

September 29th, 2008      Toulouse, France (in conjunction with MODELS 2008)

1st International Workshop on  
Model Based Architecting and Construction  
of Embedded Systems

(ACESMB 2008)

# Translating AADL into BIP - Application to the Verification of Real-time Systems

**M.Y.Chkouri**, A.Robert, M.Bozga, J.Sifakis



Laboratoire : VERIMAG  
Centre Équation - 2, avenue de Vignate 38610 GIÈRES





# Motivation

Provide a general methodology for transforming AADL models into BIP:

- AADL suffers from the absence of concrete operational semantics.
- Provide an execution environment for AADL models
- Enable the application of formal verification techniques already developed for BIP to AADL

# Outline

- Overview of AADL
- Overview of BIP
- Translation AADL to BIP
- Case study
- Conclusion



# Overview of AADL

*AADL = Architecture Analysis and Design Language*

- Standardized by the SAE (Society of Automotive Engineers).
- Dedicated to the modeling and specification of complex Real-time embedded systems.
- Describe the structure of component-based system as an assembly of software components mapped onto an execution platform.



# Component categories

**Software category**



- Data
- Subprogram
- Process
- Thread

**Execution platform category**



- Processor
- Memory
- Bus
- Device

**Composite category**



- System



# Software category (1/2)

## Data

- The ***data*** component type represents a data type in the source text that defines a representation and interpretation for instances of data.

## Subprogram

- A ***subprogram*** component represents an execution entrypoint in source text.
- A subprogram call sequence is declared in a subprogram or thread implementation.

```
data Person
end Person;
data implementation Person.impl
  subcomponents
    Name : data string;
    Adress: data string;
    Age : data integer;
end Person.impl;
```

```
subprogram operation
  features
    A: in parameter integer;
    B: in parameter integer;
    result: out parameter integer;
end operation;
```



# Software category (2/2)

## Thread

- A **thread** represents a sequential flow of control that executes instructions within a binary image produced from source text.
- A thread always executes within the virtual address space of a process;
- Several types of thread exist :
  - Periodic , Sporadic , Aperiodic, Background.

## Process

- A **process** represents a virtual address space.
- To be complete, the implementation of a process must contain at least one thread or thread group subcomponent.

### thread sensor features

inp : **in data port** integer;  
 outp : **out event port**;

### properties

**Dispatch protocol**=>Periodic;  
**Period** => 20ms;

**end sensor**;

### process implementation Partition.Impl subcomponents

Sensor\_A : **thread** Sensor Thread.A;  
 Data\_Fusion: **thread** Fusion Thread.Impl;  
 Alm 1 : **thread** Alm Thread.Impl;

### connections

**data port** Sensor A.outp->Data Fusion.inpA;  
**event port** Sensor A.launch alm->Alm.launch

A;  
**end** Partition.Impl;



# Execution platform category



- A **processor** is the execution platform component that is capable of scheduling and executing threads.



- A **memory** component represents an execution platform component that stores binary images.



- A **bus** component represents an execution platform component that can exchange control and data between memories, processors, and devices.



- A **device** component represents an execution platform component that provides an interface with the external environment.



# System

## System

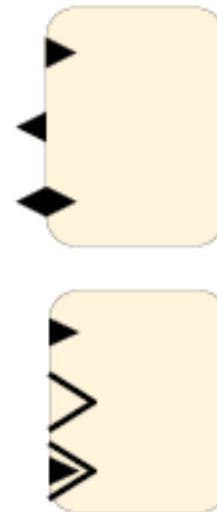
- A **system** component represents an assembly of software and execution platform components.
- It is the only composite category.

```
system Platform  
end Platform;  
system implementation Platform.Impl  
subcomponents  
    Part : process Partition.Impl;  
    p : processor myProcessor ;  
    ...  
end Platform.Impl;
```



# Connection & Port

- A **connection** is a linkage that represents communication of data and control between components.
- Types of connections:
  - Port connection
  - Parameter connection
  
- A **port** is a logical connection point between components that can be used for the transfer of control and data.
- Three directions:
  - input port (*in*)
  - output port (*out*),
  - bidirectional port (*in out*).
- Three types of port:
  - data port,
  - event port,
  - event data port.



# Outline

- Motivation
- Overview of AADL
- **Overview of BIP**
- Translation AADL to BIP
- Case study
- Conclusion

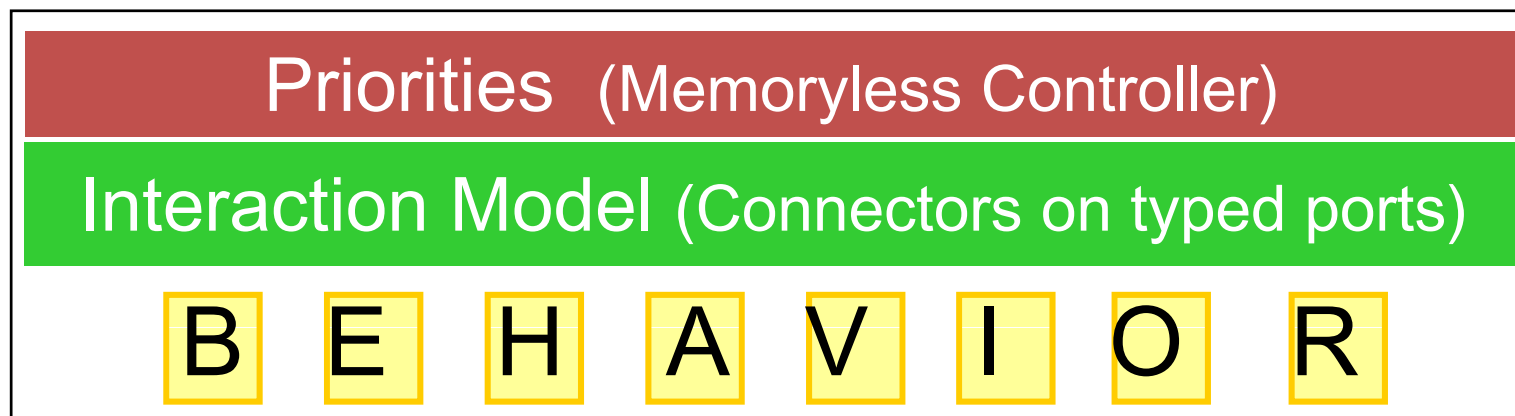


# Overview of BIP

Component based modeling: The BIP framework

BIP = *Behavior Interaction Priority*

BIP is a framework for modelling heterogeneous real-time components.



