

October 6th, 2009

Denver, USA (in conjunction with MODELS 2009)

2nd International Workshop on  
Model Based Architecting and Construction  
of Embedded Systems

(ACESMB 2009)

# Prototyping of Distributed Embedded Systems Using AADL

M.Y.Chkouri, M.Bozga



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





# Motivation

- Distributed applications are used in many safety-critical domains such as space or avionics.
- Designing distributed systems demands more attention and rigour.
- Conform to many stringent functional and non-functional requirements





# Motivation

Provide a general methodology for transforming distributed AADL models into BIP:

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

# Outline

- Overview of AADL
- Overview of BIP
- Prototyping of Distributed Implementation
- 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

## 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.

## 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.



# 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**
- Prototyping of Distributed Implementation
- Case study
- Conclusion

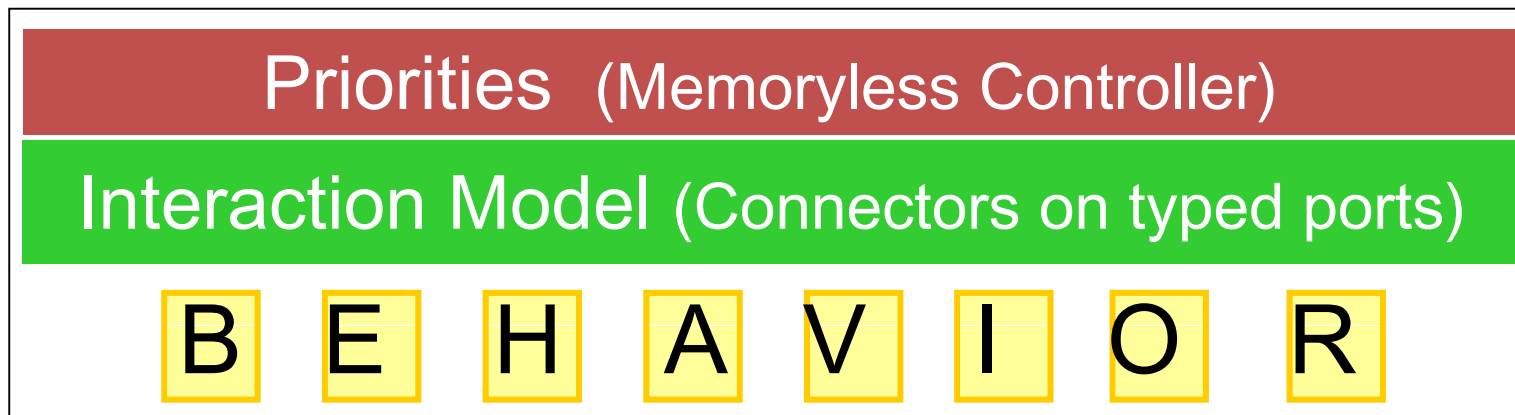


# Overview of BIP

Component based modeling: The BIP framework

BIP = *Behavior Interaction Priority*

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



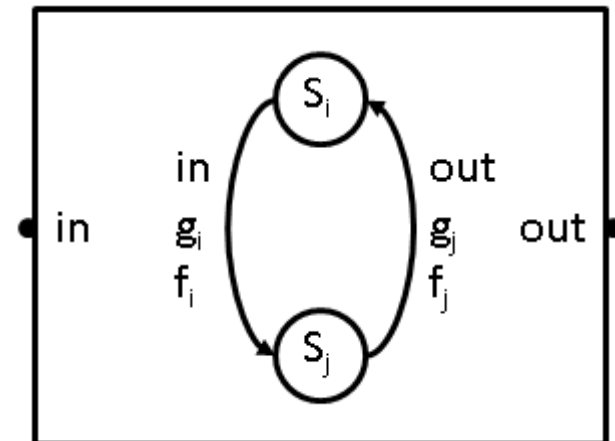


# BIP framework - Atomic component

## Atomic component :

An **atomic** component is composed of:

