvs : \mathbb{P}(I \times P \times I))$
well−definedness $isWDOntology(o)$
direct definition
$ipvs \subseteq \{i1 \mapsto p \mapsto i2 \mid i1 \in I \land p \in P \land i2 \in I \land i1 \mapsto p \mapsto i2 \in instances(o) \times properties(o) \times$
$instances(o) \land \ldots\}$
isA **predicate** $(o : Ontology(C, P, I), c1 : C, c2 : C) \cdots$
. . .
THEOREMS
. . .
END

Ontologies as Event-B theories

Ontologies Modelling Language (OML) - Theorems

Useful theorems are proved.

THEORY *OntologiesTheory*
TYPE PARAMETERS
C ,
P ,
I
DATATYPES
$Ontology(C, P, I)$
...
OPERATORS
...
THEOREMS
thm1 : $\forall o, c1, c2, c3 \cdot o \in Ontology(C, P, I) \land isWDOntology(o) \land c1 \in C \land c2 \in C \land c3 \in C \land ontologyContainsClasses(o, \{c1, c2, c3\}) \Rightarrow (\mathbf{isA}(o, c1, c2) \land \mathbf{isA}(o, c2, c3) \Rightarrow \mathbf{isA}(o, c1, c3))$
END

The ARINC 661 standard as an ontology

Formal Definition of ARINC 661: Instantiation of the Theory of ontologies

- ARINC661Theory definition $\Longrightarrow$ Axiomatic definition of the operators
- Classes, properties and instances for ARNINC 661 are introduced
- **54 operators** and **17 axioms** were needed for chapter 3 of the ARINC 661 standard

```
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
 A661_EDIT_BOX_NUMERIC_ADMISSIBLE_VALUES expression () : ℙ(ARINC661Instances)
 CheckButtonState expression () : ARINC661Classes
 Label expression () : ARINC661Classes
 RadioBox expression () : ARINC661Classes
 CheckButton expression () : ARINC661Classes
 hasChildrenForRadioBox expression () : ARINC66Properties
 hasCheckButtonState expression () : ARINC66Properties
 SELECTED expression () : ARINC661Instances
 UNSELECTED expression () : ARINC661Instances
 isWDRadioBox predicate (o : Ontology(ARINC661Classes, ARINC66Properties, ARINC661Instances))
     well-definedness isWDOntology(o)
 isWDARINC661Ontology predicate (o : Ontology(ARINC661Classes, ARINC66Properties, ARINC661Instances))
...
```

The ARINC 661 standard as an ontology (Cont.)

Formal Definition of ARINC 661: Instantiation of the Theory of ontologies

AXIOMS

ARINC661ClassesDef:
$partition(ARINC661Classes, \{Label\}, \{RadioBox\}, \{CheckButton\}, \{CheckButtonState\}, \ldots)$

ARINC66PropertiesDef: $partition(ARINC66Properties, \{hasLabelStringForLabel\},$
$\{hasChildrenForRadioBox\}, \{hasCheckButtonState\}, \{hasLabelForCheckButton\}, \ldots)$

ARINC661InstancesDef: $partition(ARINC661Instances, \{A661\_TRUE\}, \{A661\_FALSE\}, \{SELECTED\},$
$\{UNSELECTED\}, LabelInstances, RadioBoxInstances, CheckButtonInstances, \ldots)$

consARINC661OntologyDef: $\forall ii, cii, ipvs \cdot ii \in \mathbb{P}(ARINC661Instances) \land$
$cii \in \mathbb{P}(ARINC661Classes \times ARINC661Instances) \land$
$ipvs \in \mathbb{P}(ARINC661Instances \times ARINC66Properties \times ARINC661Instances) \land$
$wellbuiltTypesElements \cap cii = \emptyset \land ii \subseteq WidgetsInstances \Rightarrow consARINC661Ontology(ii, cii, ipvs) = consOntology(\ldots)$

isWDRadioBoxDef: $\forall o \cdot o \in Ontology(ARINC661Classes, ARINC66Properties, ARINC661Instances)$
$\Rightarrow (isWDRadioBox(o) \Leftrightarrow (\forall \ldots)$

isWDARINC661OntologyDef: $\forall o \cdot o \in Ontology(ARINC661Classes, ARINC66Properties, ARINC661Instances)$
$\Rightarrow (isWDOntology(o) \land isWDRadioBox(o) \land isWDEditBoxNumeric(o) \Rightarrow isWDARINC661Ontology(o))$

CheckOfSubsetA661OntologyInstancesDef: $\forall o, ipvs \cdot$
$o \in Ontology(ARINC661Classes, ARINC66Properties, ARINC661Instances) \land$
$ipvs \in \mathbb{P}(ARINC661Instances \times ARINC66Properties \times ARINC661Instances)$
$\Rightarrow (isWDARINC661Ontology(consOntology(\ldots)) \Rightarrow CkeckOfSubsetA661OntologyInstances(\ldots))$

...