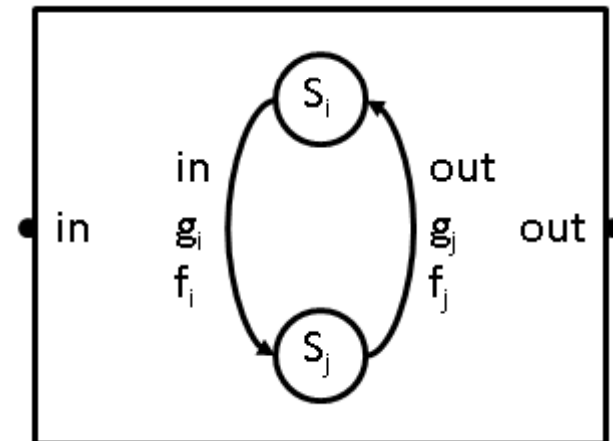


# BIP framework - Atomic component

## Atomic component :

An **atomic** component is composed of:

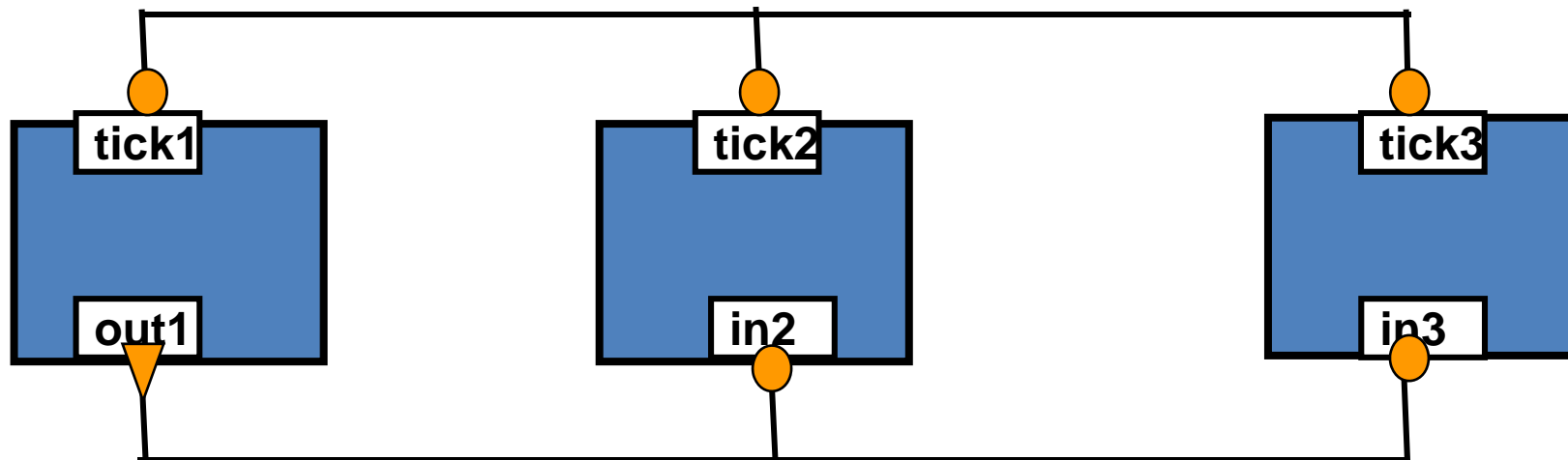
- a set of ports, e.g, {in, out}
- a set of control locations, e.g, { $S_i$ ,  $S_j$ }
- a set of variables,
- a set of transitions





# BIP framework - Composition

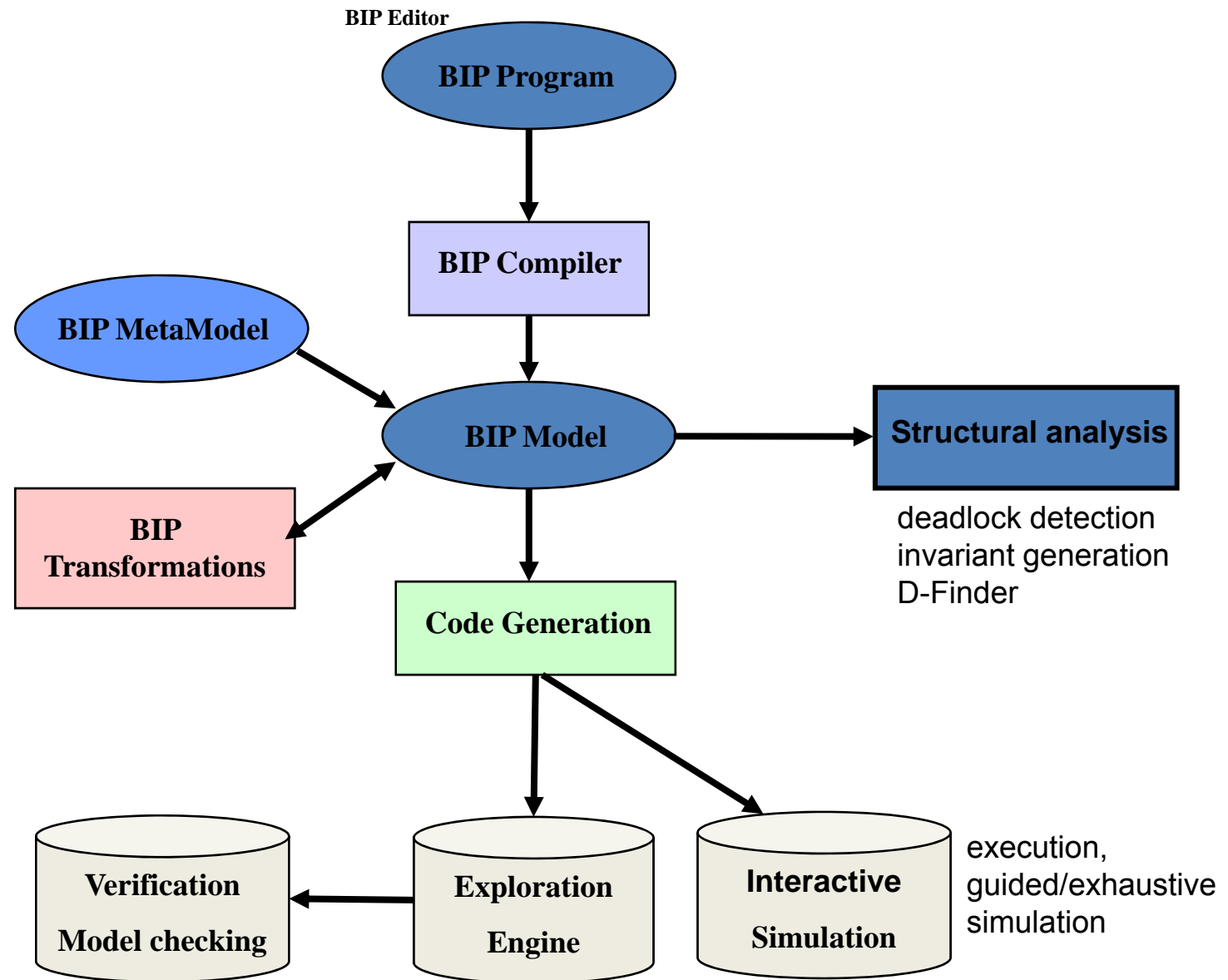
- A **connector** is a set of ports which can be involved in an interaction.
- Port types (**complete** ▲, **incomplete** ●) are used to distinguish between ports which **may** or **must** interact.



