Formal and Incremental Verification of SysML Specifications for the Design of Component-Based Systems

Oscar Carrillo

Département d'Informatique des Systèmes Complexes (DISC), Femto-ST UMR 6174 CNRS

Encadrement : Hassan Mountassir et Samir Chouali
Soutenue le 17 decembre 2015 à Besançon

Outline

1. Introduction
2. Scientific Context
3. Contributions
4. Conclusion and Perspectives

Outline

1. Introduction
2. Scientific Context
3. Contributions
4. Conclusion and Perspectives

Context

Development of Systems by Component Assembly

- Reduce complexity
- Reduce development costs
- Improve reliability

Functional Requirements

Functional properties that the system must satisfy to fulfill user needs

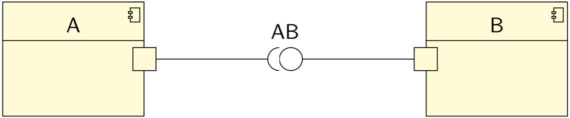
SysML

Complex systems, communicate, popular

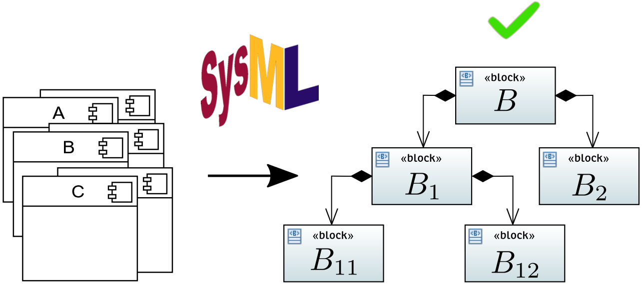
Context

Component-Based Systems (CBS)

- Components described by their interfaces
- Simple and composite components
- Built by assembling the components
- Architecture described by the connections between the components
- Leads to big systems (complex)



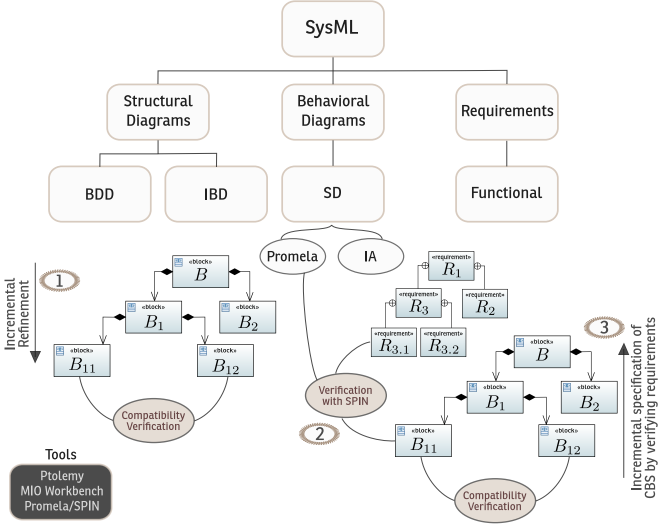
Challenge



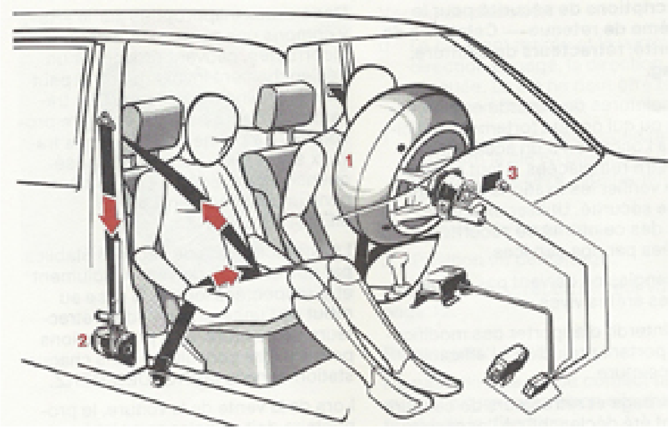
In SysML a component is defined by a block

How to formally ensure reliability of CBS described by SysML ?

Contributions



A Car Safety System



Airbag and seat-belts protecting passenger lives

Outline

1 Introduction

2 Scientific Context
- The SysML Language
- Interface automata

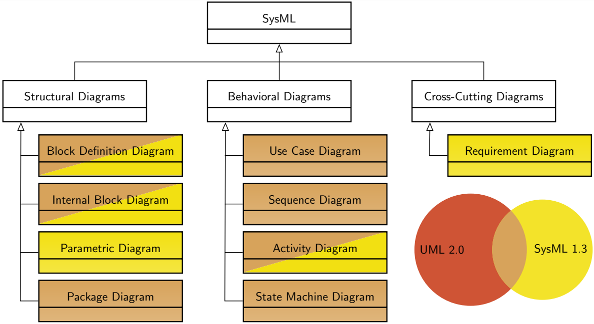
3 Contributions

4 Conclusion and Perspectives

The SysML Language

Systems Modeling Language

- Model hardware and software systems
- Functional and non-functional requirements
- Interdisciplinary
- SysML is a communication method, not a methodology

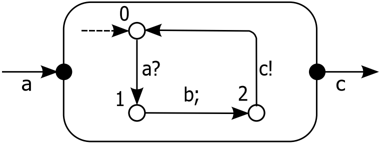


Interface Automata [Alfaro, Henzinger 2001]

Definition

*An **interface automaton** A is represented by the tuple $\langle S, I, \Sigma^I, \Sigma^O, \Sigma^H, \delta \rangle$ such as :*

- *S is a finite set of states,*
- *$I \subseteq S$ is a finite set of initial states,*
- *Σ^I, Σ^O and Σ^H, respectively denote the sets of input, output and internal actions.*
  $\Sigma_A = \Sigma^I \cup \Sigma^O \cup \Sigma^H$,
- *$\delta \subseteq S \times \Sigma \times S$ is the set of transitions between two states.*