THEOREMS

....

Case study - Weather Radar System

WXR Case study

The pilot interacts with this application (Mode selection, angle selection, etc.).

- Widgets ⇒ formalised as instances
- Action on the widgets ⇒ theory operators
- Properties of the application ⇒ Proved as theorems
- **Requirements for the WXR**
  - The selection of the check button must be exclusive
  - The tilt angle must be within a specific range
  - ...

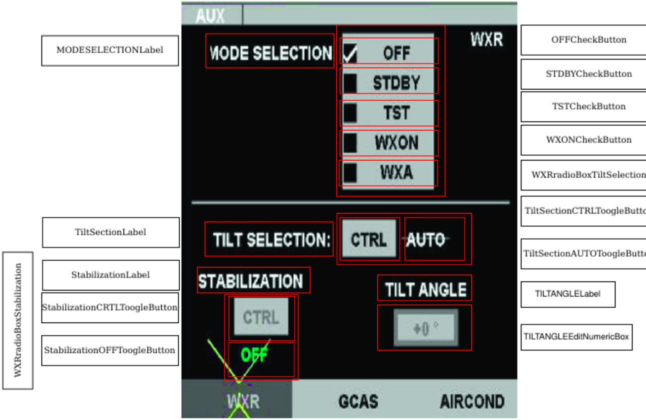


Figure: Dassault Falcon 6X cockpit

An extract of the Event-B model

State variable

The *ui* user interface is a typed by ontology concepts

MACHINE *WXRModel*
VARIABLES *ui*
INVARIANTS

$$\begin{array}{ll}
\textbf{inv1:} & ui \in \mathbb{P}(ARINC661Instances \times ARINC66Properties \times ARINC661Instances) \\
\textbf{inv2:} & \exists uiArg \cdot ((ui = initiator(A661WXROntology)) \lor \\
& \quad \exists m \cdot isWDChangeModeSelection(A661WXROntology, uiArg, m) \land \\
& \qquad ui = changeModeSelection(A661WXROntology, uiArg, m)) \lor \\
& \quad (\exists v \cdot isWDChangeTitlAngle(A661WXROntology, uiArg, v) \land \\
& \qquad ui = changeTitlAngle(A661WXROntology, uiArg, v)) \lor \ \ldots \\
\textbf{thm1:} & isVariableOfARINC661Ontology(A661WXROntology, ui) \\
\textbf{thm2:} & (\forall rb, b1, b2 \cdot rb \in RadioBoxInstances \land b1 \in CheckButtonInstances \land b2 \in CheckButtonInstances \land \\
& rb \mapsto hasChildrenForRadioBox \mapsto b1 \in ui \land rb \mapsto hasChildrenForRadioBox \mapsto b2 \in ui \\
& \quad \Rightarrow (b1 \mapsto hasCheckButtonState \mapsto SELECTED \in ui \land b2 \mapsto hasCheckButtonState \mapsto SELECTED \in ui \\
& \qquad \Rightarrow b1 = b2)) \land \\
& (\forall rb, b1, b2 \cdot rb \in RadioBoxInstances \land b1 \in ToggleButtonInstances \land b2 \in ToggleButtonInstances \land \\
& rb \mapsto hasChildrenForRadioBox \mapsto b1 \in ui \land rb \mapsto hasChildrenForRadioBox \mapsto b2 \in ui \\
& \quad \Rightarrow (b1 \mapsto hasToggleButtonState \mapsto SELECTED \in ui \land b2 \mapsto hasToggleButtonState \mapsto SELECTED \in ui \\
& \qquad \Rightarrow b1 = b2)) \land \\
& (\forall ed, v \cdot ed \mapsto hasValue \mapsto v \in ui \Rightarrow v \in A661_EDIT_BOX_NUMERIC_ADMISSIBLE_VALUES)
\end{array}$$

- inv1 and inv2 checks that state variable *ui* is manipulated by two operators of the Theory
- thm1 and thm2 guarantee the exclusive property for button selection

An extract of the Event-B model (Cont.)