Interactions:

$\{\text{tick1}, \text{tick2}, \text{tick3}\}$ ,  $\{\text{out1}\}$ ,  $\{\text{out1}, \text{in2}\}$ ,  $\{\text{out1}, \text{in3}\}$ ,  $\{\text{out1}, \text{in2}, \text{in3}\}$

# BIP Tools



# Outline

- Motivation
- Overview of AADL
- Overview of BIP
- **Translation AADL to BIP**
- Case study
- Conclusion



# Translation from AADL to BIP

## ✚ Structural translation summary:

AADL	BIP
Software component: <ul style="list-style-type: none"><li>▪ Subprogram</li><li>▪ Data</li><li>▪ Thread</li><li>▪ Process</li></ul>	Atomic/Compound component Data : C/C++ structure Atomic component Atomic component
Hardware component: <ul style="list-style-type: none"><li>▪ Processor (scheduling)</li><li>▪ Device</li></ul>	Atomic component Atomic component
▪ System	Compound component
▪ Connection	Connector
▪ Annex behavior	Behavior (state/transition)



# Translation from AADL to BIP

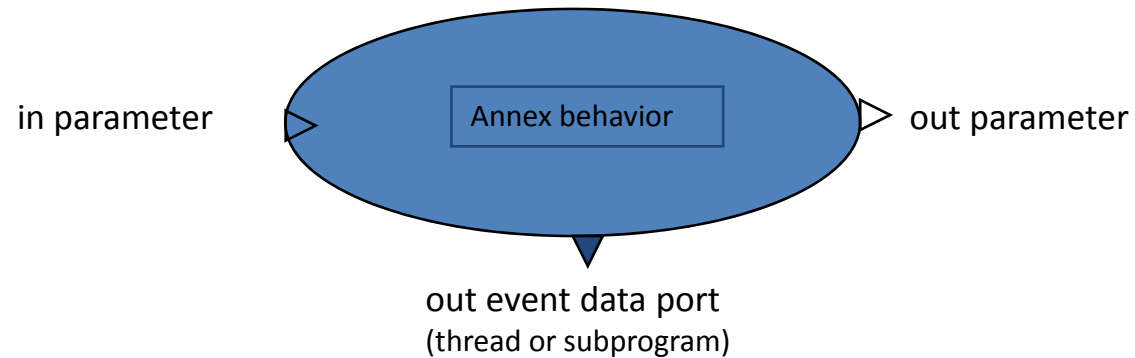
## ✚ Structural translation summary:

AADL	BIP
Software component: <ul style="list-style-type: none"><li>▪ Subprogram</li><li>▪ Data</li><li>▪ Thread</li><li>▪ Process</li></ul>	Atomic/Compound component Data : C/C++ structure Atomic component Atomic component
Hardware component: <ul style="list-style-type: none"><li>▪ Processor (scheduling)</li><li>▪ Device</li></ul>	Atomic component Atomic component
▪ System	Compound component
▪ Connection	Connector
▪ Annex behavior	Behavior (state/transition)

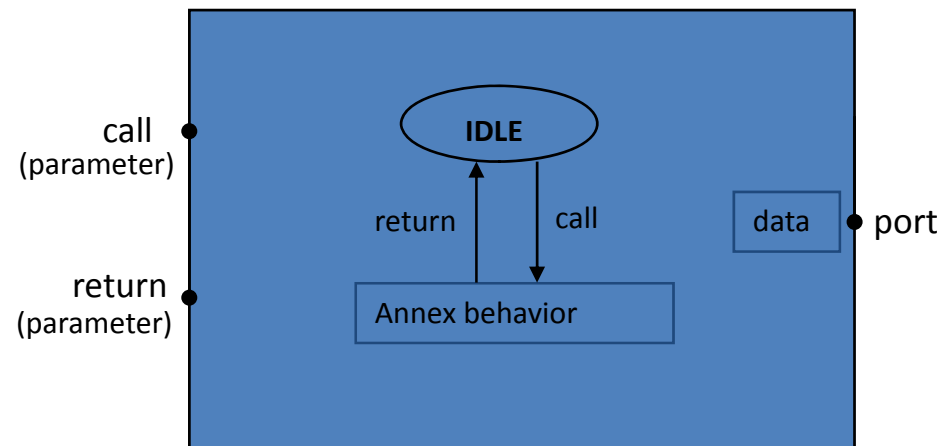


# AADL subprogram : BIP model

AADL :

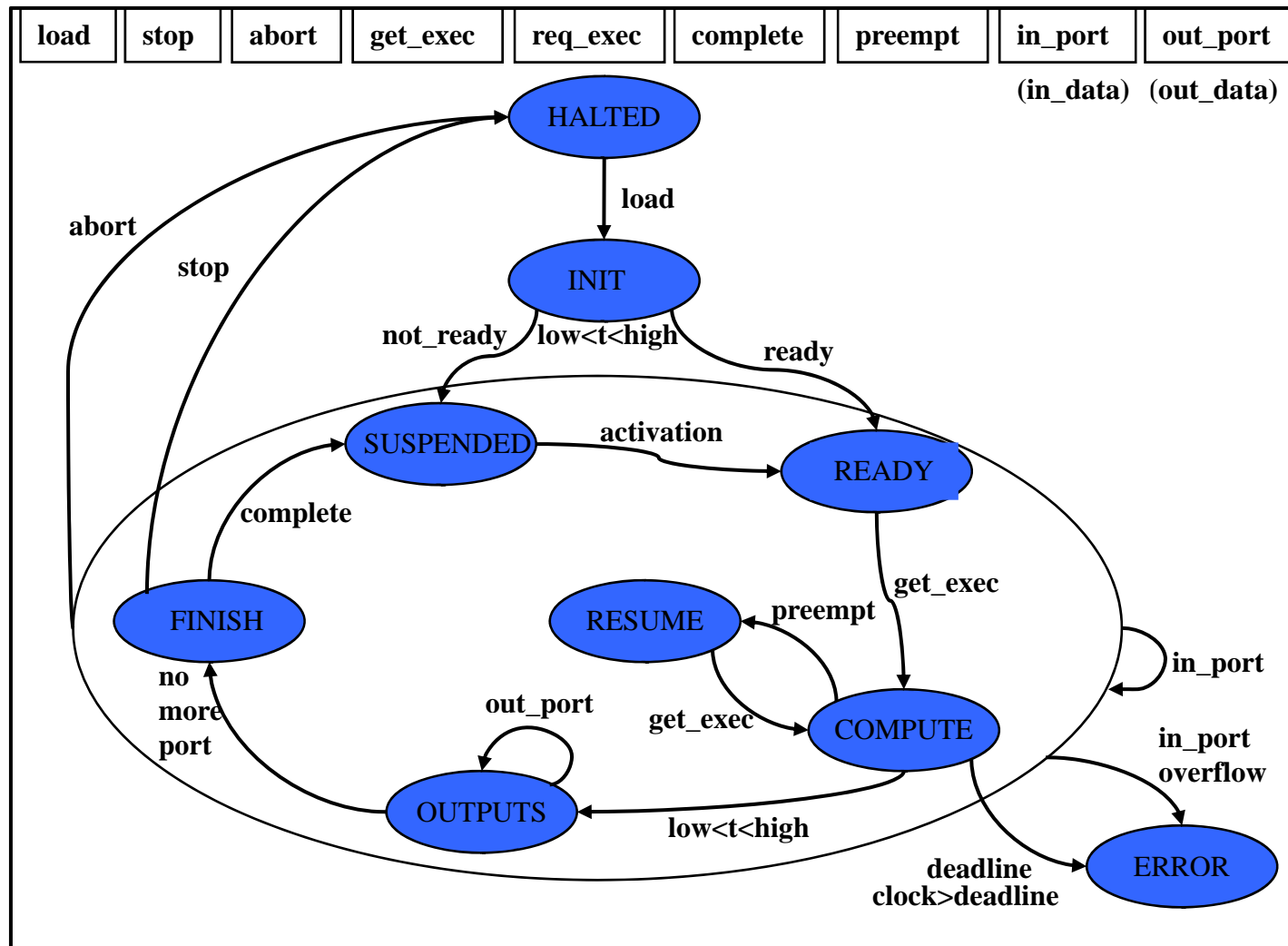


BIP :