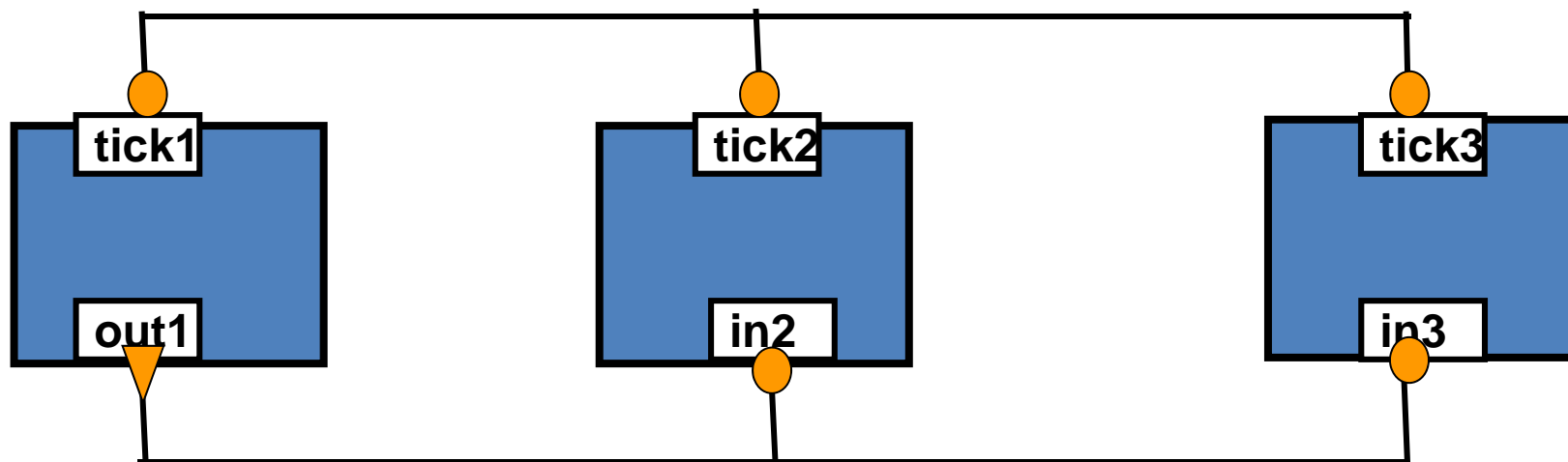
- a set of ports, e.g, {in, out}
- a set of control locations, e.g, { $S_i$ ,  $S_j$ }
- a set of variables,
- a set of transitions