Changing selection mode and anngle

Modes and Angles are modified using operators of the theory

EVENTS
  INITIALISATION
  THEN
    act1 : $ui := initiator(A661WXROntology)$
  END
  changeModeSelection
  ANY *mode*
  WHERE
    grd1 : $mode \in WXRcheckButtons$
    grd2 : $isWDChangeModeSelection(A661WXROntology, ui, mode)$
  THEN
    act1 : $ui := changeModeSelection(A661WXROntology, ui, mode)$
  END
  changeTitlAngle
  ANY *newAngle*
  WHERE
    grd1 : $newAngle \in A661\_EDIT\_BOX\_NUMERIC\_VALUES$
    grd2 : $newAngle \in A661\_EDIT\_BOX\_NUMERIC\_ADMISSIBLE\_VALUES$
    grd3 : $isWDChangeTitlAngle(A661WXROntology, ui, newAngle)$
  THEN
    act1 : $ui := changeTitlAngle(A661WXROntology, ui, newAngle)$
  END
...
END

Annotating Events

How to handle domain knowledge related to events ?

Still in the context of Interactive Critical Systems, let us consider a requirement of the form

Any **Input** event **must be followed** by a **confirmation/abortion** event

- **Input** and **confirmation/abortion** may correspond to various events in an interactive system (using keyboard, voice, finger designation, box, timeout, etc.
- **must be followed** relates to a temporal logic property

According to our view, this property is domain knowledge oriented (standard, regulation, etc.)

Can we manage this this kind of properties ?

- Formal methods are not equipped with the capability to express such properties
- Classical solution $\Longrightarrow$ Use of ad'hoc modelling by hard encoding the property in the model

Annotating Events

Our approach

- Define an ontology of Events (in an Event-B theory)
- Use the Meta-Theory of Event-B namely EB4EB allowing to manipulate states and Events (Another Event-B theory)
- Events are annotated using a relation of the form $EVENTS \longleftrightarrow EVT\_TAGS$

  Use the predicate operator *isNecFollowedBy* modelling this property

Can we manage this this kind of properties ?

- Formal methods are not equipped with the capability to express such properties
- Classical solution $\Longrightarrow$ Use of ad'hoc modelling by hard encoding the property in the model

Annotating Events

Composition of different algebraic theories

- A case where the modelling language does not offer built-in operators to express specific properties.

```
THEORY DomainDynamicPropertiesTheory
IMPORT THEORY OntologiesTheory , EventBTheory ,
TYPE PARAMETERS Tg, Prop , St, Ev
OPERATORS
    ...
    isNecFollowedByWD predicate (...)
    isNecFollowedBy predicate
                    (m : Mach(St, Ev),
                     eo : Ontology(Tg, Prop, Ev) ,
                     StartTag : Tg ,
                     IntermediateTags : Pow(Tg)
                     EndTag : Tg
                     v : Pow(Ev × Pow(St × NAT)) )

      well−definedness isNecFollowedByWD (...)

      direct definition
          ∀EvtInst.EvtInst ∈ classInstances(eo, startTag) ⟹
                     IsReachable(
                                m ,
                                EvtInst ,
                                classInstances(eo, endTag) ,
                                classInstances(eo, intermediateTags) ,
                                v(EvtInst) )
END
```

Proofs

- An ontology of tagged events
- A meta-theory to manipulate Event-B models i.e state-transitions systems
- Definition of a specific temporal logic operator composing events and tags

Conclusion

Event-B + Theories

- A framework integrating both
  - Event-B machines for system models i.e. **prescriptive models**
  - Algebraic data type theories for domain knowledge i.e. **sharable descriptive models**
  - Data types and operators annotate states/transitions (events) i.e **model annotation**
- Well-Definedness (WD) conditions are useful to guarantee correct by construction use of operators
- Outsourcing: complex proofs at the theory level, once and for all
- Reuse of theorems in formal Event-B models and reduction of proof efforts for engineers

Our experiments

Many theories for domain knowledge have been developed following the presented approach

- Differential equations for Hybrid systems, braking systems for trains
- Interactive systems: Arinc 661, widgets
- Tanks, Logistics, Units
- Autonomous vehicles (collaborative work with NII)

Other are under development with RATP

- Trains and railway systems
- Environment models

Conclusion

Conformance/Compliance: towards a Conformance/Compliance by construction

- Standards could be formalised as algebraic theories
  - Independent of any system
  - Stateless sharable theories
- Two approaches are identified
  - **A priori** $\Longrightarrow$ system models are designed based on formalised standards
  - **A posteriori** $\Longrightarrow$ system models are aligned with standards (annotation uses gluing mappings)
- The design of standards as theories is not free and requires trained humans resources !!!!

To Do

- More formalised domain theories
- Consistence of defined theories. Are they inhabited ?
- Build bridges for Event-B Theories with other formal modelling approaches and proof assistants
- An engineering process: what is the level of granularity for axiomatic theories ? How to manage the complexity of the developments ? ....
- ...

Thank You

yamine@n7.fr