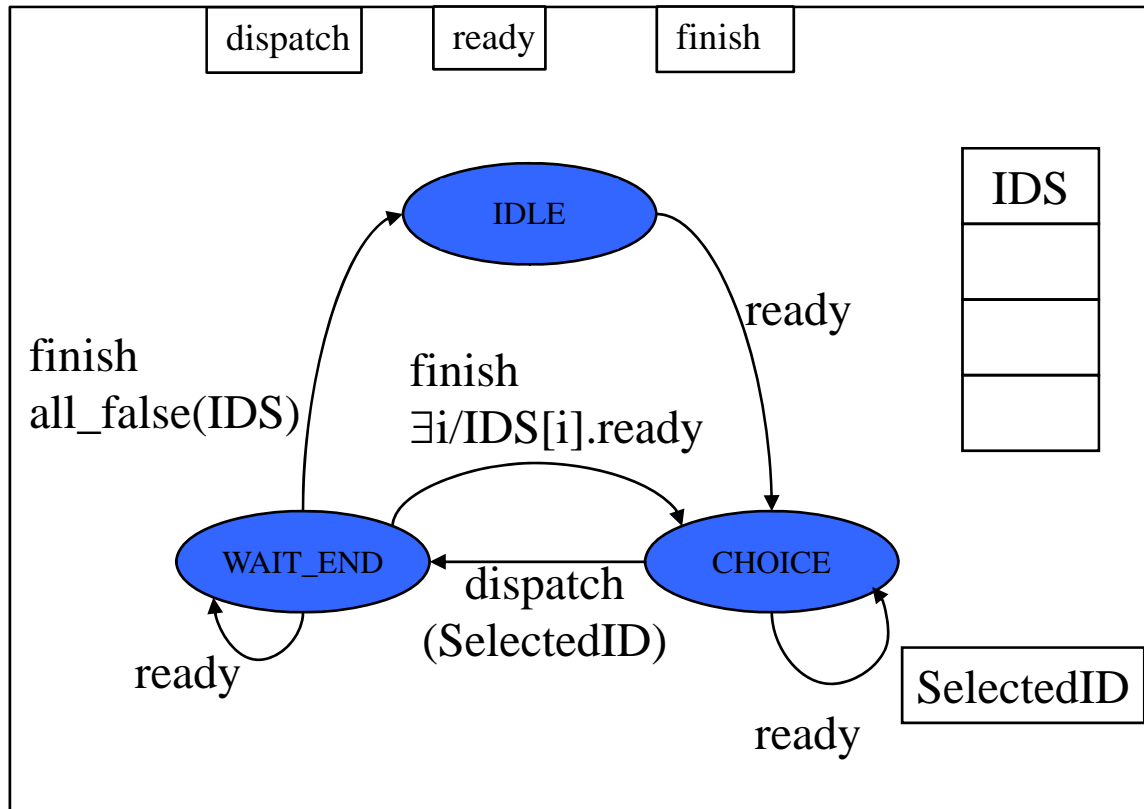


# AADL thread : BIP model



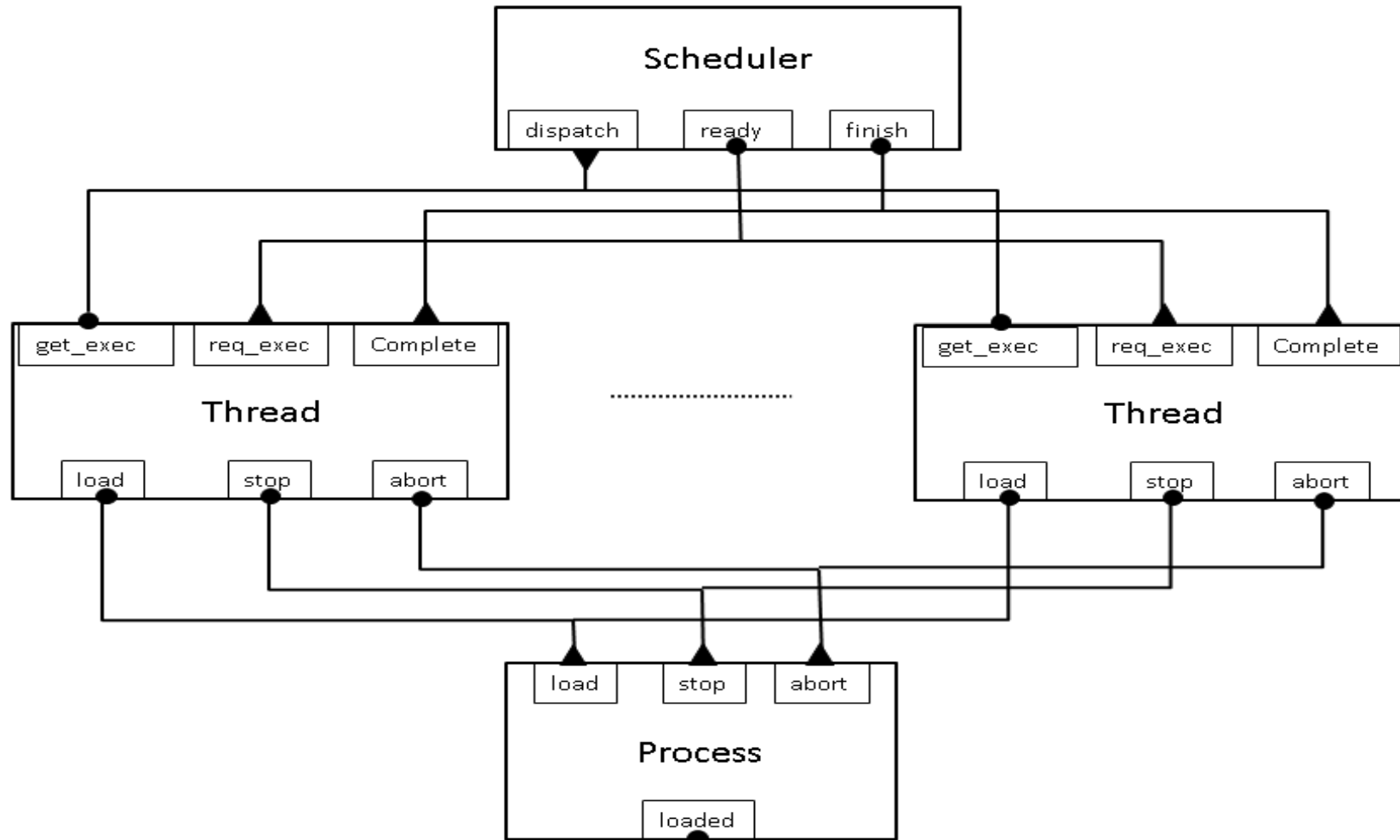


# AADL Processor : BIP component





# System : BIP compound component





# Annex Behavior Specification

The behavioral annex describes a transition system

[ *annex behavior\_specification* {\*\*

[ *state variables*

(*Identifier* : *data\_type*;) + ]