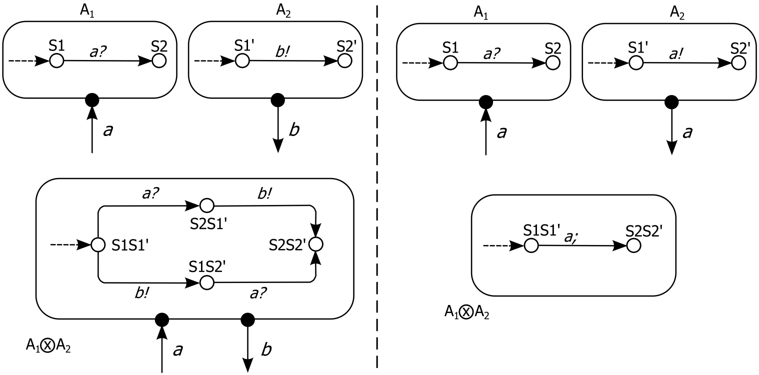
Interface automata synchronized product

Definition

Let A_1 and A_2 two composable interface automata. The ***synchronized product*** $A_1 \otimes A_2$ *of* A_1 *and* A_2 *is defined by :*

- $S_{A_1 \otimes A_2} = S_{A_1} \times S_{A_2}$ and $I_{A_1 \otimes A_2} = I_{A_1} \times I_{A_2}$;
- $\Sigma^I_{A_1 \otimes A_2} = (\Sigma^I_{A_1} \cup \Sigma^I_{A_2}) \setminus Shared(A_1, A_2)$;
- $\Sigma^O_{A_1 \otimes A_2} = (\Sigma^O_{A_1} \cup \Sigma^O_{A_2}) \setminus Shared(A_1, A_2)$;
- $\Sigma^H_{A_1 \otimes A_2} = \Sigma^H_{A_1} \cup \Sigma^H_{A_2} \cup Shared(A_1, A_2)$;
- $((s_1, s_2), a, (s'_1, s'_2)) \in \delta_{A_1 \otimes A_2}$ if
  - $a \notin Shared(A_1, A_2) \wedge (s_1, a, s'_1) \in \delta_{A_1} \wedge s_2 = s'_2$
  - $a \notin Shared(A_1, A_2) \wedge (s_2, a, s'_2) \in \delta_{A_2} \wedge s_1 = s'_1$
  - $a \in Shared(A_1, A_2) \wedge (s_1, a, s'_1) \in \delta_{A_1} \wedge (s_2, a, s'_2) \in \delta_{A_2}$.

Interface automata synchronized product



Illegal states

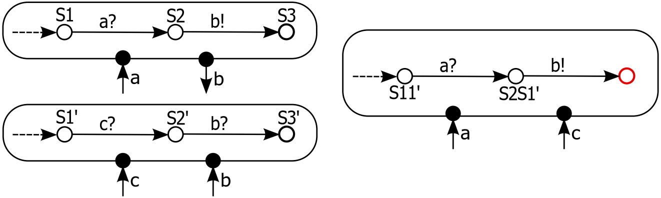
Definition

Let two composable interface automata A_1 *and* A_2*, the set of* ***illegal states*** $Illegal(A_1, A_2) \subseteq S_{A_1} \times S_{A_2}$ is defined by

$$\{(s_1, s_2) \in S_{A_1} \times S_{A_2} \mid \exists a \in Shared(A_1, A_2) \,.\, C\}$$

where C *is :*

$$C = (a \in \Sigma^O_{A_1}(s_1) \wedge a \notin \Sigma^I_{A_2}(s_2)) \vee (a \in \Sigma^O_{A_2}(s_2) \wedge a \notin \Sigma^I_{A_1}(s_1))$$

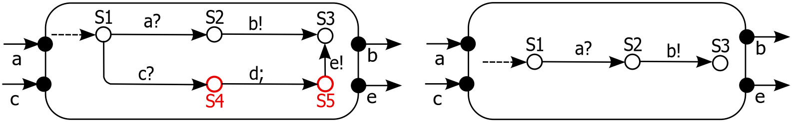


Composition

Definition

*The **composition** $A_1 \parallel A_2$ of two IA A_1 and A_2 is defined by :*
(i) $S_{A_1 \parallel A_2} = Comp(A_1, A_2)$,
(ii) $I_{A_1 \parallel A_2} = I_{A_1 \otimes A_2} \cap Comp(A_1, A_2)$
(iii) $\delta_{A_1 \parallel A_2} = \delta_{A_1 \otimes A_2} \cap Comp(A_1, A_2) \times \Sigma_{A_1 \parallel A_2} \times Comp(A_1, A_2)$

Where $Comp(A_1, A_2) = A_1 \otimes A_2 - Illegal(A_1, A_2)$



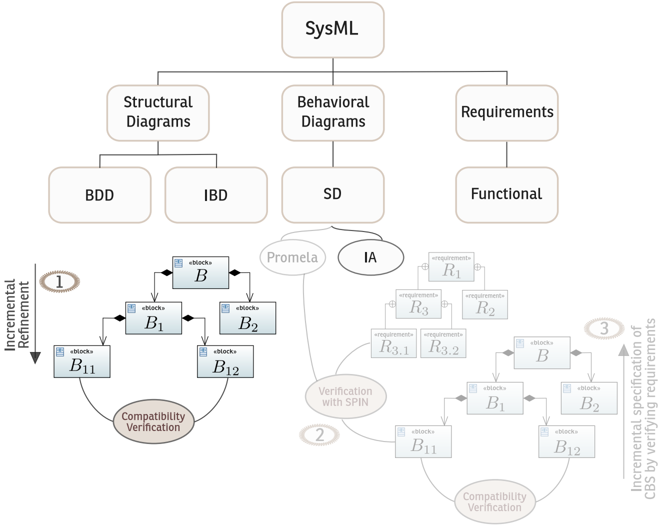
Compatibility

Two interface automata A_1 and A_2 are compatibles if and only if their composition $A_1 \parallel A_2$ has at least one reachable state.

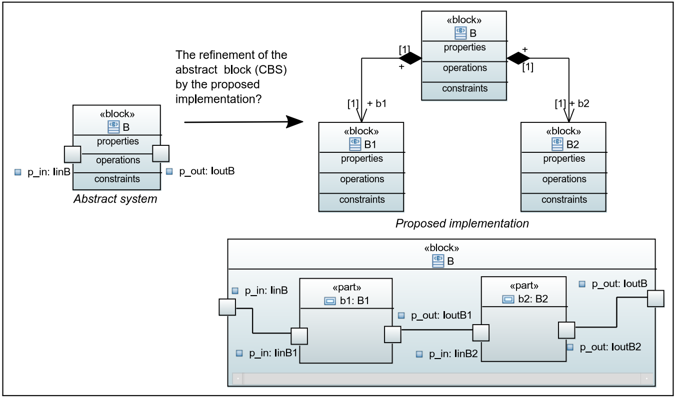
Outline

1. Introduction
2. Scientific Context
3. Contributions
   - Incremental Refinement of a CBS Architecture
   - Formal Verification of SysML Requirements
   - Incremental Specification of CBS Architecture
4. Conclusion and Perspectives

