AADL

Louise Avila
Marcus Chou
Taehyun Kim
11/9/2006

AADL Overview

- Architecture Analysis and Design Language
- Models software and execution platform architectures of performance critical, embedded, real-time systems
- Standard way to describe systems components and interfaces

AADL Standardization

- International standard by Society of Automotive Engineers (SAE)
  - Textual and graphical language
  - XML/XMI data exchange format
  - Semantics of AADL for UML
  - Support for fault/reliability modeling and hazard analysis
- Standard published in November 2004
- Derived from MetaH

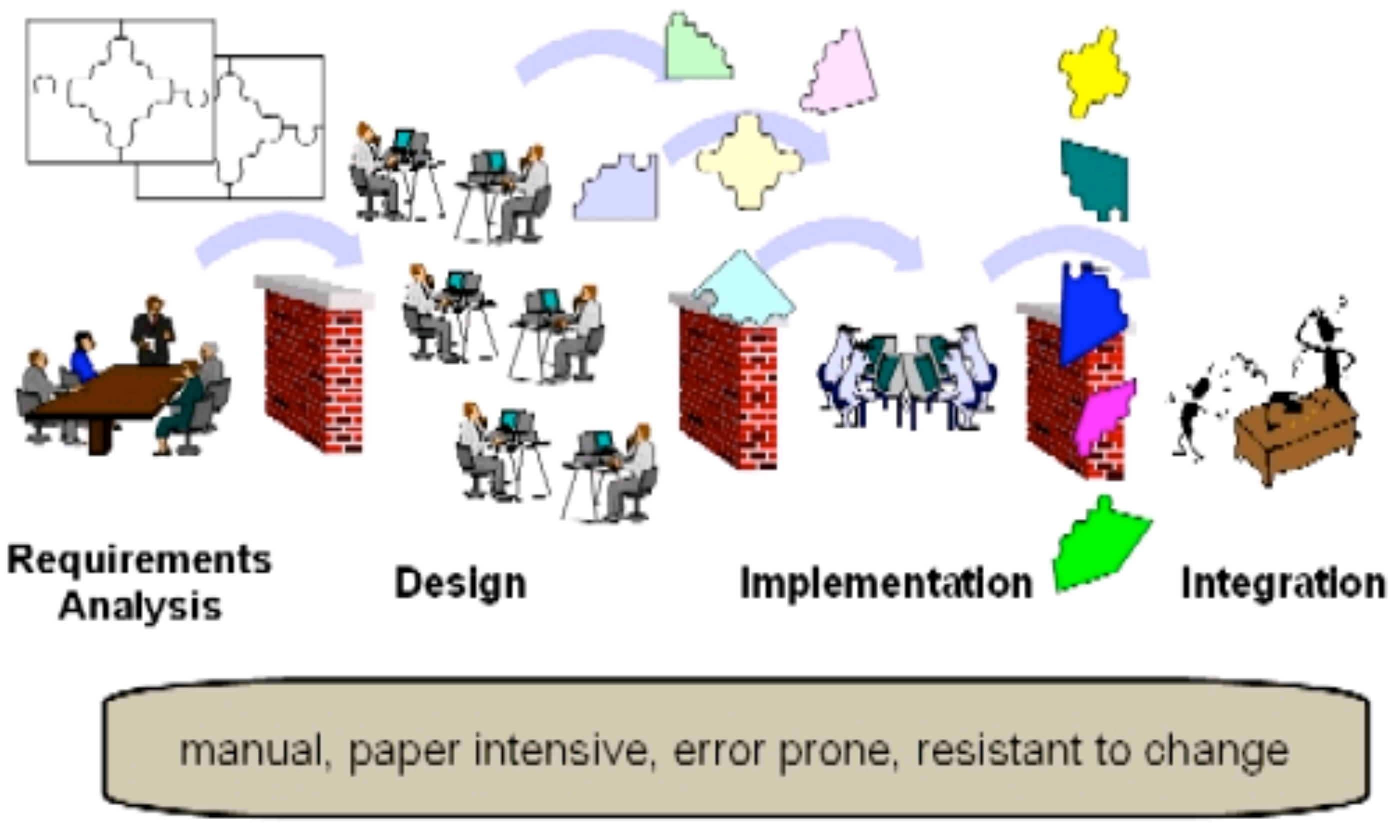
Uses in Industry

- Honeywell
- Airbus
- Axlog
- European Space Agency
- General Dynamics

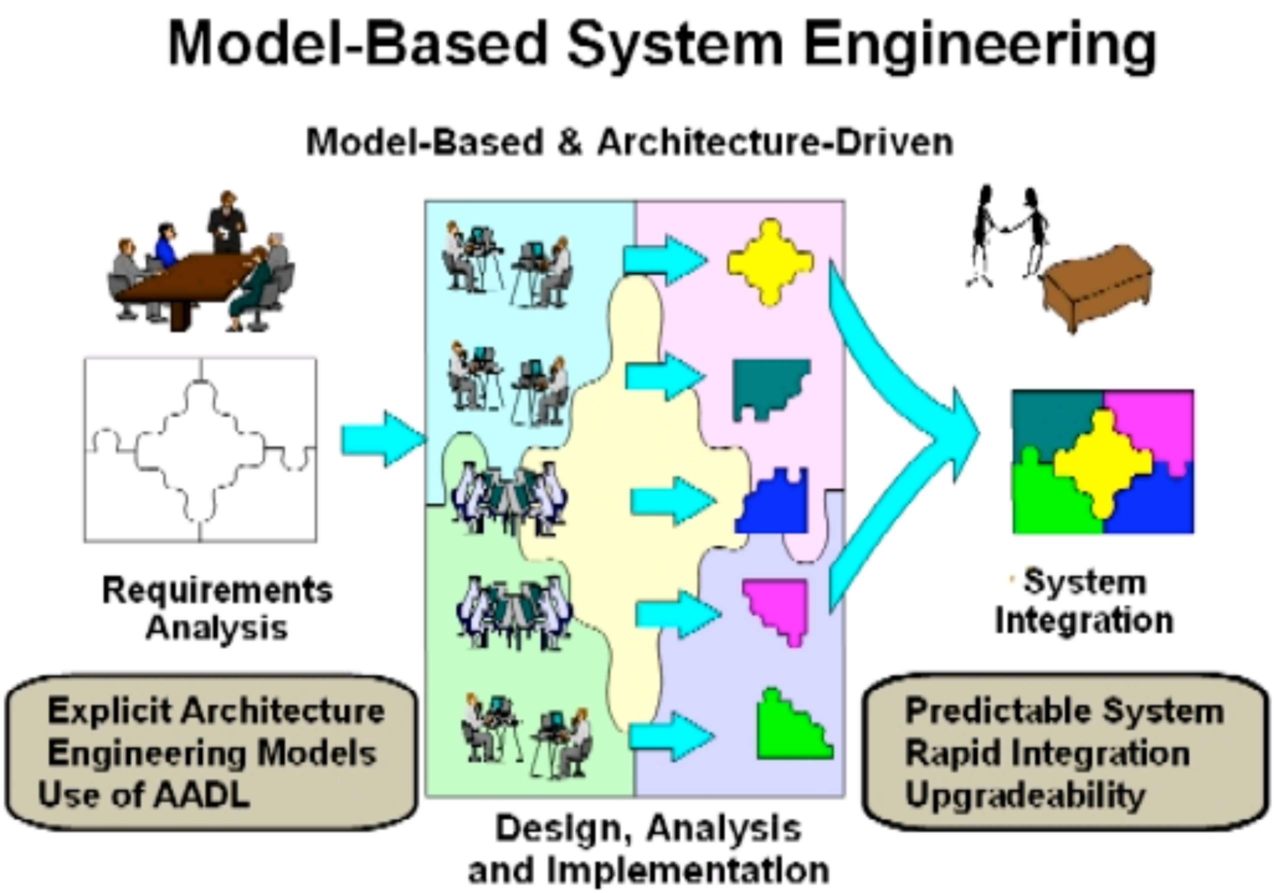
Motivation for AADL

- Current practices often error prone, manual, paper intensive, and resistant to change
- System architecture hard to capture for specification, design, and validation
- Lack of insight into critical system characteristics
  - Performance
  - Safety
  - Time criticality
  - Security
  - Fault Tolerance

Current Practices are Inefficient and Not Robust Enough



Model-Based System Engineering



Benefits of Model-Based Engineering

- Precise syntax and semantics for performance critical systems
- Large scale model can be incrementally refined
- Early lifecycle tracking of modeling and analysis
- Analyze runtime computer system simulation rather than just functional behavior

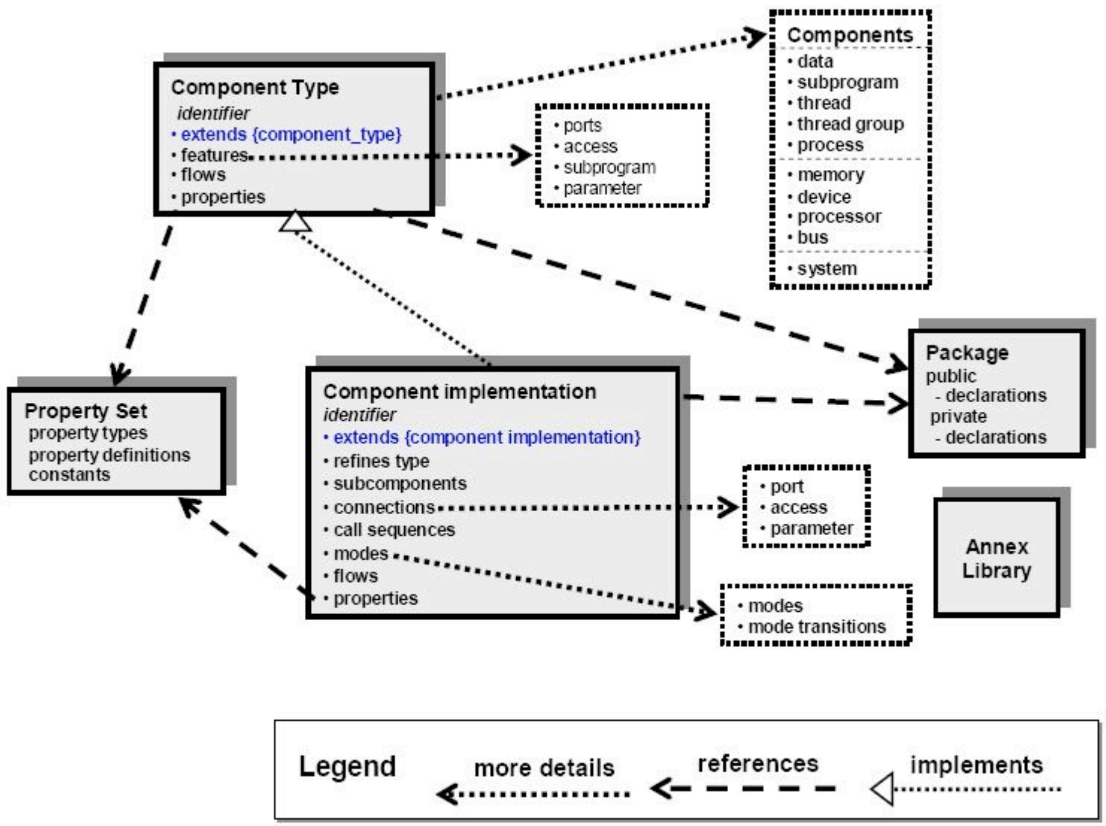
Additional Benefits of AADL

- Exchange engineering data between multiple organizations
- Framework for modeling and analysis
- Facilitate automation of code generation
- Reduce design and implementation defects
- System model precisely capture the architecture

AADL Language Abstractions