Included in the variables part

[ *initial* (<*assignment*> ; ) + ]



Included in the Initialization part

*states*

(*state\_identifier* : [*initial*] [*return*] [*complete*] *state*;) +

*transitions*

( <*state\_identifier*> -[ <*guard*> ] -> <*state\_identifier*> { <*action*>\* }; ) +

\*\*};

]



# Guard

AADL :

```
<guard> ::=  
  [on <expression> -->] <event> [when<expression>]  
  | <expression>
```

BIP :

```
on <event> [provided <expression>]  
| provided <expression>
```

*when* part expresses a past condition over the data to be read.

# Action

AADL :

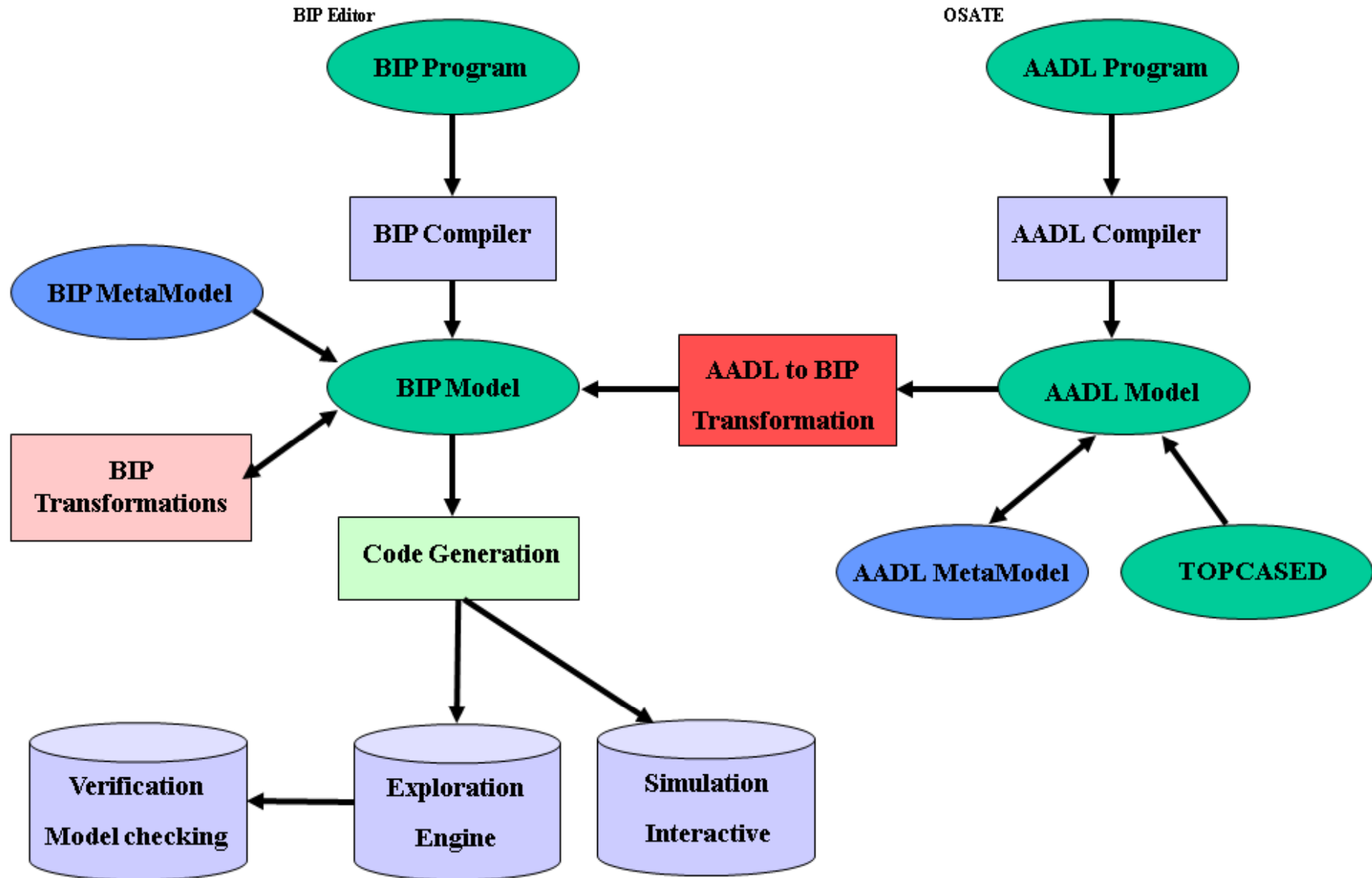
```
<action> ::=  
  computation ( expression , expression ) ;  
  | delay ( expression , expression ) ;  
  | communication ;  
  | assignment ;  
  | if ( expression ) action  
    ( elseif ( expression ) action)* ( else action ) ?  
  end if ;
```

BIP :

**Transition**  
**Or**  
**Set of transition connected**

- expresses use of the cpu for a non-deterministic period of time between min and max.
- expresses a suspension for a non-deterministic period of time between min and max.