# BIP framework - Composition

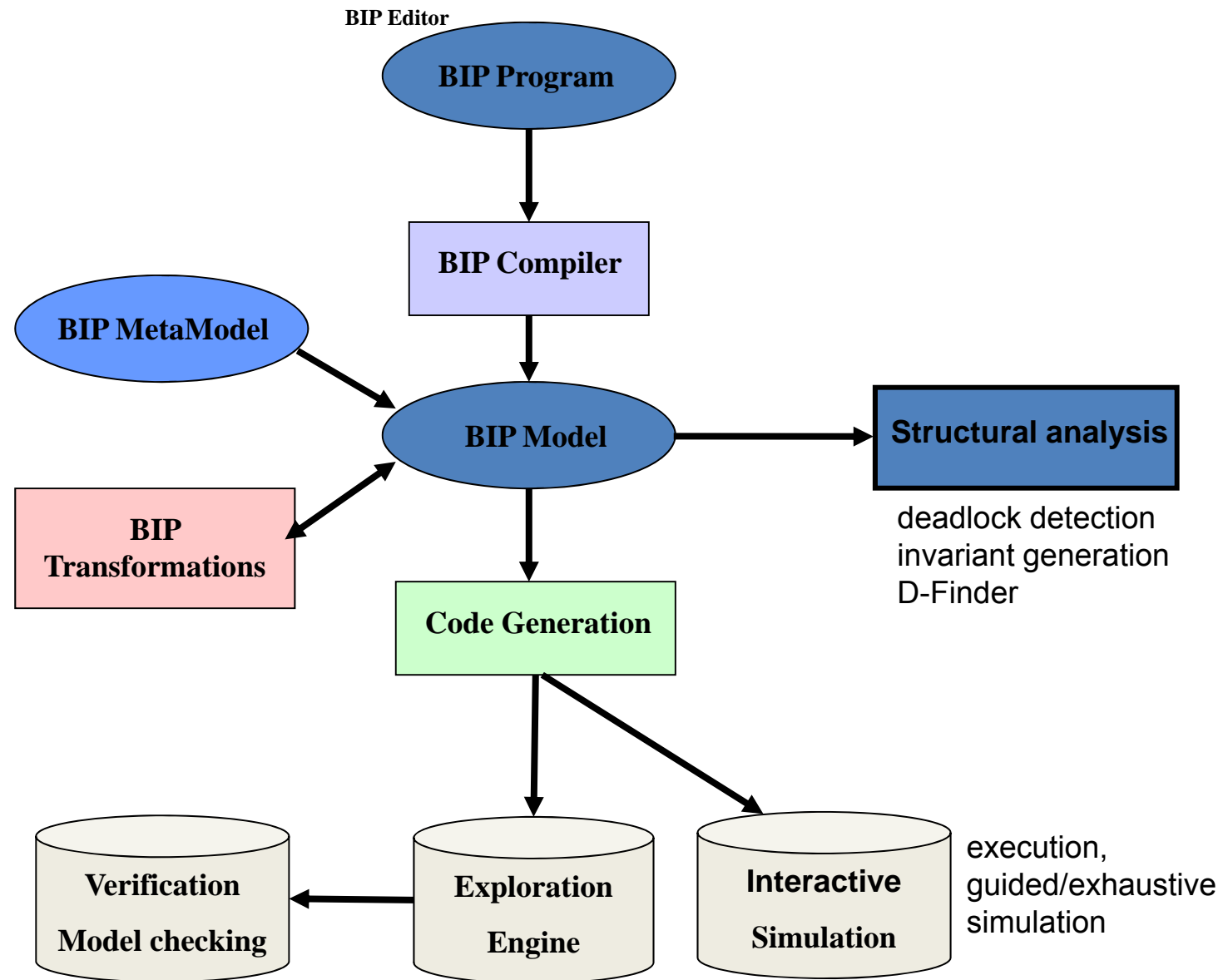
- A **connector** is a set of ports which can be involved in an interaction.
- Port types (**complete** ▲, **incomplete** ●) are used to distinguish between ports which **may** or **must** interact.



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
- **Prototyping of Distributed Implementation**
- Case study
- Conclusion