- Component
  - Component Types
  - Component Implementations
- Packages
- Property sets and Annex Libraries

AADL Elements



- ttp://www.sei.cmu.edu/pub/documents/06.reports/pdf/06tn011.pdf

Components

- Defines interactions with other components and internal structure
- Assigned a unique identity
- Defined through type or implementation
- 3 distinct component categories
  - Application Software
  - Execution Platform
  - Composite

AADL Component Types

- Model physical system components
- Specification of a component
- Software Types
  - Model source text, initialize address space, units of concurrent execution
- Execution Platform Types
  - Support execution of threads, storage of data and code, communication between threads

Component Type Example

process simple_speed_control
  features
  raw_speed: **in data port** speed_type;
  toggle_mode: **in event port**;
  throttle_cmd: **out data port** throttle_data;
  flows none;
end simple_speed_control;

Component Implementation

- Specify internal structure of component
- Implementation composition
  - Subcomponents
  - Interaction among features of subcomponents
  - Flows across sequences of subcomponents
  - Modes that represent operation states
  - Properties

Component Implementation Example

```
thread control_laws
end control_laws;
data static_data
end static_data;
thread implementation control_laws.control_input
    subcomponents
    configuration_data: data static_data;
    calls none;
end control_laws.control_input;
```

Packages, Property Sets, and Annexes

- Packages declare a namespace for components
- Property sets
  - Named grouping of property declarations
  - Declares new properties and property types
- Annex
  - Enables user to extend AADL
  - Incorporate specialized notation within AADL model