Incremental Refinement of a CBS Architecture



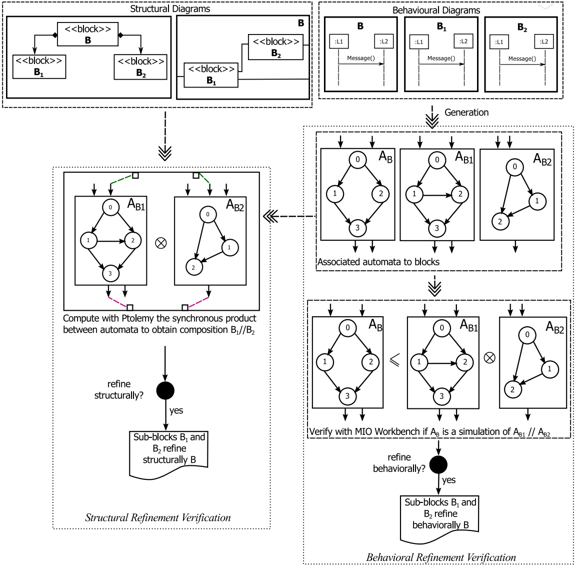
Overview



Refinement by decomposition

Structural and behavioral refinement relation.

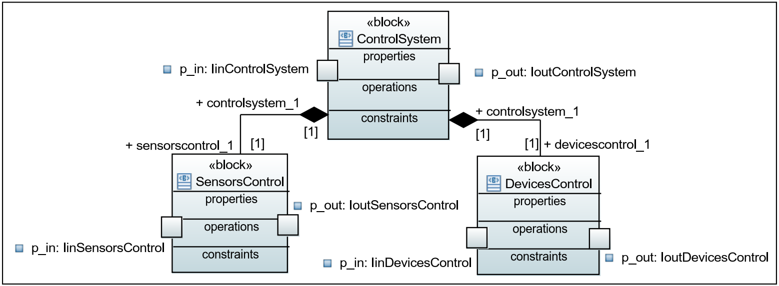
Refinement Process



CBS Specification with SysML 1.3

Block Definition Diagram (BDD)

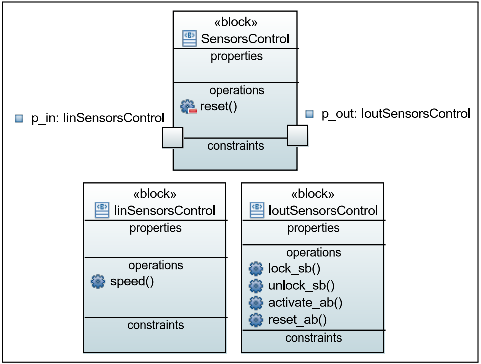
Structure of abstract system



CBS Specification with SysML 1.3

Block Definition Diagram (BDD)

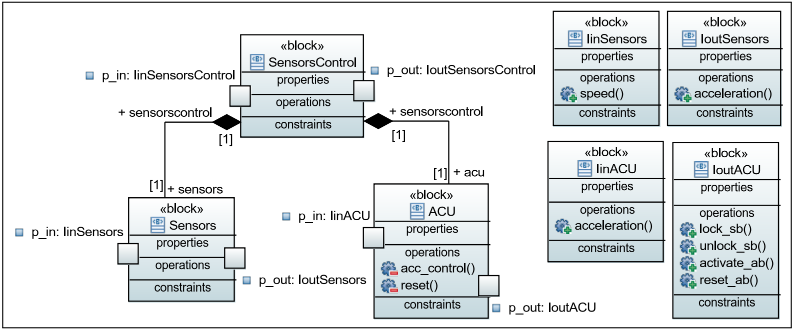
Description of *SensorsControl* block



CBS Specification with SysML 1.3

Block Definition Diagram (BDD)

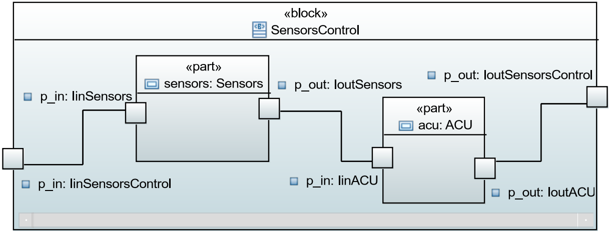
Proposed decomposition for abstract block.



CBS Specification with SysML 1.3

Internal Block Diagram (IBD)

Proposed internal structure for abstract block

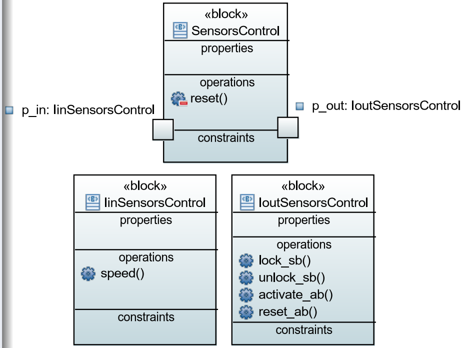


Formal SysML Specification

Definition : SysML Block

Let SB a set of blocks modeled with a BDD, a SysML block B in SB is a tuple $\langle \Phi_B, P_{in}, P_{out}, TypePort \rangle$, where :

- Φ_B is the set of the private operations in B,
- P_{in} the unique input port of B,
- P_{out} the unique output port of B.
- $TypePort : P_{in} \cup P_{out} \rightarrow SB$ determines the interface that types each port.

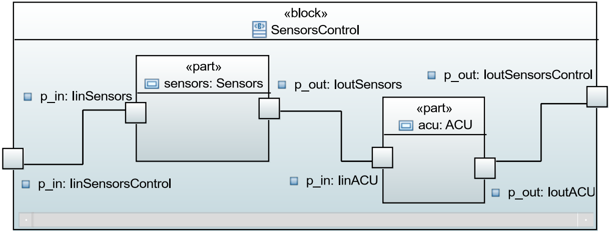


Formal SysML Specification

Definition : SysML IBD

A SysML IBD, of a composite block, is a tuple $\langle \Phi Parts, iP_{in}, iP_{out}, eP_{in}, eP_{out}, Connector \rangle$, where :

- $\Phi Parts$ is the set of parts,
- iP_{in} and iP_{out} are the sets of internal input and output ports,
- eP_{in} and eP_{out} are the external input and output ports,
- $Connector : P_{in} \cup P_{out} \rightarrow P_{in} \cup P_{out}$ associates input and output ports to other input and output ports.