# Translation from AADL to BIP: Basic principles

## ✚ 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)



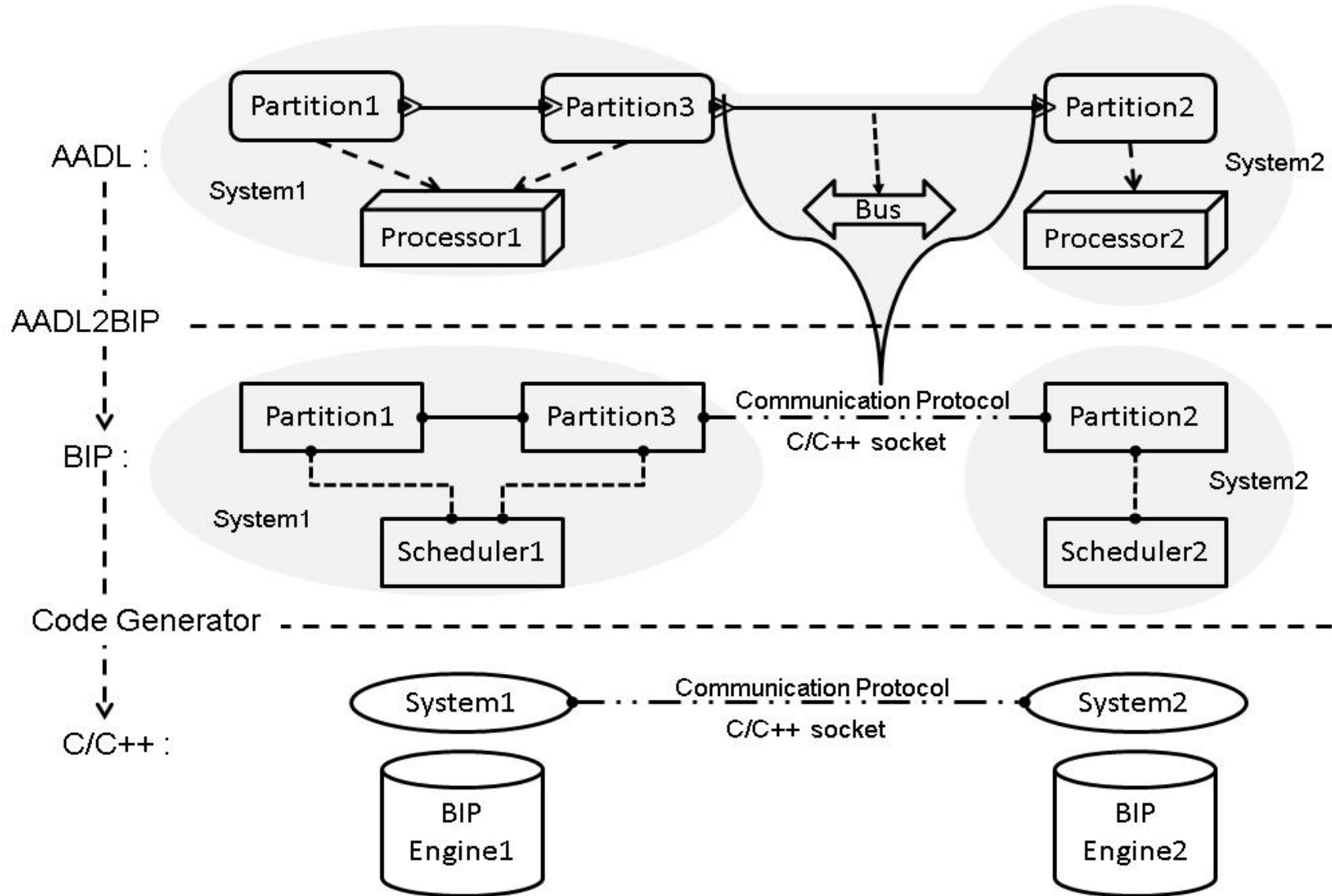
# Prototyping Distributed Implementation (1/4)

## AADL Requirement :

1. Data types and related functions to operate on them
2. Supporting runtime entities (threads) and interactions between them (through ports and connections)
3. Association of subprograms to threads
4. Mapping of threads onto processes and binding processes to hardware entities to form the deployed system.
5. Binding connections to buses to form the deployed system.



# Prototyping Distributed Implementation (2/4)





# Prototyping Distributed Implementation (3/4)

## Data exchange:

- Client: Convert data into encoded version before sending.
- Server: Convert back data after reception.

Encoding/Decoding functions to be provided by the user for each data type.

## Communications:

- running on one computer are managed by the BIP Engine,
- running on more computers connected by a network, use a network communication protocol:
  - rely on TCP/IP network communication
  - use sockets: are associated with the concept of network communication in the form of client-server.



# Prototyping Distributed Implementation (4/4)

## Client side :

The steps involved in establishing a communication protocol on the client side:

- (1) Create a communication protocol;
- (2) Connect the communication to the address of the server;
- (3) Send and receive data.

## Server side :

The steps involved in establishing a communication protocol on the server side :

- (1) Create a communication protocol;
- (2) Bind the communication to an address. For a server, an address consists of a port number on the host machine;
- (3) Listen for connections;
- (4) Accept a connection. This call typically blocks until a client connects with the server;
- (5) Send and receive data.