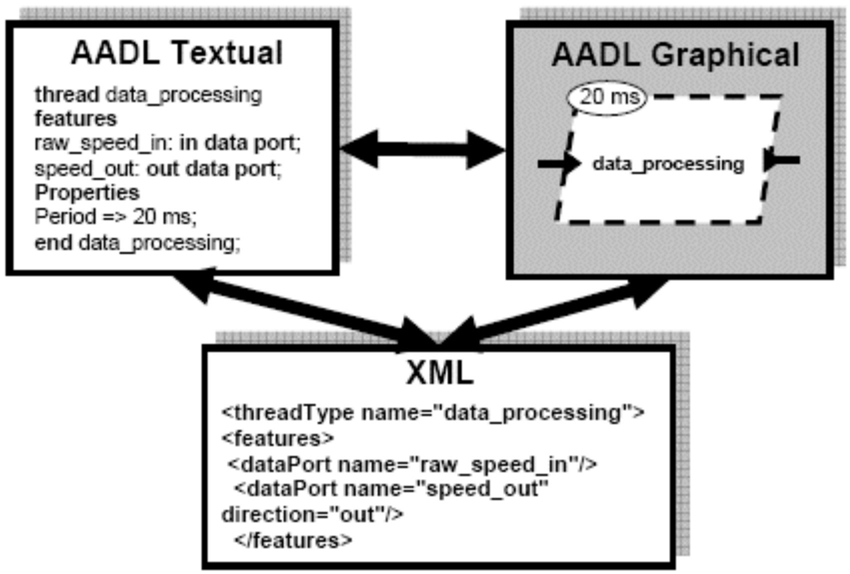
Package Example

```
package actuators_sensors
    public
    device speed_sensor
    end speed_sensor;
    -- …
end actuators_sensors;
system control
end control;
system implementation control.primary
    subcomponents
    speed_sensor: device actuators_sensors::speed_sensor;
    -- …
end control.primary;
system implementation control.backup
subcomponents
speed_sensor: device actuators_sensors::speed_sensor;
```

Property Set Example

```
system implementation data_processing.accelerometer_data
properties
    set_of_faults::comm_error_status => true;
end data_processing.accelerometer_data;
property set set_of_faults is
    -- An example property name declaration
    comm_error_status: aadlboolean applies to (system, device);
    -- An example property type declaration
    Speed_Range : type range of aadlreal 0.0 mph..150.0 mph units (mph);
    -- An example property constant declaration
    Maximum_Faults : constant aadlinteger => 3;
end set_of_faults;
```

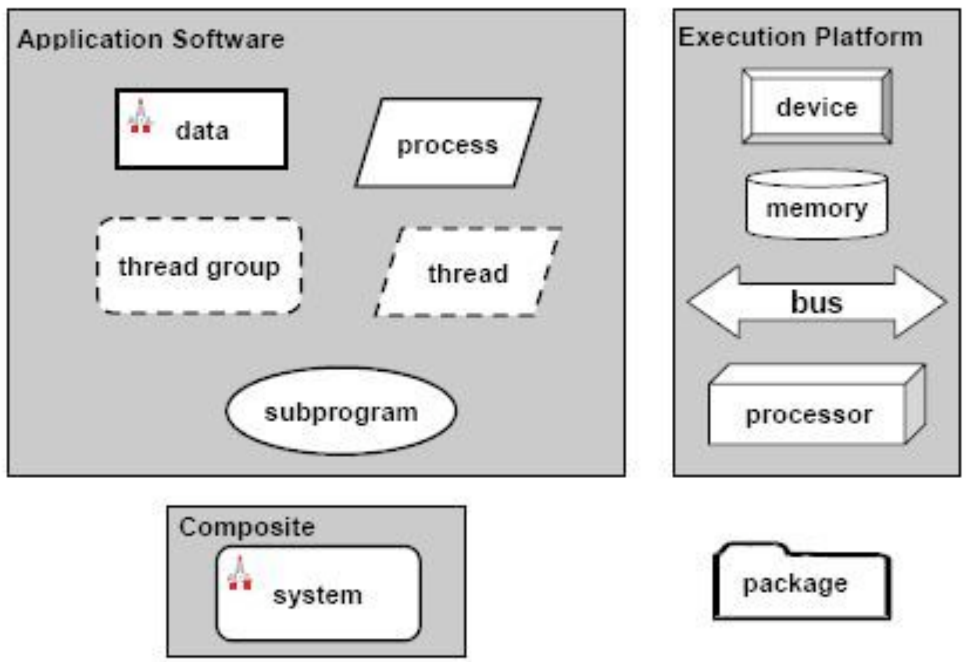
AADL Representations



Textual Specification

- Component Type: system, process, thread, thread group data, subprogram, processor, device, memory, and bus
- Component Implementation: system, process, thread, thread group data, subprogram, processor, device, memory, and bus

Graphical Representation



Communication Interaction

- Port connections
- Component access connections
- Subprogram calls
- Parameter connections

Features - Definition

- Specify interaction points with other components
- Interface through which control and data exchanged
  - Ports – support directional flow of control and data
  - Subprograms – synchronous procedure calls
  - Requires access
    - Use to access external components
  - Provides access
    - Make subcomponent accessible to external components

Ports

- Data port: Interfaces for typed state data transmission among components without queuing
- Event port: Interfaces for the communication of **events** raised by **subprograms, threads**, **processors**, or **devices** that may be queued
- Event data port: Interfaces for message transmission with queuing

Port Declarations

- Declared as features in the component type declaration
- Ports are direction (in/out)
- Pattern for port connection
  - name : [**descriptor**] [**source port**] [connection symbol] [**destination port**]
  - Graphically, port connections are solid lines between the ports involved in the connection

AADL Model

- Describes properties and interfaces of components
- Software components
  - Application software modules
- Execution platform components
  - Processors
  - Bus
  - Memory

AADL Model

- Describes how components interact and are integrated to form complete systems
- Functional interfaces
- Performance critical aspects
- Implementation details specified by software programming and hardware description languages

Software Components

- Abstractions to represent process source text and execution paths through executable code
  - Data
  - Subprograms
  - Threads
  - Thread Groups
  - Processes

Data: Definition

- Data component represents a data type in source text
- Data subcomponents
  - Represent internal structure
  - Example: fields in a record or structure or instance variables in a class
- *Features* model concept of operations performed on a data type
- Components can have shared access to data

Data: Example

- Data implementation with 4 data subcomponents

```
data address
end address;

data implementation
address.others
  subcomponents
  street : data string;
  streetnumber: data int;
  city: data string;
  zipcode: data int;
end address.others;
```

```
//Supporting data declarations
data string
end string;

data int
  properties
   Source_Data_Size => 64b;
end int;
```

Data Example

- Data type weather_DB has associated access functions getCurrent and getFuture
- Represented by subprogram declarations in features subclause

```
data weather_DB
features
    getCurrent: subprogram getCurrent;
    getFuture: subprogram getFuture;
end weather_DB;
```

Subprogram: Definition

- Callable source text that is executed sequentially
  - Function, method
- Operates on data or provides server functions to components that call it
  - With or without parameters
  - In and in out parameters
  - Out and out in parameters

Subprogram: Definition

- Type declaration specified interactions with other parts of source text
  - Required access to shared data
- Thread and subprogram implementations can contain subprogram calls