Structural Refinement

Definition : Structural refinement of SysML blocks

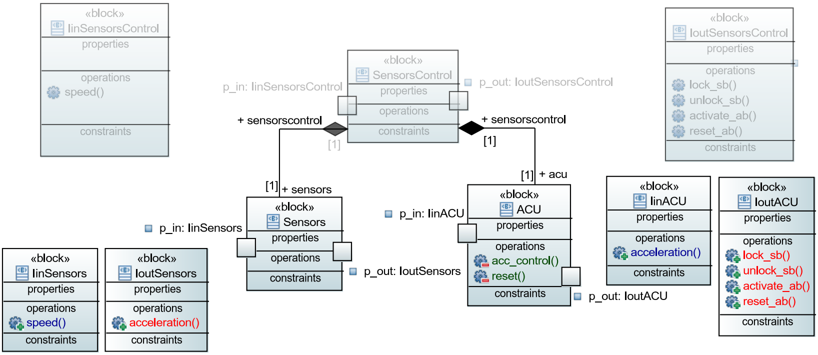
Let B be an abstract block described with the BDD, and IBD_B the internal block diagram of B. Let $B_1, ..., B_n$ be the set of blocks composing B according to the BDD, so $B_1, ..., B_n$ refine structurally B iff :

- $B_1, ..., B_n$ are consistent with B,
- the interacting blocks $B_1, ..., B_n$ according to IBD_B are compatible.

Consistency Verification

Condition 1 (Composability)

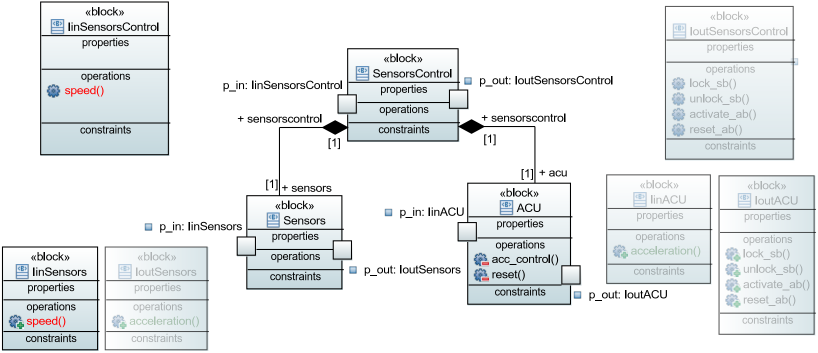
For every pair of connected sub-blocks $\{B_i, B_j\}$, it holds that : $\Phi_{inBi} \cap \Phi_{inBj} = \Phi_{outBi} \cap \Phi_{outBj} = \Phi_{Bi} \cap (\Phi_{Bj} \cup \Phi_{inBj} \cup \Phi_{outBj}) = \Phi_{Bj} \cap (\Phi_{Bi} \cup \Phi_{inBi} \cup \Phi_{outBi}) = \emptyset$



Consistency Verification

Condition 2 (At least same inputs)

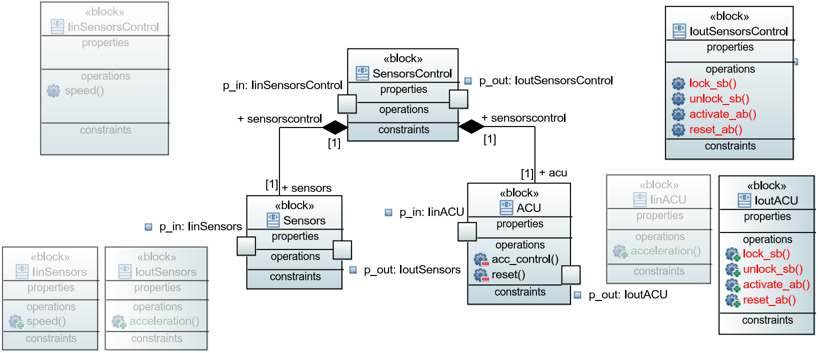
For a sub-block B_i connected to the external input port eP_{in} it holds that : $\Phi_{inB} \subseteq \Phi_{inBi}$



Consistency Verification

Condition 3 (At most same outputs)

For a sub-block B_i connected to the external port eP_{out} it holds that : $\Phi_{outBi} \subseteq \Phi_{outB}$.



Compatibility Verification

Interface Automata Generation

Obtained by applying [Chouali *et al.* 2011] approach, from sequence diagrams.

Condition 4 (Compatibility)

Two connected sub-blocks B_1 and B_2 are compatible if their interface automata A_1 and A_2 are compatible.

Ptolemy II [Barais *et al.* 2005]

Verification module for interface automata composition

Behavioral Refinement

- Let $P = A_1 \parallel \ldots \parallel A_n$, be the composite automaton of the composition of a set of blocks $B_1, \ldots, B_n$
- Let Q be the interface automaton for an abstract block B

Definition : Interface Automata Refinement [Alfaro *et al.* 2005]

An interface automaton P refines an interface automaton Q, written $P \leq_a Q$, if

1. $\Sigma_Q^I \subseteq \Sigma_P^I$ and $\Sigma_Q^O \supseteq \Sigma_P^O$
2. there is an alternating simulation $\leq_a$ by Q of P such that $I_P \leq_a I_Q$

Behavioral Refinement

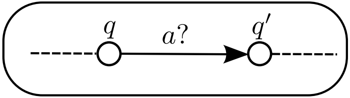
Definition : Alternating Simulation [Alfaro *et al.* 2005]

For a pair of interface automata $P = \langle S_P, I_P, \Sigma_P^I, \Sigma_P^O, \Sigma_P^H, \delta_P \rangle$ and $Q = \langle S_Q, I_Q, \Sigma_Q^I, \Sigma_Q^O, \Sigma_Q^H, \delta_Q \rangle$ with the same signature, a binary relation $\leq_a \subseteq S_P \times S_Q$ is an alternating simulation if whenever $p \leq_a q$ and $a \in \Sigma_P$ it holds that :

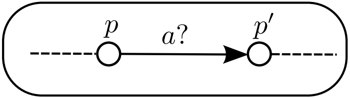
if $q \xrightarrow{a?} q'$ and $a \in \Sigma_Q^I$ then $\exists p'.p \xrightarrow{a?} p'$ and $(p', q') \in \leq_a$

if $p \xrightarrow{a!} p'$ and $a \in \Sigma_P^O$ then $\exists q'.q \xrightarrow{\tau}{}^* q'.\exists q''.q' \xrightarrow{a!}{}^* q''$ and $(p', q'') \in \leq_a$

if $p \xrightarrow{a;} p'$ and $a \in \Sigma_P^H$ then $\exists q'.q \xrightarrow{\tau}{}^* q'$ and $(p', q') \in \leq_a$