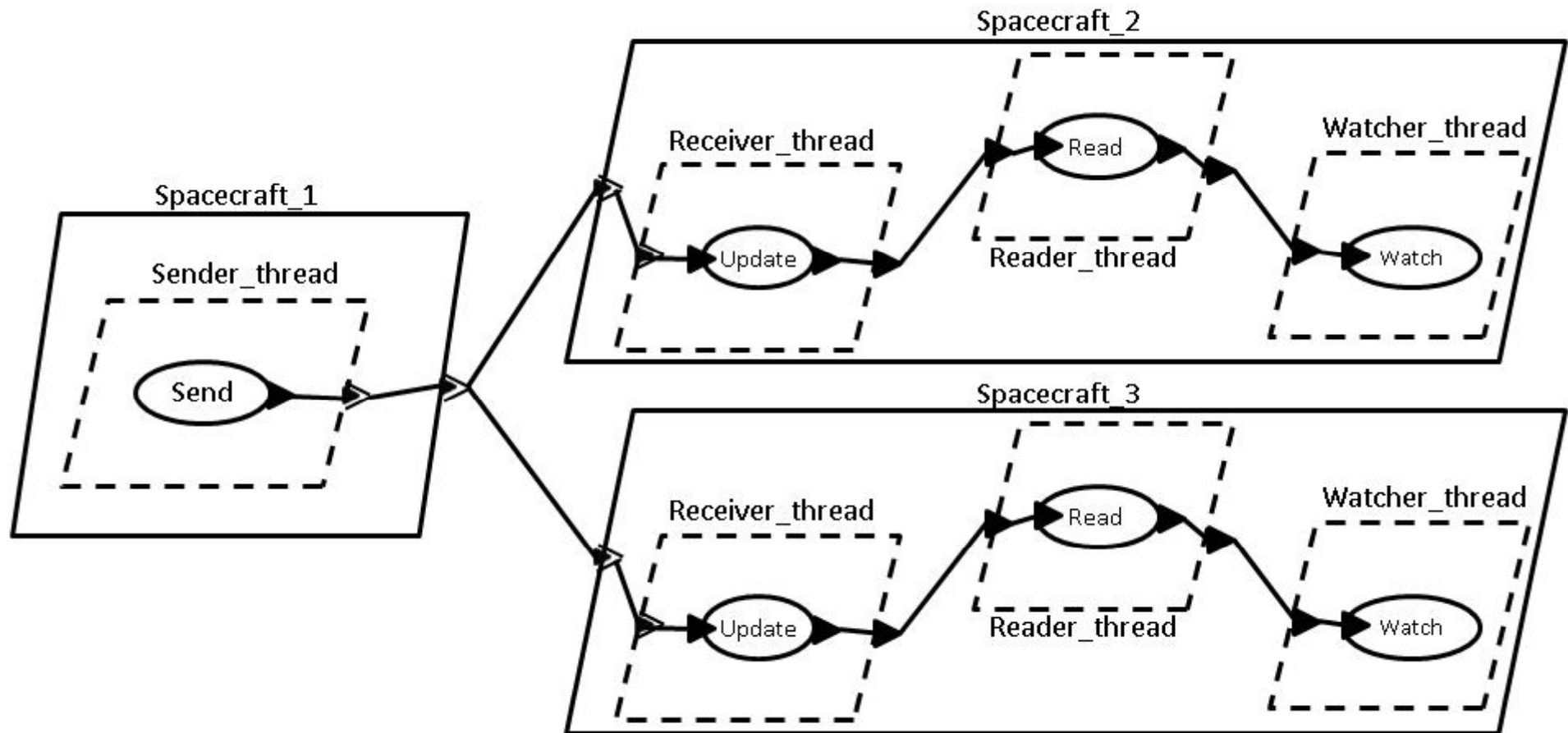
# Outline

- Motivation
- Overview of AADL
- Overview of BIP
- Prototyping of Distributed Implementation
- **Case study**
- Conclusion