Subprogram: Example

```
data Matrix
end Matrix;

subprogram getCurrent
features
  result: out parameter Matrix;
end getCurrent;

subprogram getFuture
features
  date: in parameter date;
  result: out parameter Matrix;
  bad_data: out event port; //handle an exception
  wdb: requires data access weather_DB;
end getFuture;
```

Thread Definition

- Represent sequence of instructions in a executable produced from source text
- Model schedulable units of control
  - Transition between different scheduling states
  - Can execute concurrently
- Can interact with each other through:
  - Exchanges of control and data specified in port connections
  - Server subprogram calls
  - Shared data components

Thread: Definition

- Executes in the virtual address space of a process
- Executes a code sequence when dispatched and scheduled to execute
- State transitions
  - Thread halted
  - Initialized
  - Suspended awaiting dispatch
  - Thread deactivation

Thread: Example

- Thread type declaration

```
thread Predict_Weather
  features
      target_date: in event data port date;
      prediction: out event data port weather_forecast;
      past_date: out event port;
      weather_database: requires data access weather_DB;
end Predict_Weather;
```

Thread Example

- Thread implementation

```
Thread implemementation Predict_Weather.others
Calls {
    current: subprogram weather_DB.getCurrent;
    future: subprogram weather_DB.getFuture;
    diff: subprogram Matrix_delta;
    interpret: subprogram Interpret_result;
};
connections
    parameter target_date -> future.date;
    event port future.bad_date -> past_date;
    parameter current.result -> diff.A;
    parameter future.result -> diff.B;
    parameter interpret.result -> prediction;
    data access weather_database -> future.wdb;
end Predict_Weather.others;
```

Thread Properties

- Used to specify critical runtime aspects of a thread within the architectural representation
- Enables early analyses of thread behavior
- Properties
  - Timing (WCET)
  - Dispatch protocols (periodic, aperiodic)
  - Memory size
  - Processor binding

Thread Properties: Example

```
thread control
properties
-- nominal execution properties
  Compute_Entrypoint => "control_ep";
  Compute_Execution_Time => 5 ms .. 10 ms;
  Compute_Deadline => 20 ms;
  Dispatch_Protocol => Periodic;
-- initialization execution properties
  Initialize_Entrypoint => "init_control";
  Initialize_Execution_Time => 2 ms .. 5 ms;
  Initialize_Deadline => 10 ms;
end control;
```

Thread and Events

- Every thread has default in event port named Dispatch
  - If connected (i.e. named as destination in a connection declaration), arrival of event results in dispatch of thread
  - Ignored by periodic threads (dispatches are determined by the clock)
- Every thread has default out event port named Complete
  - If connected, event raised on port when execution of thread dispatch completes

Thread Group: Definition

- Organizational component to logically group threads contained in processes
- Type specifies features and required subcomponent access
- Implementation represents contained threads and their connectivity
- Single reference to multiple threads and associated data
  - Threads with a common execution rate
  - Threads and data components needed for processing input signals

Thread Group: Example

- Thread group contains a thread, 2 data components and another thread group

```
thread group control
  properties
  Period => 50 ms;
end control;

thread group implementation control.roll_axis
subcomponents
  control_group: thread group control_laws.roll;
  control_data: data data_control.primary;
  error_data: data data_error.log;
  error_detection: thread monitor.impl;
end control.roll_axis;
```

Processes: Definition

- Represents a protected address space
  - A space partitioning where protection is provided from other components accessing anything inside the process
- Contains
  - Executable code and data
  - Executable code and data of subcomponents
  - A Thread to represent an actively executing component

Processes: Example

- Implementation with 3 subcomponents
  - Two ports: input and output

```
process
control_processing
features
input: in data port;
output: out data port;
end
control_processing;
```

```
process implementation
control_processing.speed_control
subcomponents
control_input: thread
control_in.input_processing_01;
control_output: thread
control_out.output_processing_01;
control_thread_group: thread group
control_threads.control_thread_set_01;
set_point_data: data set_point_data_type;
end control_processing.speed_control;
```

Execution Platform Components

- Represent computational and interfacing resources within a system
  - Processor
  - Memory
  - Bus
  - Device
- Software components mapped onto execution platforms
  - Threads bound to processor
  - Processes bound to memory

Processor

- Represents hardware and associated software that execute and schedule threads
- May have embedded software that implements scheduling and other capabilities that support thread execution

Memory

- Represent storage components for data and executable code
  - Subprograms, data and processes are bound to memory components
- Randomly accessible physical storage
  - RAM or ROM
- Complex permanent storage
  - Disks
- Physical runtime properties
  - Word size and word count

Bus

- Represents hardware and associated communication protocols that enable interactions among other execution platform components
  - Connection between 2 threads on separate processors
- Communication specified using access and binding declarations to a bus
- Represent complex inter-network communication by connecting buses to other buses

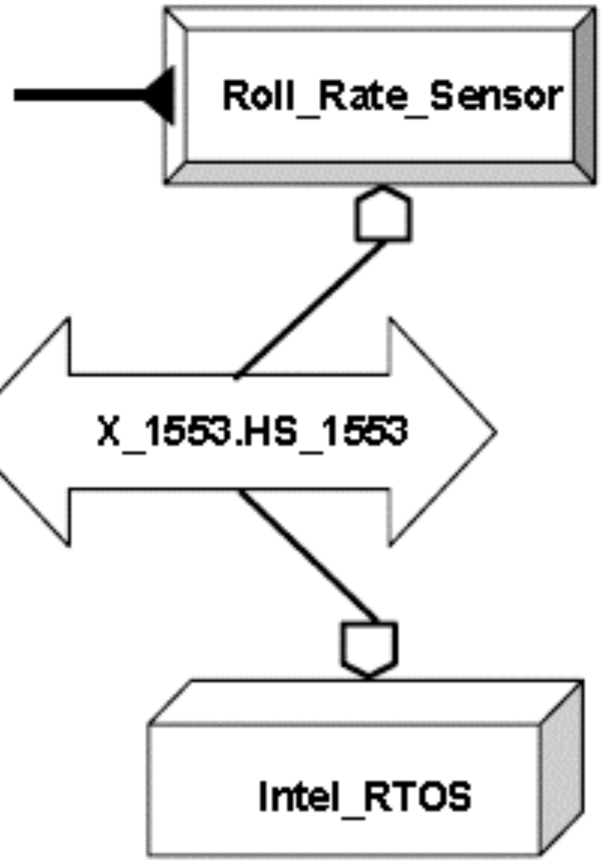
Device

- Represent entities that interface with the external environment of an application system
- Examples
  - Sensors, actuators
  - Standalone systems (GPS)
- Complex behavior

Example

- Device Roll_Rate_Sensor interacts with processor Intel_RTOS through a bus
- Bus access requirement specified in both type declarations
- Out data port on roll rate sensor device provides rate data from the sensor