Q



P

Behavioral Refinement

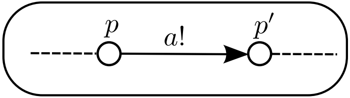
Definition : Alternating Simulation [Alfaro *et al.* 2005]

For a pair of interface automata $P = \langle S_P, I_P, \Sigma_P^I, \Sigma_P^O, \Sigma_P^H, \delta_P \rangle$ and $Q = \langle S_Q, I_Q, \Sigma_Q^I, \Sigma_Q^O, \Sigma_Q^H, \delta_Q \rangle$ with the same signature, a binary relation $\leq_a \subseteq S_P \times S_Q$ is an alternating simulation if whenever $p \leq_a q$ and $a \in \Sigma_P$ it holds that :

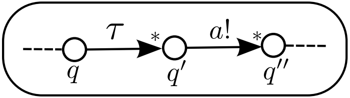
if $q \xrightarrow{a?} q'$ and $a \in \Sigma_Q^I$ then $\exists p'.p \xrightarrow{a?} p'$ and $(p', q') \in \leq_a$

if $p \xrightarrow{a!} p'$ and $a \in \Sigma_P^O$ then $\exists q'.q \xrightarrow{\tau}{}^* q'.\exists q''.q' \xrightarrow{a!}{}^* q''$ and $(p', q'') \in \leq_a$

if $p \xrightarrow{a;} p'$ and $a \in \Sigma_P^H$ then $\exists q'.q \xrightarrow{\tau}{}^* q'$ and $(p', q') \in \leq_a$



P



Q

Behavioral Refinement

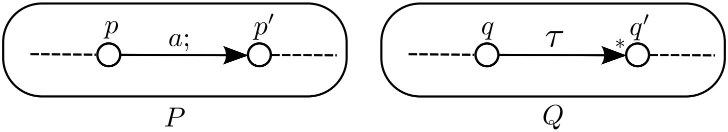
Definition : Alternating Simulation [Alfaro *et al.* 2005]

For a pair of interface automata
$P = \langle S_P, I_P, \Sigma_P^I, \Sigma_P^O, \Sigma_P^H, \delta_P \rangle$ and $Q = \langle S_Q, I_Q, \Sigma_Q^I, \Sigma_Q^O, \Sigma_Q^H, \delta_Q \rangle$ with the same signature, a binary relation $\leq_a \subseteq S_P \times S_Q$ is an alternating simulation if whenever $p \leq_a q$ and $a \in \Sigma_P$ it holds that :

if $q \xrightarrow{a?} q'$ and $a \in \Sigma_Q^I$ then $\exists p'.p \xrightarrow{a?} p'$ and $(p', q') \in \leq_a$

if $p \xrightarrow{a!} p'$ and $a \in \Sigma_P^O$ then $\exists q'.q \xrightarrow{\tau}{}^* q'.\exists q''.q' \xrightarrow{a!}{}^* q''$ and $(p', q'') \in \leq_a$

if $p \xrightarrow{a;} p'$ and $a \in \Sigma_P^H$ then $\exists q'.q \xrightarrow{\tau}{}^* q'$ and $(p', q') \in \leq_a$



MIO Workbench [Bauer *et al.* 2010]

Modal automaton

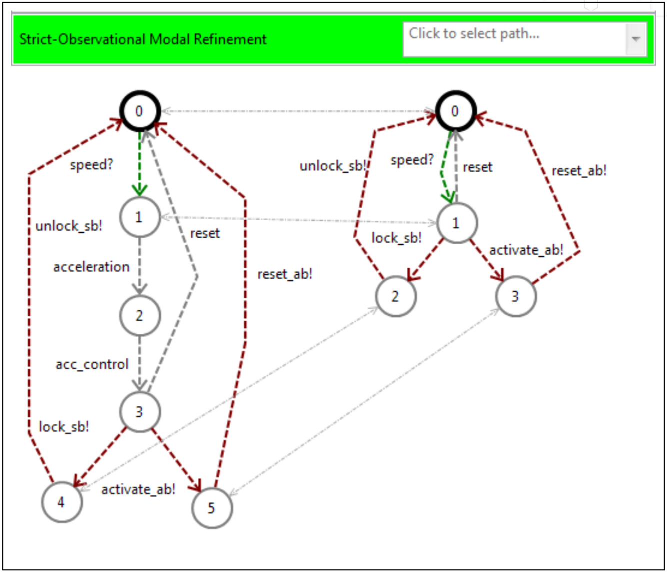
- Larsen *et al.* 2007
- A modal automaton S is a six tuple :
  $S = (S_S, I_S, \Sigma_S^{ext}, \Sigma_S^H, \longrightarrow_\Diamond^S, \longrightarrow_\Box^S)$
- $\mathcal{T}(S_P, I_P, \Sigma_P^I, \Sigma_P^O, \Sigma_P^H, \delta_P) = (S_S, I_S, \Sigma_S^{ext}, \Sigma_S^H, \longrightarrow_\Diamond, \longrightarrow_\Box)$

Alternating simulation and observational modal refinement

Alternating simulation and observational modal refinement coincide for interface automata in the following sense :
For any two interface automata P, Q :