# Tool architecture



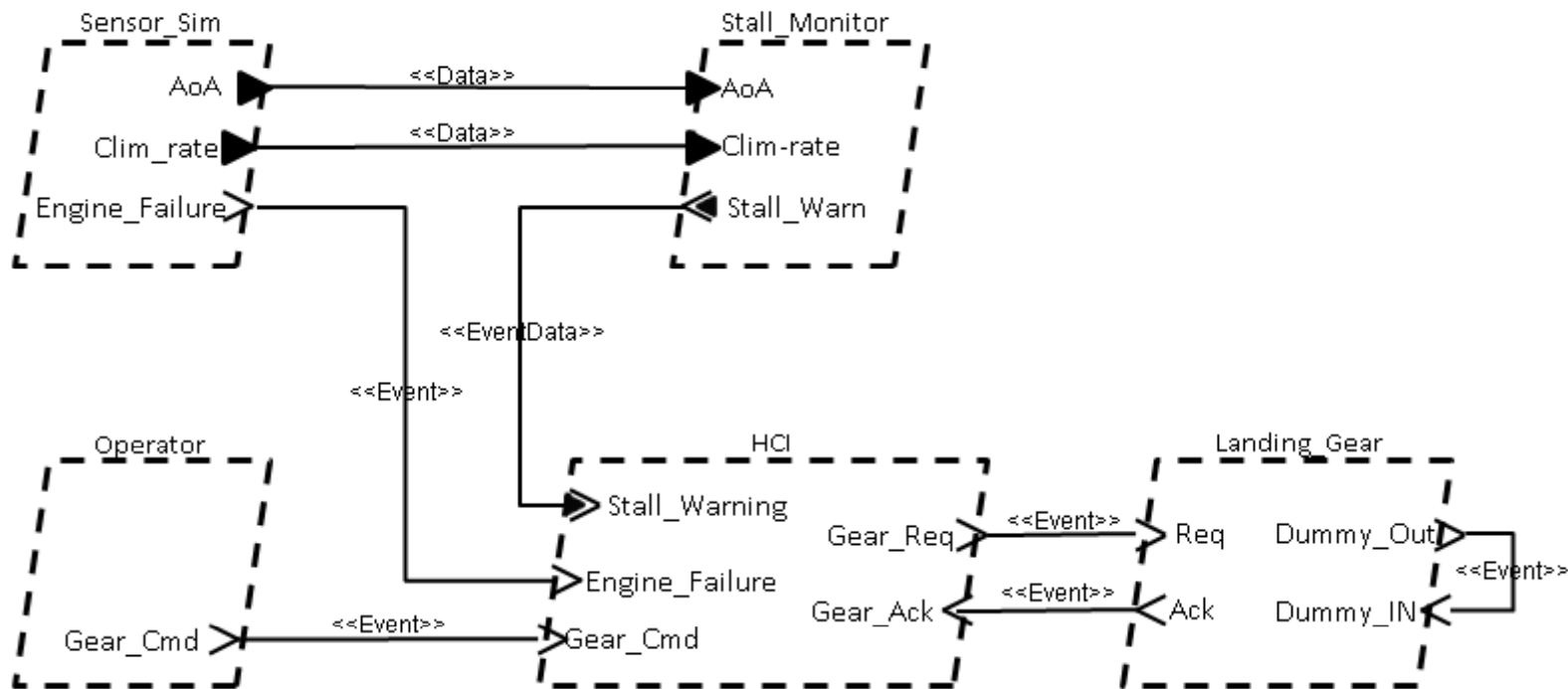
# Outline

- Motivation
- Overview of AADL
- Overview of BIP
- Translation AADL to BIP
- **Case study**
- Conclusion



# Case study (1/4)

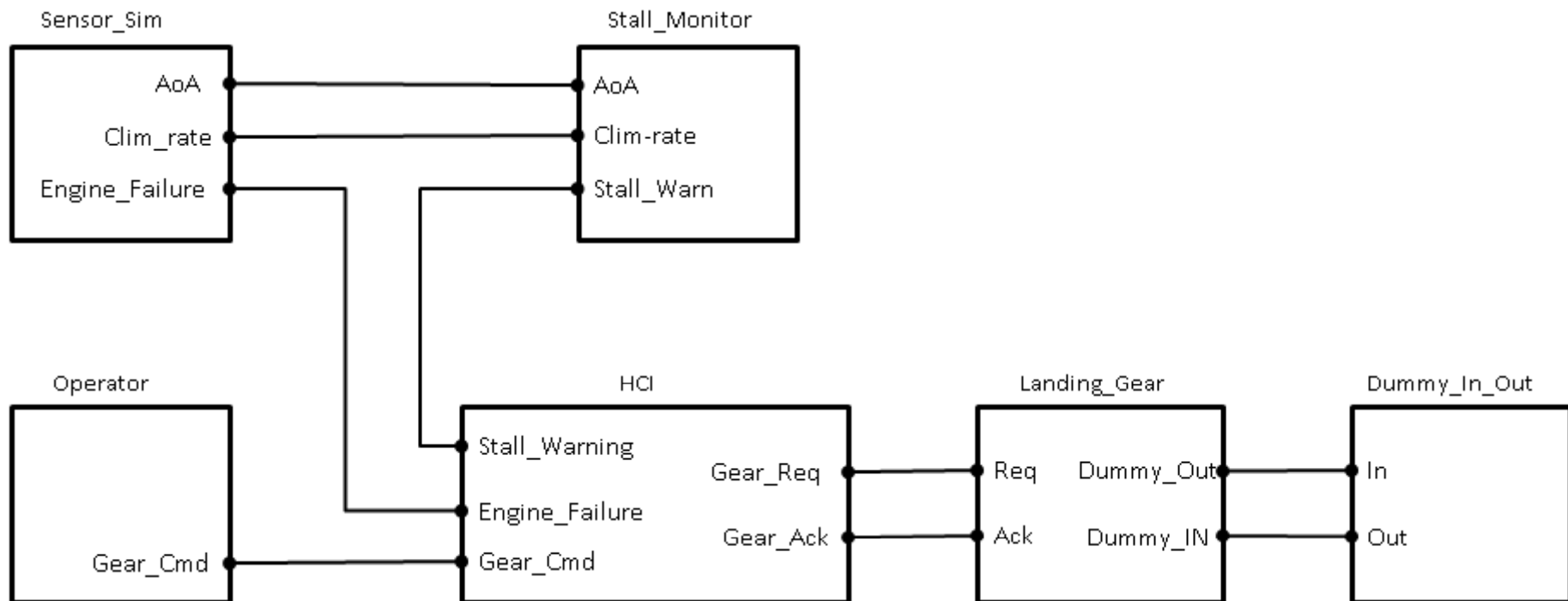
**Flight computer** : <http://aadl.enst.fr/arc/doc/>





# Case study (2/4)

## BIP Flight computer :





# Case study (3/4)

## BIP Verification :

BIP exploration engine, generates a Labeled Transition System (LTS).

### + Model checking by Aldebaran:

- Checks for deadlock-freedom.

### + Model checking with observers:

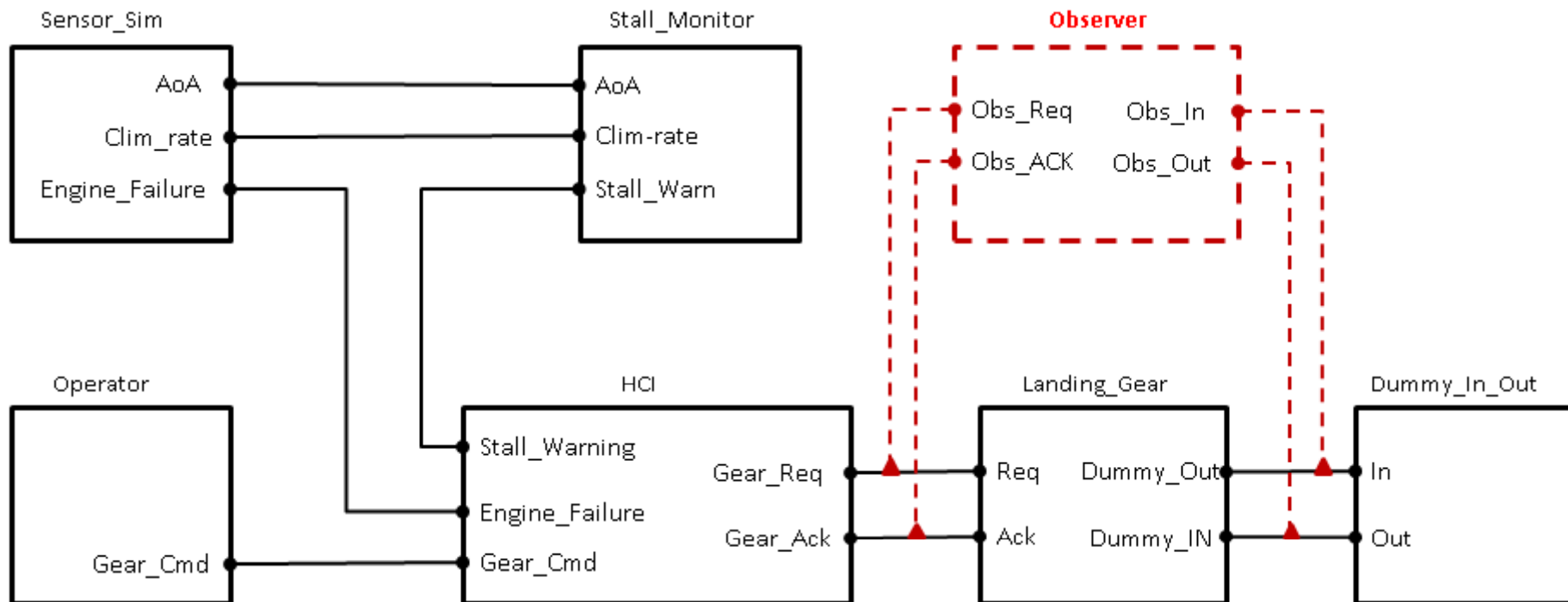
Observers allow us to express in a simple manner most safety requirements.