Example



Example

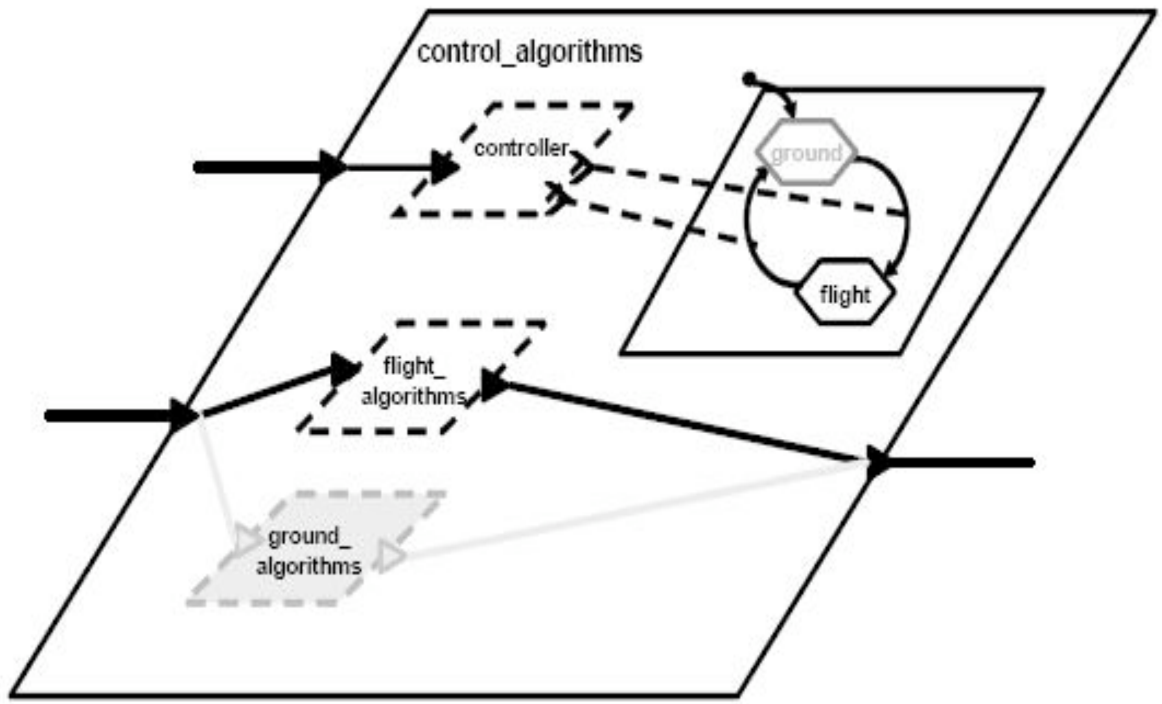
```
processor Intel_RTOS
  features
   A1553: requires bus access X_1553.HS_1553;
end Intel_RTOS;

device Roll_Rate_Sensor
  features
   A1553: requires bus access X_1553.HS_1553;
   raw_roll_rate: out data port;
end Roll_Rate_Sensor;

bus X_1553
end X_1553;

bus implementation X_1553.HS_1553
end X_1553.HS_1553;
```

Modes

- Modes represent alternative operational states of a system or component.

- Modes can establish
  - alternative configuration of active components and connections.
  - variable call sequences within a thread.
  - mode-specific properties for software or hardware components.

Mode Example (Graphical)



Mode Example (Textual)

```
process control_algorithms
features
status_data: in data port;
aircraft_data: in data port;
command: out data port;
end  control_algorithms;
--
process implementation control_algorithms.impl
subcomponents
controller: thread controller;
ground_algorithms: thread ground_algorithms in modes (ground);
flight_algorithms: thread flight_algorithms in modes (flight);
connections
C1: data port aircraft_data -> ground_algorithms.aircraft_data in
modes (ground);
C2: data port aircraft_data -> flight_algorithms.aircraft_data in
modes (flight);
C3: data port ground_algorithms.command_data -> command in modes
(ground);
C4: data port flight_algorithms.command_data -> command in modes
(flight);
modes
ground: initial mode;
flight: mode;
ground -[controller.switch_to_flight]-> flight;
flight -[controller.switch_to_ground]-> ground;
end control_algorithms.impl;
```

```
thread controller
features
status_data: in data port;
switch_to_ground: out event port;
switch_to_flight: out event port;
end controller;
--
thread ground_algorithms
features
aircraft_data: in data port;
command_data: out data port;
end ground_algorithms;
--
thread flight_algorithms
features
aircraft_data: in data port;
command_data: out data port;
end flight_algorithms;
```

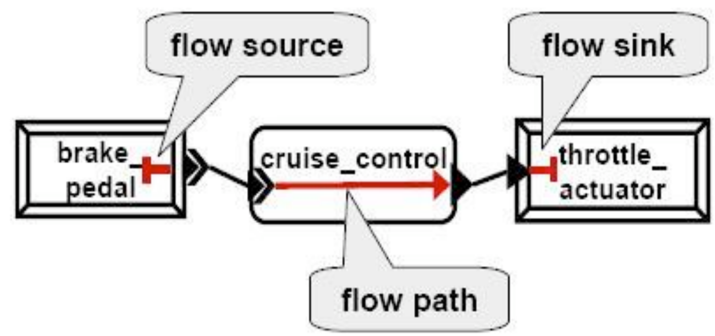
Flows

- Flows enable the detailed description and analysis of an abstract information path through a system.

- Flow declaration
  - source: a feature of a component
  - sink: a feature of a component
  - flow path: flows through a component

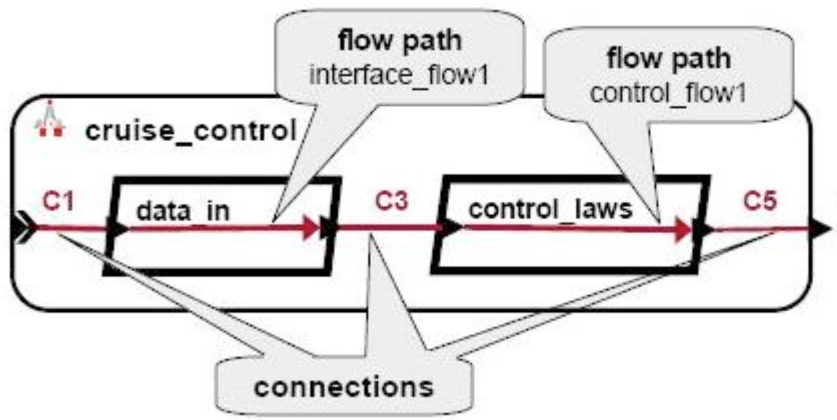
Flow Declaration

```
device brake_pedal
features
        brake_event: out event data port float_type;
flows
        Flow1: flow source brake_event;
end brake_pedal;
--
system cruise_control
features
        brake_event: in event data port;

        throttle_setting: out data port float_type;
flows
        brake_flow: flow path brake_event -> throttle_setting ;
end cruise_control;
--
device throttle_actuator
features
         throttle_setting: in data port float_type;
flows
         Flow1: flow sink throttle_setting;
end throttle_actuator;
```