$$P \leq_a Q \text{ iff } \mathcal{T}(P) \leq_m^* \mathcal{T}(Q)$$

MIO Workbench



Outline

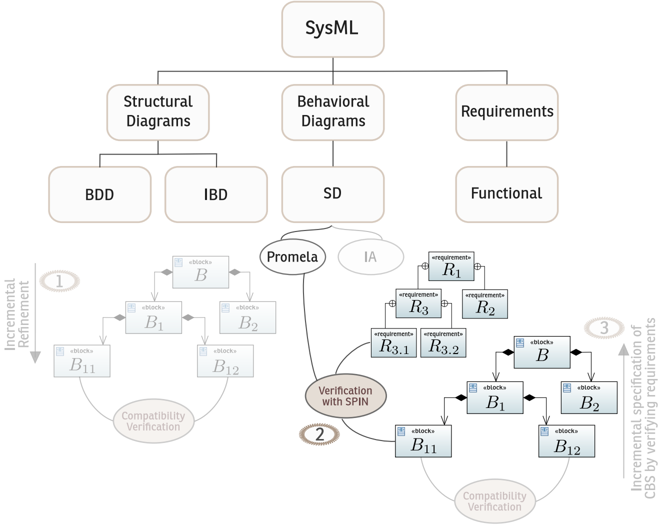
1 Introduction

2 Scientific Context

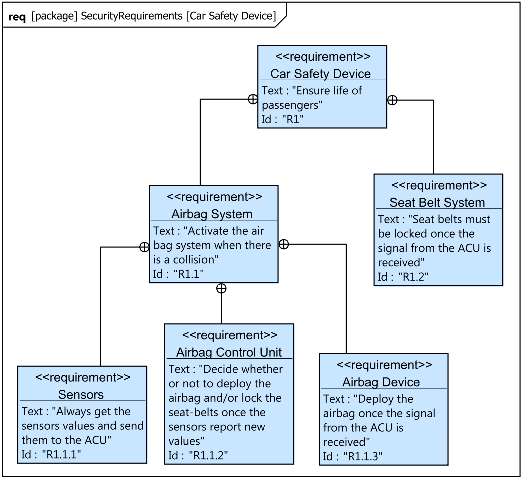
3 Contributions
- Incremental Refinement of a CBS Architecture
- Formal Verification of SysML Requirements
- Incremental Specification of CBS Architecture

4 Conclusion and Perspectives

Formal Verification of SysML Requirements



Requirements for a Car Safety System

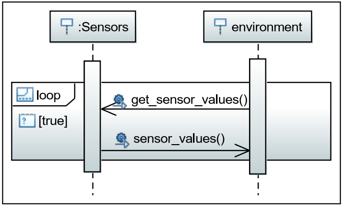


Requirements Refinement for a Safety System

Case Study

Sensors Requirements

Always get the sensor values and send them to the ACU.



From SD to Promela

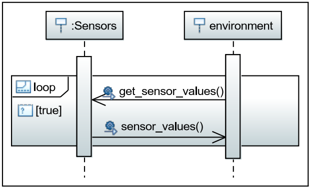
SD element	Promela Element	Promela Statement
Lifeline	Process	`proctype{...}`
Message	Message	`mtype{m1,...,mn}`
Connector	Communication channel for each message arrow	`chan chanName = [1] of {mtype}`
Send and receive events	Send and receive operations	Send ⇒ `ab!m`, Receive ⇒ `ab?m`

Alt combined fragment	if condition	`if`
		`  ::(guard)->ab_p?p;`
		`  :: else -> ab_q?q;`
		`fi;`

Loop combined fragment	do operator	`do`
		`  ::ab_p?p;`
		`od`

Mapping of basic concepts from Sequence Diagrams to Promela
Lima *et al.* 2009

Sensors block Promela description



SD for sensors block

```
...
proctype proc_sensors(){
do
sensors_environment_get_sensor_values?get_sensor_values;
sensors_environment_sensor_values!sensor_values;
od
}
proctype proc_environment(){
do
sensors_environment_get_sensor_values!get_sensor_values;
sensors_environment_sensor_values?sensor_values;
od
}
init{
atomic{run proc_sensors();
run proc_environment();}
}
```

Promela code for sensors block

Verification with SPIN

- Promela description must keep track of *who is sending/receiving what message* at any time of the execution.

Flags for sensor component

- `send, receive`
- `msg_get_sensor_values, msg_send_sensor_values`
- `sensors, environment`

- All flags updated by `d_step`

LTL Property with flags

$\square$((sensors && receive && msg_get_sensor_values) $\rightarrow$
$\lozenge$ (sensors && send && msg_sensor_values))

Outline

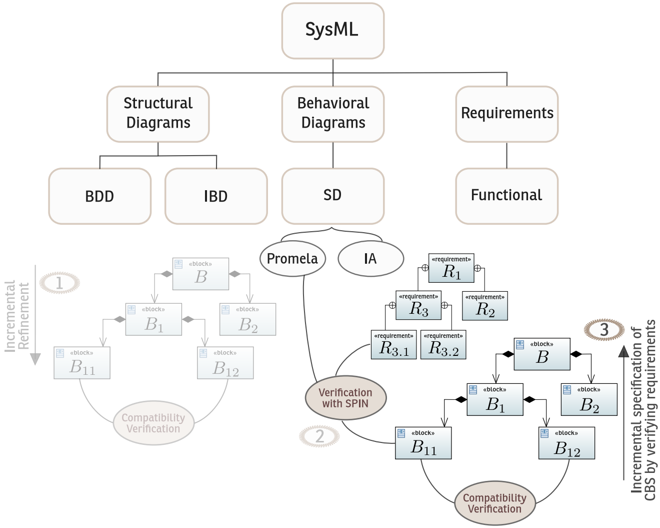
1 Introduction

2 Scientific Context

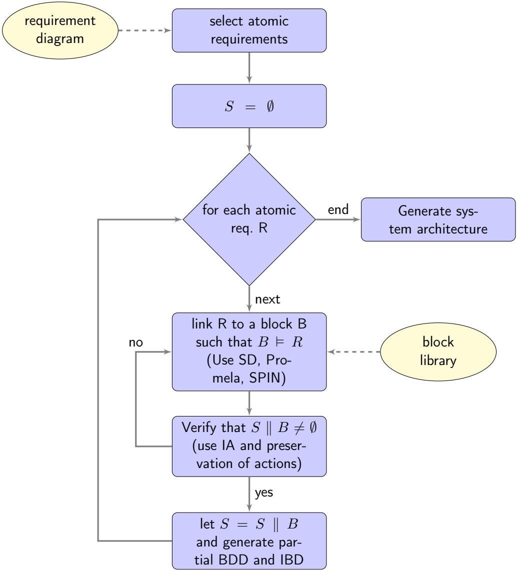
3 Contributions
- Incremental Refinement of a CBS Architecture
- Formal Verification of SysML Requirements
- Incremental Specification of CBS Architecture

4 Conclusion and Perspectives

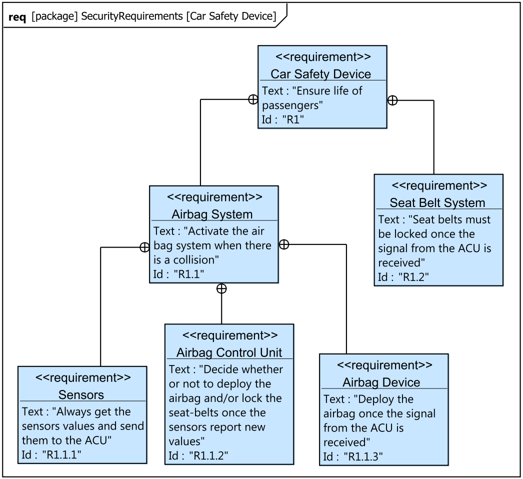
Incremental Specification of CBS Architecture