# Case study (1/4)

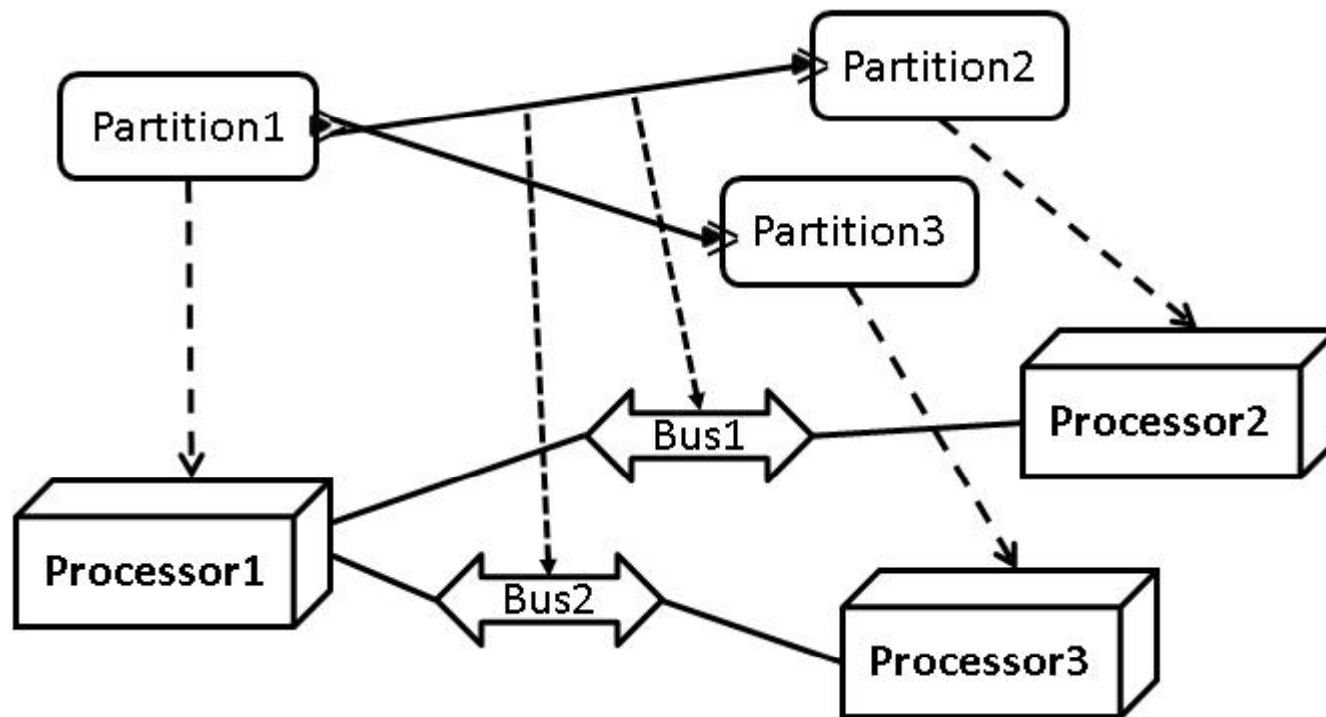
## Software view:





# Case study (2/4)

## hardware view:





# Case study (3/4)

## BIP model :

- We generate for each AADL partition mapped to the processor its corresponding description in BIP, and for each connection mapped to the bus a network communication protocol (sender/receiver).
- BIP help to analyse the case study in a native platform (PC) in order to easily debug and evaluate it before running it on an embedded platform.
- BIP gives a strong semantique to AADL model

## Comparison between AADL & BIP

	AADL	BIP		
		Spacecraft_1	Spacecraft_2	Spacecraft_3
Components	20	4	8	8
Connectors	21	8	18	18
Lines of code	350	250	600	600