Flow Paths

```
system implementation cruise_control.impl
subcomponents
data_in: process interface;
control_laws: process control;
connections
C1: event data port brake_event -> data_in.brake_event;
C3: data port data_in.out_port -> control_laws.in_port;
C5: data port control_laws.out_port -> throttle_setting;
flows
brake_flow: flow path brake_event -> C1 -> data_in.interface_flow1 ->
                                    C3 -> control_laws.control_flow1 -> C5 ->
throttle_setting;
end cruise_control.impl;
--
process interface
features
brake_event: in event data port ;
out_port: out data port float_type;
flows
interface_flow1: flow path brake_event -> out_port;
end interface;
--
process control
features
in_port: in data port float_type;
out_port: out data port float_type;
flows
control_flow1: flow path in_port -> out_port;
end control;
```



Properties

- Properties provide descriptive information about components, features, modes, or subprogram calls.
- A property has a name, type, and an associated value.

- Property set

```
property set set name is
 { property type | property name | property constant }+
end set name ;
```

  - property type declaration

    ```
    identifier: type property type definition;
    ```

  - property name declaration

    ```
    name: property type applies to (property owner category);
    ```

  - property constant declaration

    ```
    identifier: constant (type) => property value
    ```

Property Declaration

```
property set my_set is
queue_access: aadlboolean applies to (data);
array_size: set_of_types::array applies to (system,
   process, thread);
maximum_faults: constant addlinteger => 3;
end my_set;
--
property set set_of_types is
length: type aadlreal 7.5 .. 150.0 units( feet );
array: type enumeration (single, double, triplex);
end set_of_types;
```

Property Association

- Property Association assigns a value or list of values to a named property.

```
thread data_processing
features
Sensor_data: in data port {Required_Connection => false;};
end data_processing;
--
thread implementation data_processing.speed_data
properties
   Period => 100 ms;
   Compute_Execution_Time => 2 ms .. 5 ms in binding (Intel);
   Compute_Execution_Time => 3 ms .. 7 ms in binding (AMD);
end data_processing.speed_data;
```

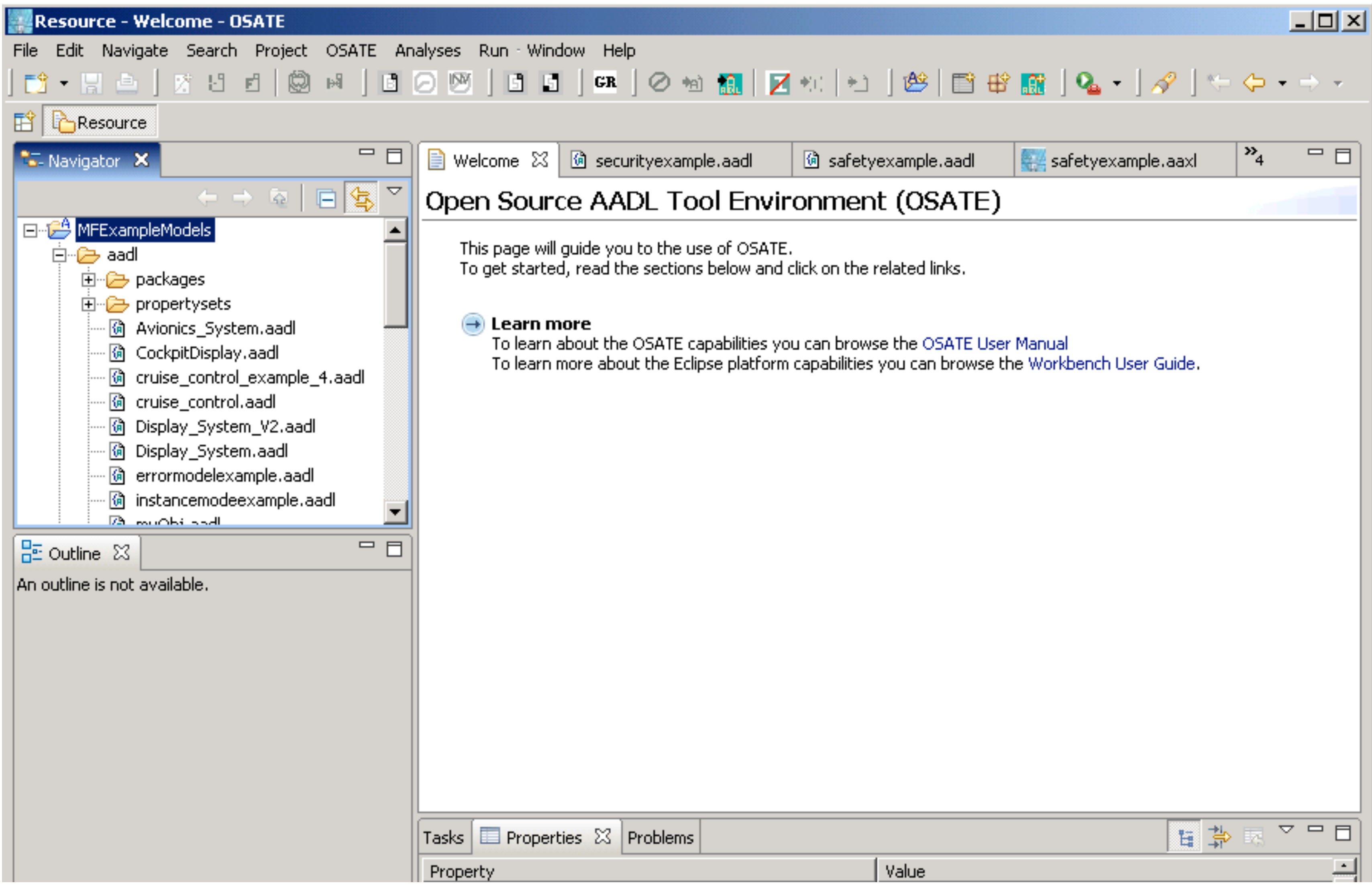
OSATE Introduction

- Open Source AADL Tool Environment
- Developed by Software Engineering Institute
- Set of plug-ins to the open source Eclipse platform
- Supports processing of AADL models
- Available at:
  - www.aadl.info

OSATE Features

- Syntax-sensitive text and AADL object model editor
- Parser and semantic checker for textual AADL
- AADL XML viewer and editor
- Auto-build support
- Analysis tools for performing architecture consistency checks
- A graphical AADL editing by the TOPCASED