- Verification of thread deadlines.
- Verification of synchronization between components:



# Case study (4/4)

## BIP Flight computer :



# Outline

- Motivation
- Overview of AADL
- Overview of BIP
- Translation AADL to BIP
- Case study
- **Conclusion**

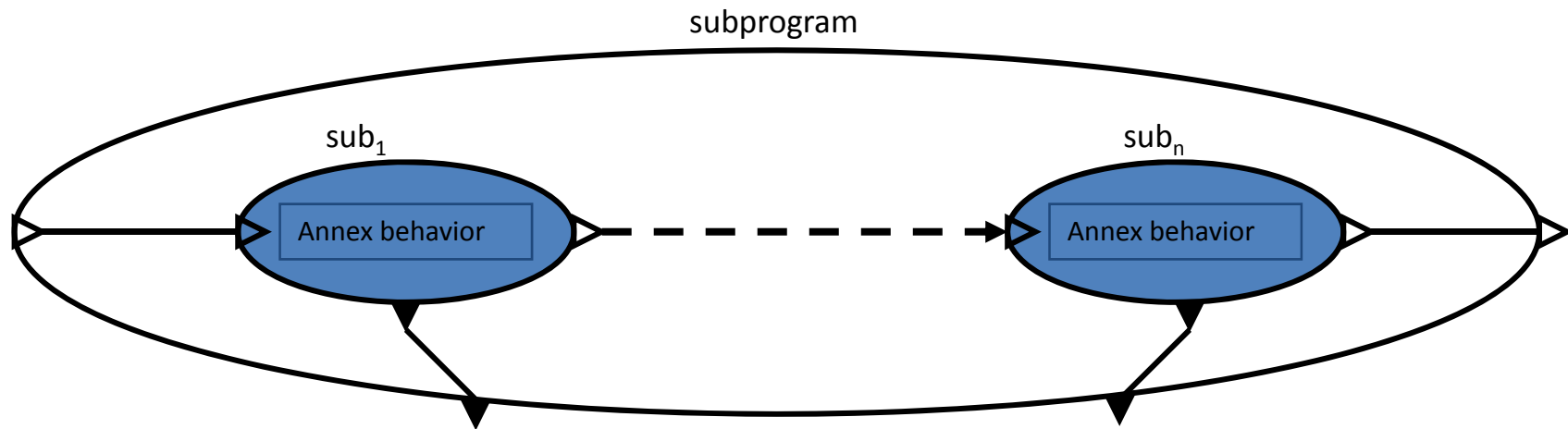


# Conclusion

- We provide a translation from AADL to BIP, which has an operational semantics formally defined in terms of labelled transition systems.
- Translation allows simulation of AADL models, as well as application verification techniques, such as state exploration (using IF toolset) or component-based deadlock detection (D-Finder tool).
- **Limitation** : there are AADL features ignored : bus, memory, ...
- **Future work** : incorporating features that will appear with V2.0 of the AADL standard.