# Case study (4/4)

## Distributed Execution :

### Computer 1 :

```

X chkouri@ventoux: /home/chkouri/instalation_test/eclipse/runtime-EclipseAp...
scheduler : BIP_Top/tick1/CPU_SC_1_inst;tick1
scheduler : BIP_Top/tick1/CPU_SC_1_inst;tick1
scheduler : BIP_Top/tick1/CPU_SC_1_inst;tick1
scheduler : BIP_Top/tick1/CPU_SC_1_inst;tick1
scheduler : CPU_SC_1_inst/readyConn0/Sender;require_exec_unit|CPU_SC_1_scheduler;ready
13200 ;thread 0 is ready!;0;T
1;0;T
13200 ;***** choose thread 0 *****
1;0;F
IDLE to CHOICE< selectedID=0, ID =0
scheduler : CPU_SC_1_inst/dispatchConn0/Sender;get_exec_unit|CPU_SC_1_scheduler;dispatch
CHOICE to WAIT_END< selectedID=0, ID =0
scheduler : CPU_SC_1_inst/Sender_Thread_Wrapper_impl_call_cnx/Sender_Thread_Wrapper_impl_inst;CALL|Sender;Sende
Local object , Initial value 142
scheduler : CPU_SC_1_inst/Sender_Thread_Wrapper_impl_return_cnx/Sender;Sender_Thread_Wrapper_impl_return|Sender
scheduler : CPU_SC_1_inst/X/Sender;Data_Source_port
scheduler : CPU_SC_1_inst/finishConn0/Sender;release_exec_unit|CPU_SC_1_scheduler;finish
WAIT_END to CHOICE_OR_IDLE< selectedID=0, ID =0
1;0;F
1;0;F
CHOICE_OR_IDLE to IDLE< selectedID=0, ID =0
scheduler : BIP_Top/tick1/CPU_SC_1_inst;tick1
scheduler : BIP_Top/tick1/CPU_SC_1_inst;tick1
scheduler : BIP_Top/tick1/CPU_SC_1_inst;tick1

```

### Spacecraft\_2

### Computer 2 :

```

X chkouri@ventoux: /home/chkouri/instalation_test/eclipse/runtime-EclipseAp...
3;1;F;T;F
80 ;***** choose thread 1 *****
3;1;F;F;F
IDLE to CHOICE< selectedID=1, ID =1
scheduler : CPU_SC_2_inst/dispatchConn1/Receiver;get_exec_unit|CPU_SC_2_scheduler;dispatch
CHOICE to WAIT_END< selectedID=1, ID =1
scheduler : CPU_SC_2_inst/Update_impl_call_cnx/Update_impl_inst;CALL|Receiver;Update_impl_call
Updating the local object, 137 , 118
Local object updated, New value 137
scheduler : CPU_SC_2_inst/Update_impl_return_cnx/Receiver;Update_impl_return|Update_impl_inst;RETURN
scheduler : CPU_SC_2_inst/Receiver_Protected_Local/Local_Object;update_port|Receiver;Protected_Local_port
receiving event update port
scheduler : CPU_SC_2_inst/finishConn1/Receiver;release_exec_unit|CPU_SC_2_scheduler;finish
WAIT_END to CHOICE_OR_IDLE< selectedID=1, ID =1
3;1;F;F;F
3;1;F;F;F
CHOICE_OR_IDLE to IDLE< selectedID=1, ID =1
scheduler : BIP_Top/tick1/CPU_SC_2_inst;tick1
scheduler : CPU_SC_2_inst/X/Receiver;Data_Sink_port
receiving event Data_Sink_port
Receive NEW VALUE 138
Receive NEW VALUE 138
scheduler : BIP_Top/tick1/CPU_SC_2_inst;tick1
scheduler : BIP_Top/tick1/CPU_SC_2_inst;tick1

```

### Spacecraft\_2

# Outline

- Motivation
- Overview of AADL
- Overview of BIP
- Prototyping of Distributed Implementation
- Case study
- **Conclusion**



# Conclusion

- We provide a prototyping method from AADL to distributed implementation via BIP, which has an operational semantics formally defined in terms of labelled transition systems.
- Translation allows distributed execution of AADL models
- Using BIP, allows application of verification techniques, such as state exploration (using IF toolset) or component-based deadlock detection (D-Finder tool).

## **Future work :**

- Large Library of encoding/decoding data type to automate the generation.
- Support different delay characteristics for communication between different processes.

Thank you