OSATE

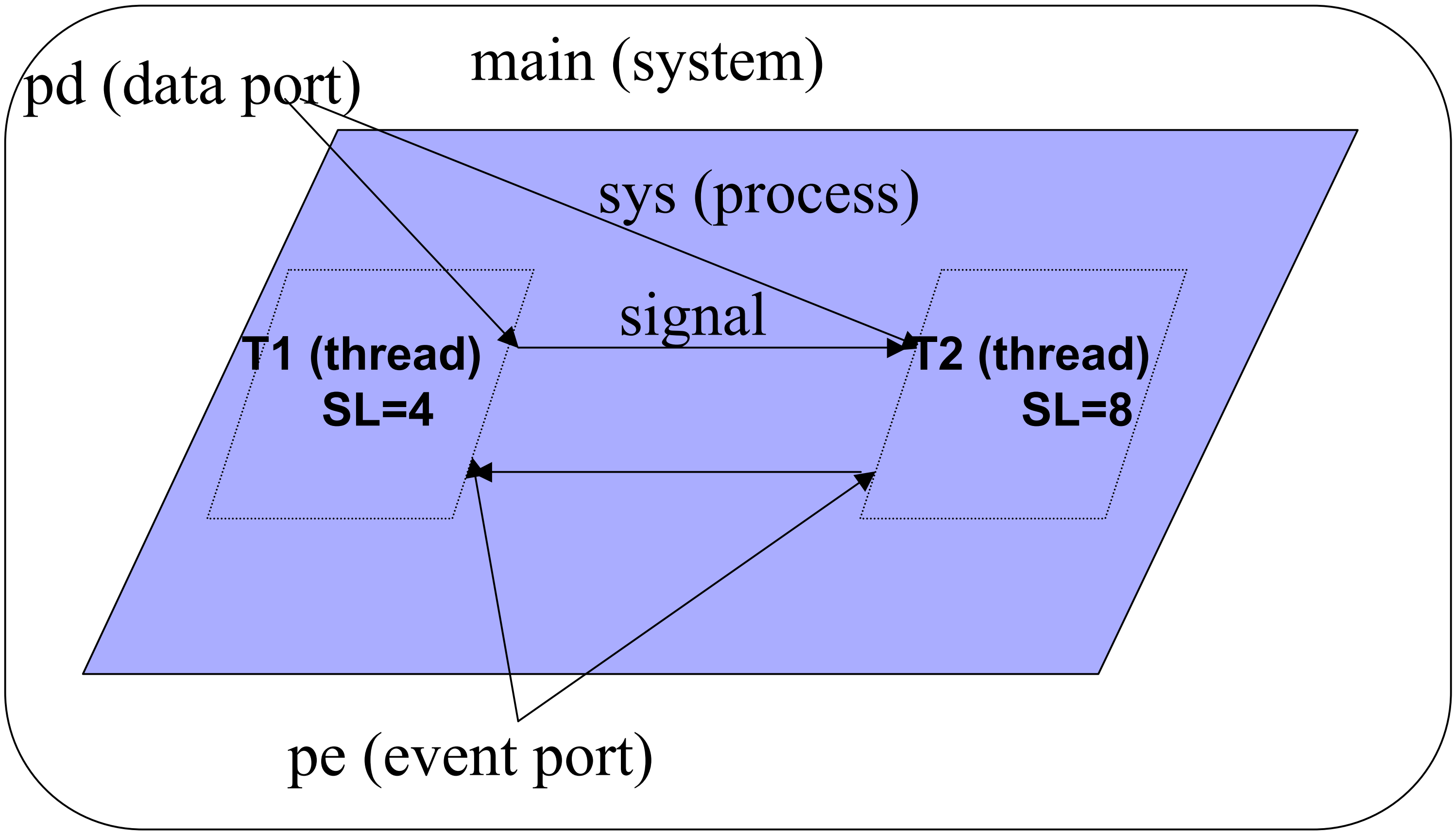


Resource - Welcome - OSATE
File Edit Navigate Search Project OSATE Analyses Run Window Help
Resource
Navigator
MFExampleModels
aadl
packages
propertysets
Avionics_System.aadl
CockpitDisplay.aadl
cruise_control_example_4.aadl
cruise_control.aadl
Display_System_V2.aadl
Display_System.aadl
errormodelexample.aadl
instancemodeexample.aadl
Outline
An outline is not available.
Welcome
securityexample.aadl
safetyexample.aadl
safetyexample.aaxl
Open Source AADL Tool Environment (OSATE)
This page will guide you to the use of OSATE.
To get started, read the sections below and click on the related links.
Learn more
To learn about the OSATE capabilities you can browse the OSATE User Manual
To learn more about the Eclipse platform capabilities you can browse the Workbench User Guide.
Tasks
Properties
Problems
Property
Value

Simple Example 1

- Security Example
  - System
  - Process
  - Threads with Security Level Property
  - Features
    - Externally visible characteristic or component type
    - Used to interact with other components
  - Connections
    - Directional link between features of two components
    - Used to exchange data, events or subprogram calls
  - Data and Event Ports
    - Connection points between components