Thank you

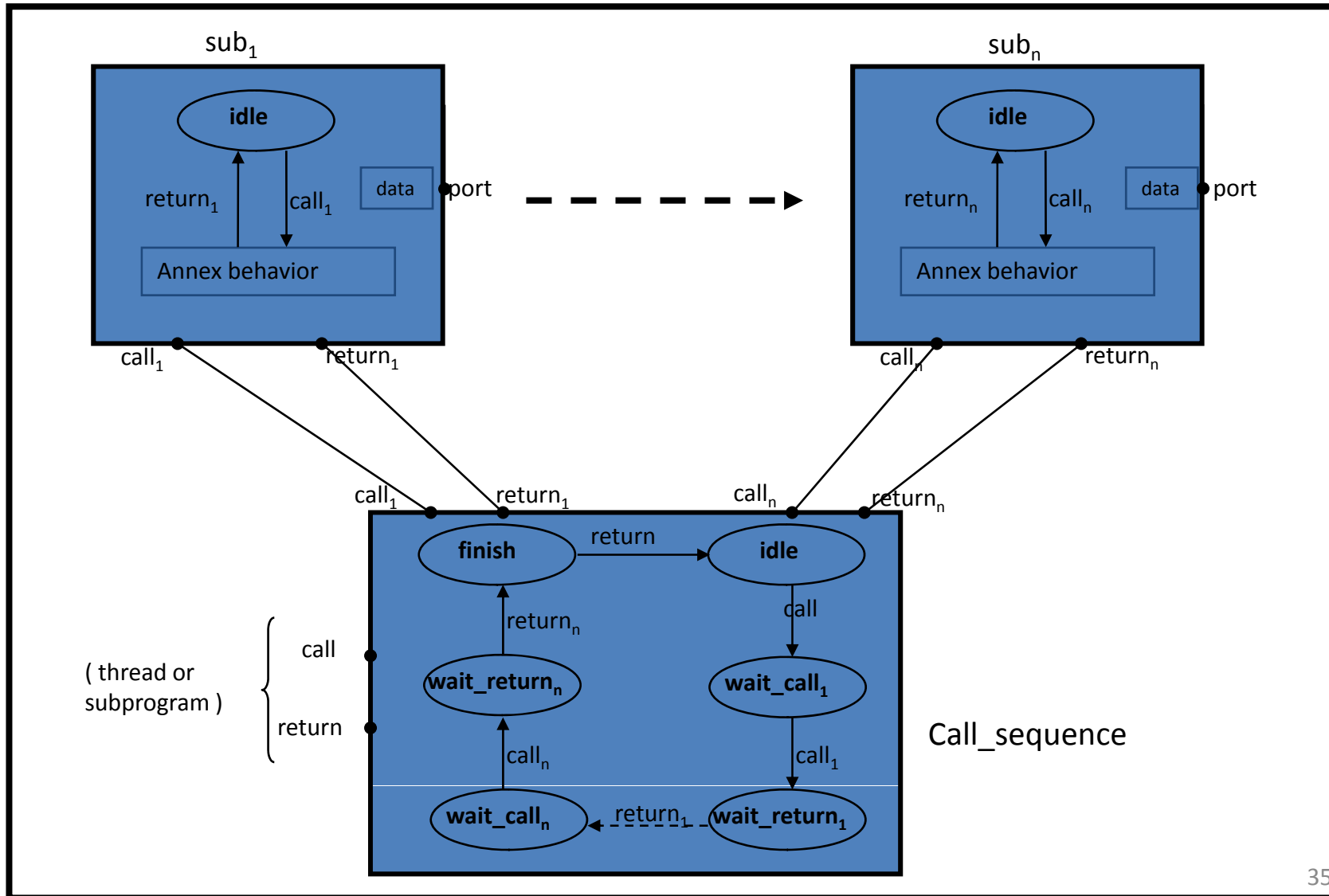
# AADL subprogram



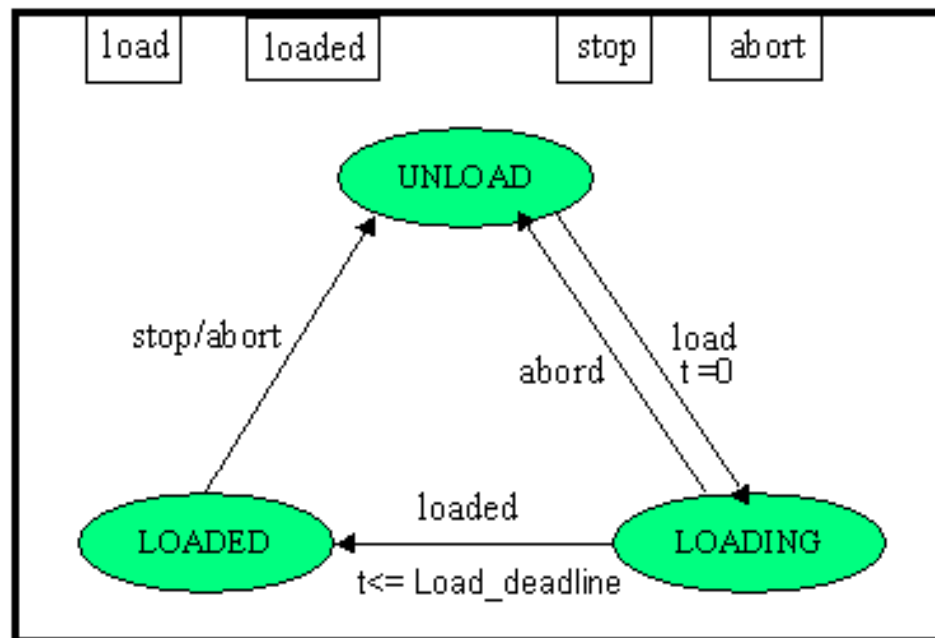
▷ parameter connexion

▶ out event data port connexion

# BIP : Compound component



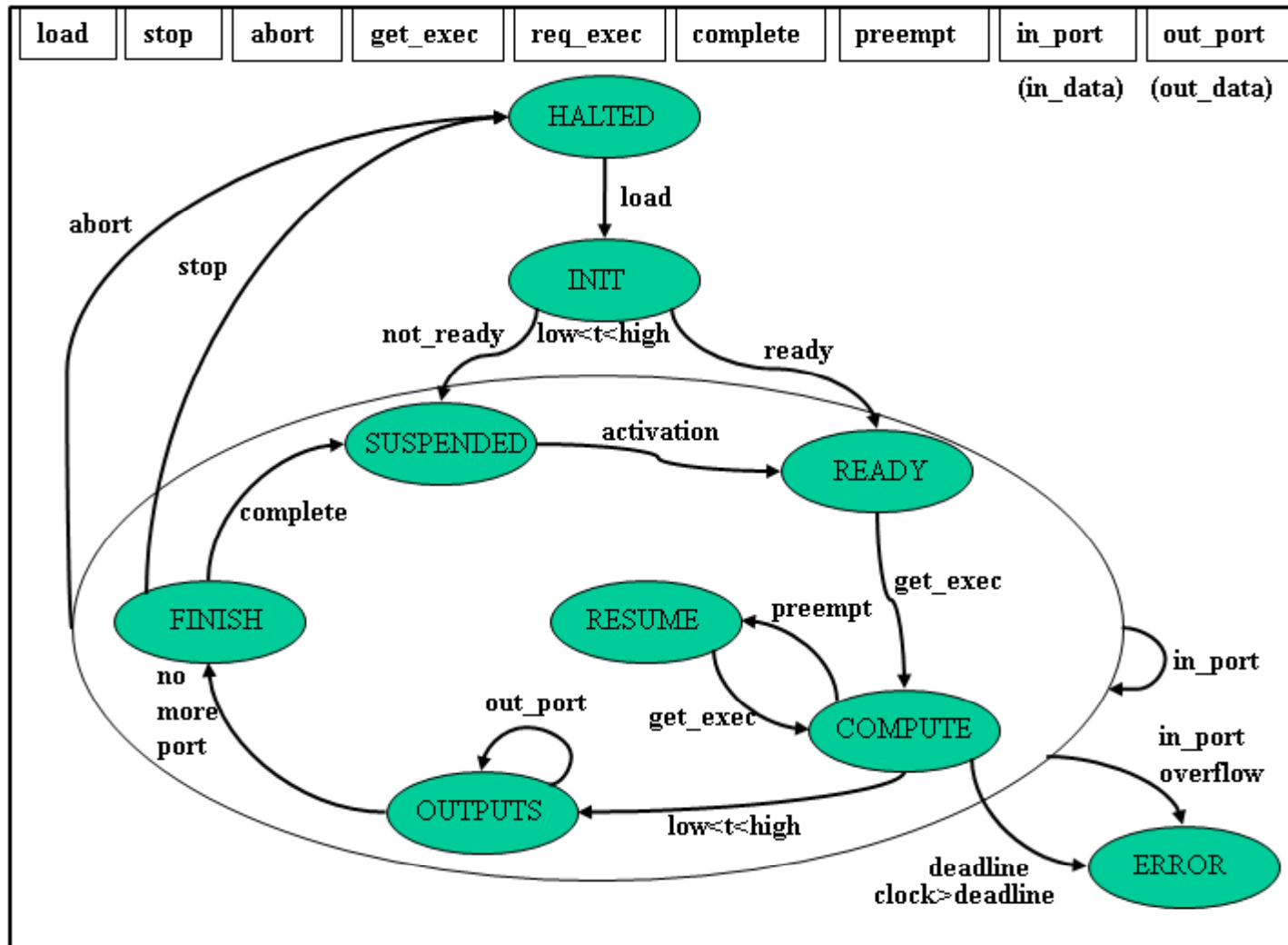
# AADL Processes: BIP Component



Process States and Transition



# AADL thread : BIP model





# AADL Processor : BIP component