Approach Steps



Requirements for a Car Safety System



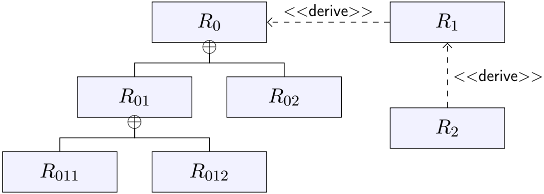
Requirements Refinement for a Safety System

Requirement Diagram Analysis

Definition : Requirement diagram specification

We specify a SysML requirement diagram by $RD = \langle IR, SR, RelC, RelD \rangle$ such that :

- IR : define the set of initial requirements,
- SR : the set of all requirements.
- $RelC \subseteq SR \times P(SR)$ the relation of containment, where $P(SR)$ is the set of the subsets of SR.
- $RelD \subseteq SR \times P(SR)$ the relation of derivation.



Atomic Requirements

Definition : Atomic requirements

The set of atomic requirements in the requirement diagram specified by $RD = \langle IR, SR, RelC, RelD \rangle$ is the set $AR = \{R | R \in SR, \nexists (R, \{R_i, ...R_n\}) \in RelC\}$

Theorem : System satisfying all atomic requirements

Let S be a CBS, let $RD = \langle IR, SR, RelC, RelD \rangle$ be the specification of a requirement diagram, and let AR be the set of atomic requirements of RD. S satisfies all the requirements in SR iff it satisfies the atomic requirements AR.

Atomic Requirements in Case Study

R1.1.1 : Sensors

Always get the sensor values and send them to the ACU.

$\Box$((sensors && receive && msg_get_sensor_values) $\rightarrow$
$\Diamond$ (sensors && send && msg_sensor_values))

R1.1.2 : Airbag Control Unit

Decide whether or not to deploy the airbag and/or lock the seat-belts once the sensors report new values.

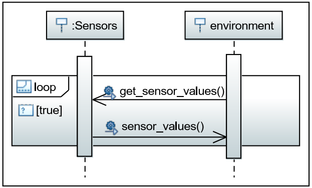
$\Box$((acu && receive && msg_sensor_values) $\rightarrow$
$\Diamond$ (acu && send && (msg_act_sb || msg_act_ab)))

Connected Requirements

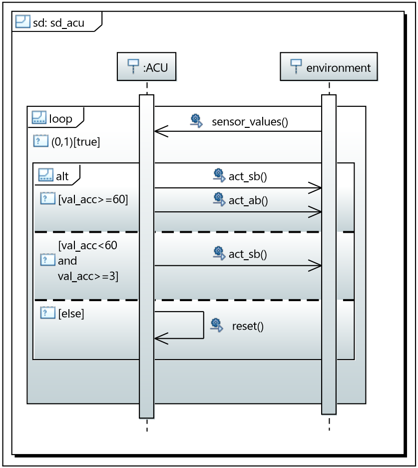
R1.1.1 and R1.1.2 share input and output actions.

Block Library

Component interfaces are described by SysML Sequence Diagrams

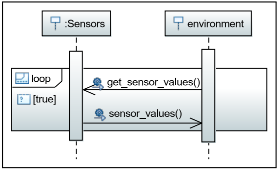


SD for sensors block



SD for the ACU block

Block Sensors

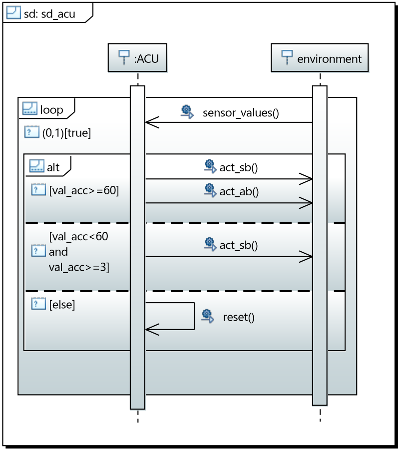


SD for sensors block

```
...
proctype proc_sensors(){
do
  sensors_environment_get_sensor_values?get_sensor_values;
  sensors_environment_sensor_values!sensor_values;
od
}
proctype proc_environment(){
do
  sensors_environment_get_sensor_values!get_sensor_values;
  sensors_environment_sensor_values?sensor_values;
od
}
init{
  atomic{run proc_sensors();
  run proc_environment();}
}
```

Promela code for sensors block

Block ACU

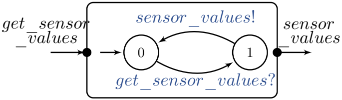


SD for the ACU block

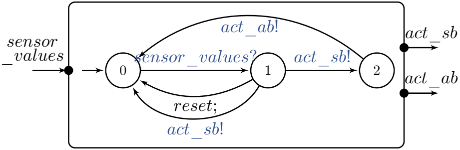
```
...
proctype proc_acu(){
do
  ::acu_environment_sensor_values?
                    sensor_values;
  if
    ::(val_acc>=60)->
        {acu_environment_act_sb!act_sb;
         acu_environment_act_ab!act_ab;}
    ::((val_acc<60) && (val_acc>=3))->
        acu_environment_act_sb!act_sb;
    ::else{acu_reset!reset;
           acu_reset?reset;}
  fi;
od}
```

Promela code for ACU block

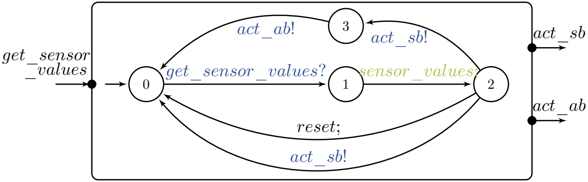
Compatibility Verification



IA for the Sensors block



IA for the ACU

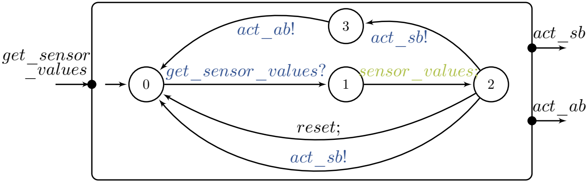


IA composition generated by `Ptolemy` (Lee *et al.* 2004)