Security Example



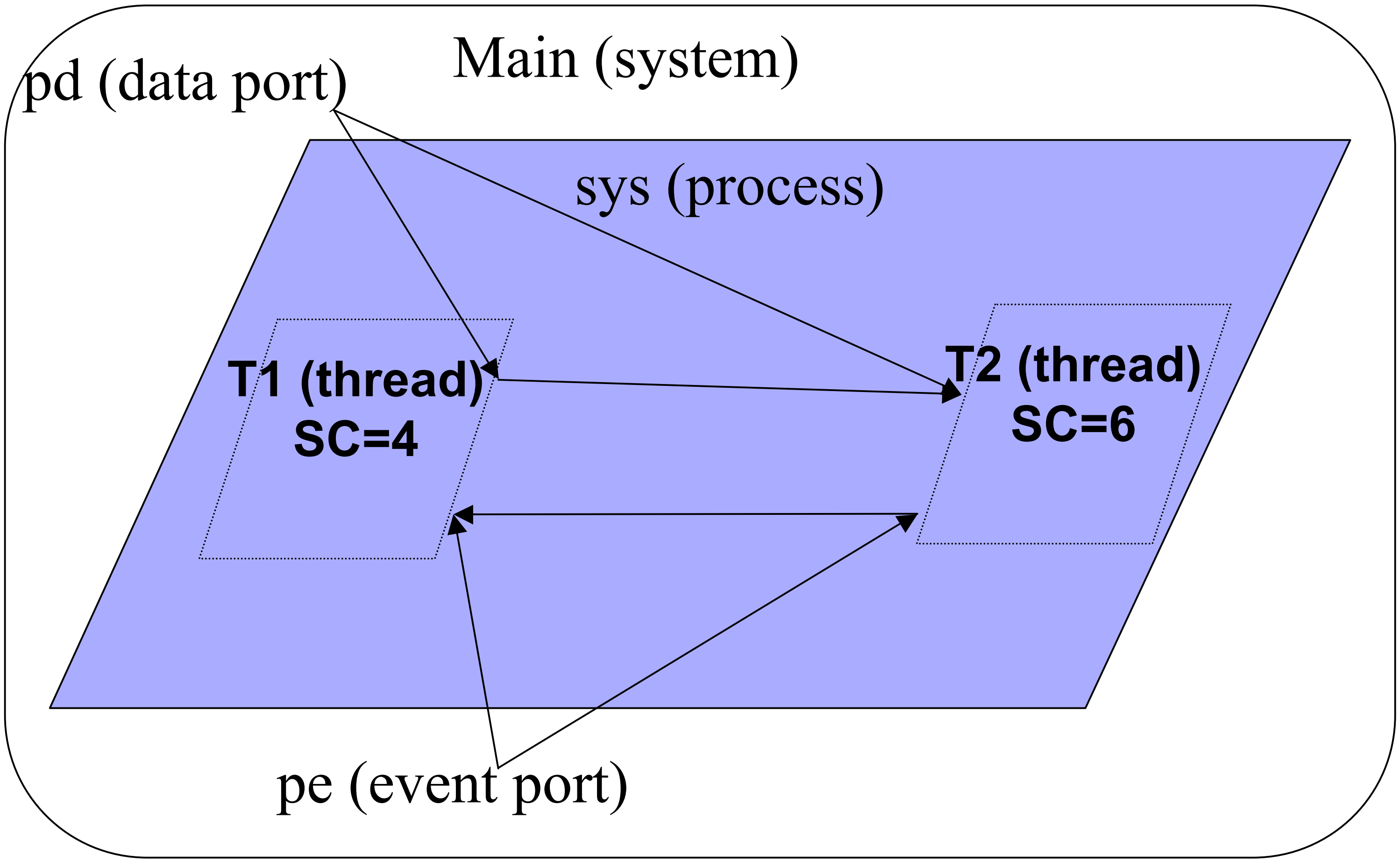
OSATE Analysis

- Security Level Checks
- Compares security level of source and destination components in a connection declaration
- Is the security level of the source component the same or lower than destination?

Simple Example 2

- Safety Example
  - Similar to Security Example
  - Threads with Safety Criticality property

Safety Criticality Example



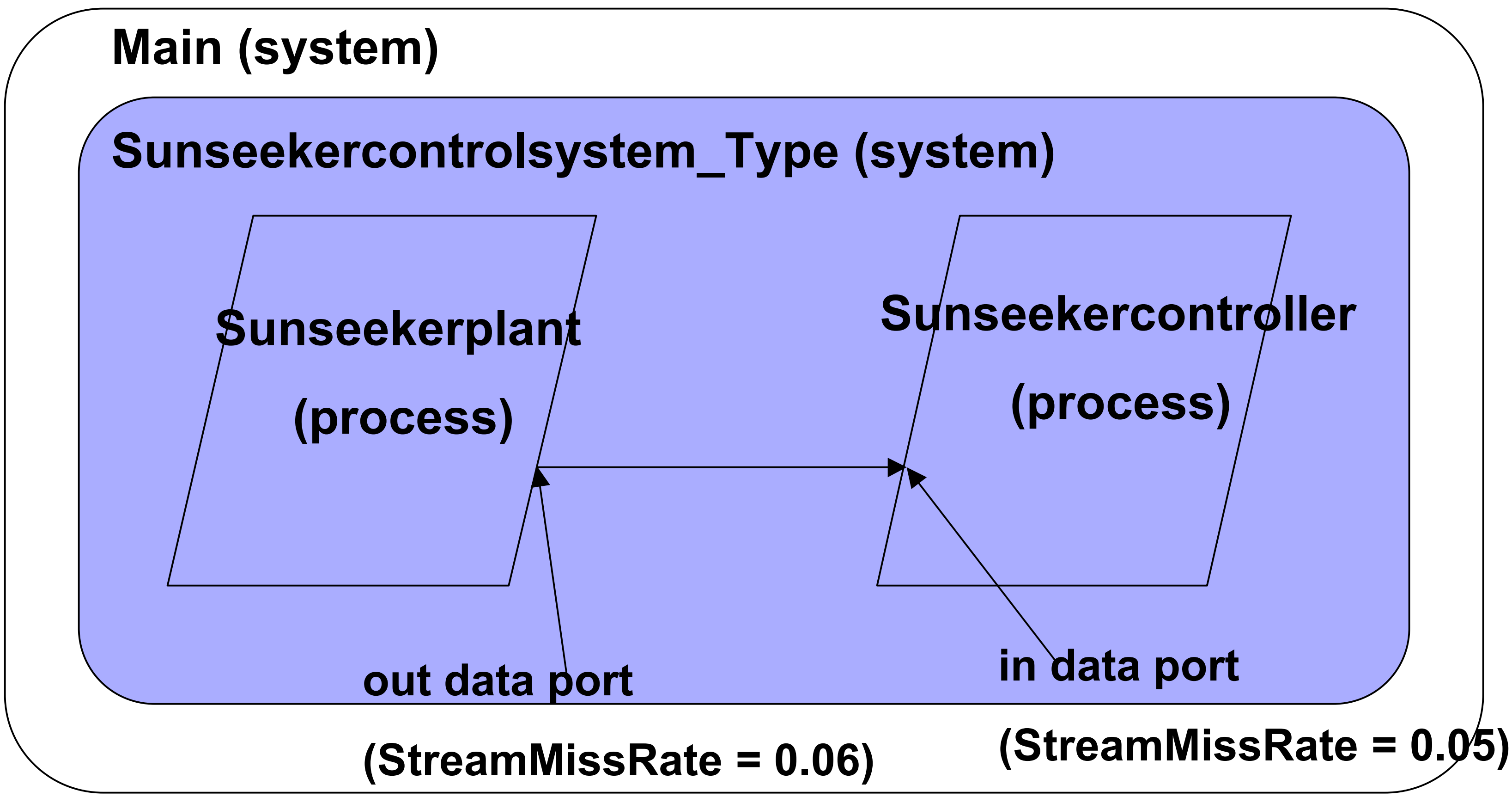
OSATE Analysis

- Safety Level Checks
- Component with lower safety should not drive the operation of a component with a higher safety criticality.
- Is the safety criticality level of the source component higher or equal to the safety criticality level of the destination component?

Simple Example 3

- Sunseekerdemo
- A simple missile guidance example
  - Process
    - Sunseekerplant
      - out data port has StreamMissRate 0.06
    - Sunseekercontroller
      - In data port has StreamMissRate 0.05
  - Connection
    - From out data port of Sunseekerplant to in data port of Sunseekercontroller

Miss Rate Example



OSATE Analysis

- Check Miss Rates
- The outgoing rate specifies the maximum produced miss rate.
- The incoming rate specifies the maximum expected rate that the controller can handle.
- Is the outgoing rate lower than or equal to the incoming rate?

References

- http://www.aadl.info
- http://www.sae.org/technical/standards/AS5506