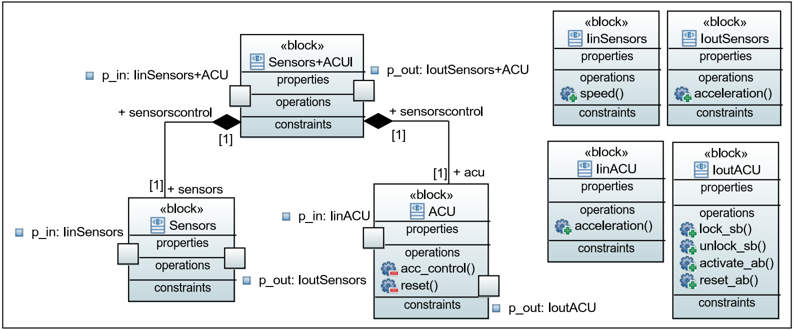
Requirement Preservation over Composition

Theorem : Preservation of requirements

The composite block $S = B_i \parallel B_{i+1}$ preserves the requirements $\{R_i, R_{i+1}\}$ iff the interface automata A_i, and A_{i+1}, are compatible, and the input and output actions, I_i, I_{i+1}, O_i, and O_{i+1} are preserved in S.

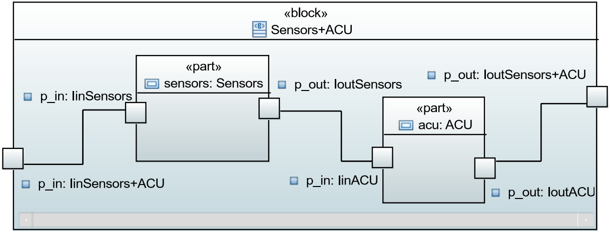


Architecture Specification



BDD for the second iteration

Architecture Specification



IBD for the second iteration

Outline

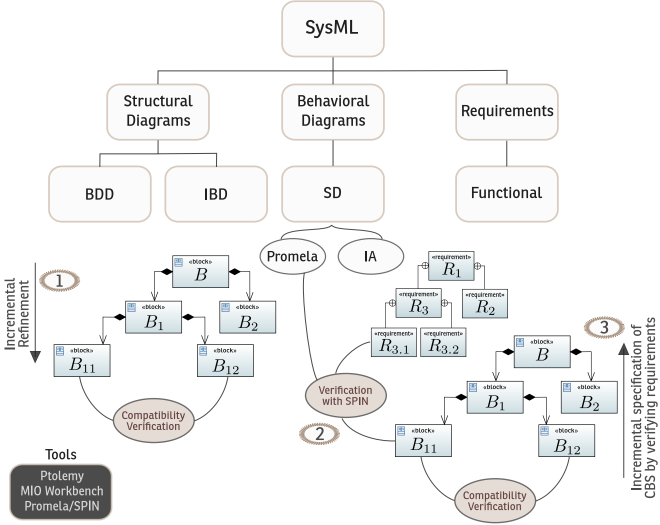
1 Introduction

2 Scientific Context

3 Contributions

4 Conclusion and Perspectives

Contributions



Contributions

Formalize SysML

- Compatibility of SysML blocks
- Refinement of abstract SysML blocks

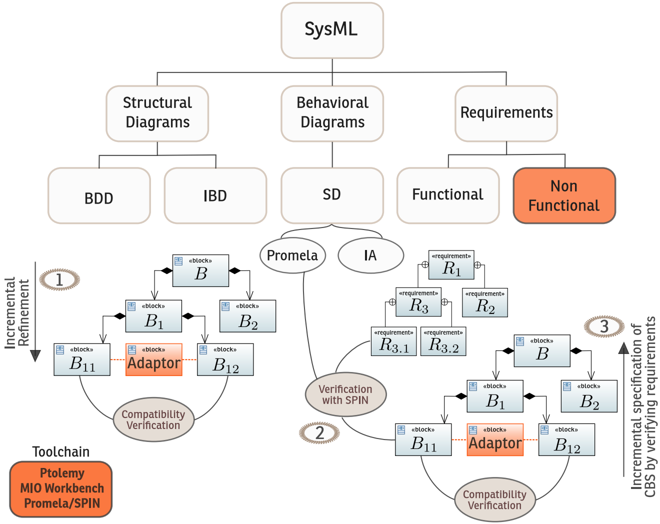
Verification of SysML Requirements

- SysML Requirements as LTL properties
- Promela description from SysML Sequence Diagrams
- Verification with SPIN model-checker

Incremental CBS Architecture Specification

- Guided by requirements
- Reuse of SysML blocks

Future Work


SysML
Structural Diagrams
Behavioral Diagrams
Requirements
BDD
IBD
SD
Functional
Non Functional
Promela
IA
Incremental Refinement
1
2
3
«block» B
«block» B1
«block» B2
«block» B11
«block» Adaptor
«block» B12
Compatibility Verification
«requirement» R1
«requirement» R2
«requirement» R3
«requirement» R3.1
«requirement» R3.2
Verification with SPIN
Incremental specification of CBS by verifying requirements
Toolchain
Ptolemy
MIO Workbench
Promela/SPIN

Future Work

Block adapters

Automatic generation of a block adapter when assembled blocks are incompatible

Non-functional requirements

Validation by simulation

Requirements when refining

Preservation over a decomposition

Toolchain for verification

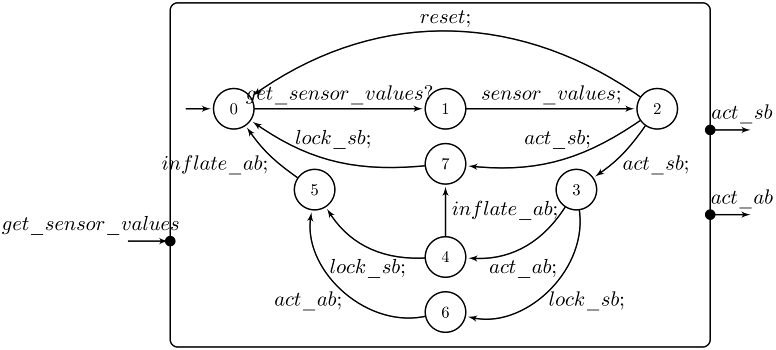
SysML, SD to IA, SD to Promela, Ptolemy, MIO Workbench, SPIN, SysML Model

Any questions ?

Thank you

for your attention

Final Architecture for the Vehicle Safety System



IA for the fourth iteration

Final Architecture for the Vehicle Safety System