# CIS 721 - Real-Time Systems

## Lecture 32: SAE AADL: An Architecture Analysis and Design Language for developing embedded real-time systems

Mitch Neilsen

neilsen@ksu.edu

Office: 219D Nichols Hall

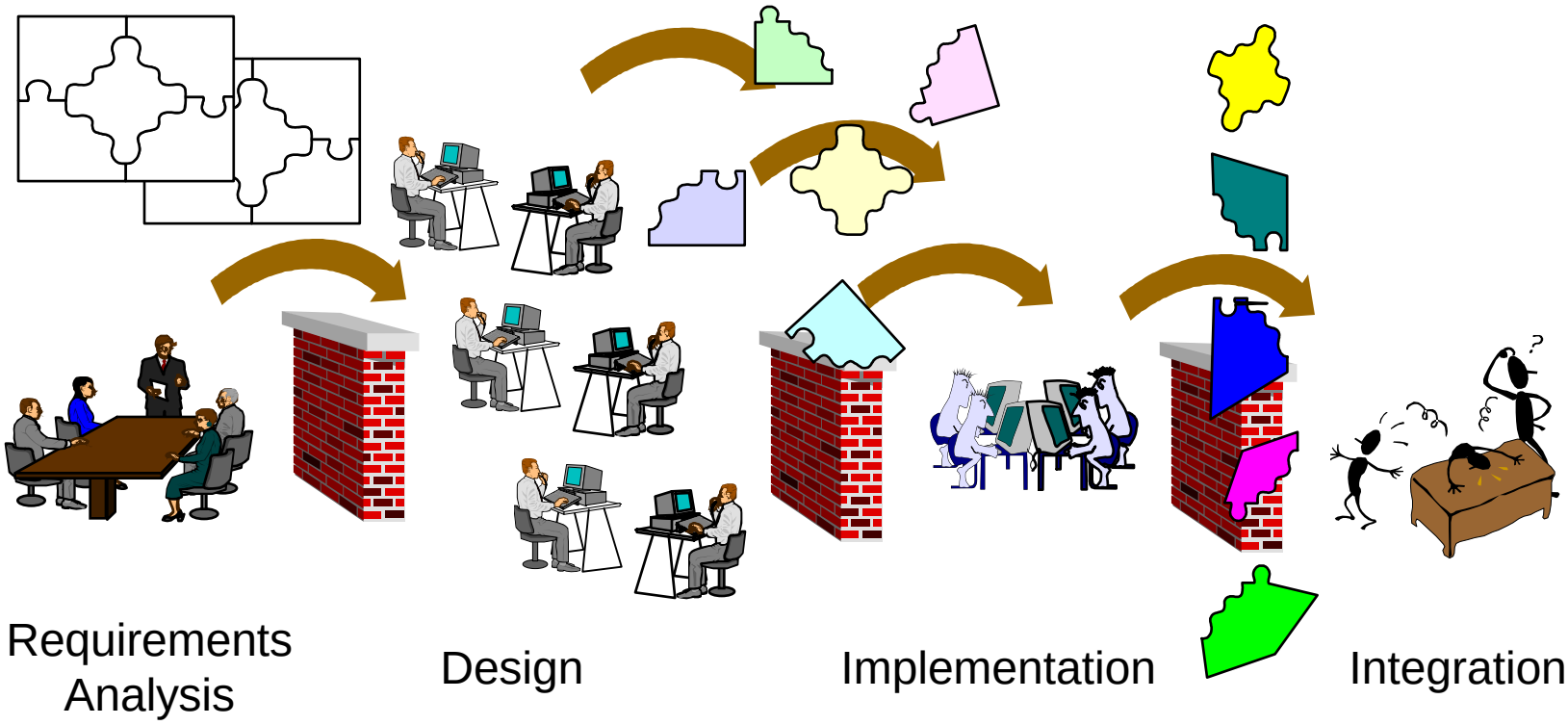
# Outline

- SAE AADLv2 – Society of Automotive Engineers (SAE) Architecture Analysis and Design Language (AADL), version 2
- OSATE2 – Eclipse plugin for AADLv2
- AADL Key Modeling Constructs
  - AADL Components
  - Properties
  - Connections

# Architecture Analysis & Design Language (AADL)

- Specification of computer systems and SoS.
  - Real-time
  - Embedded
  - Fault-tolerant
  - Securely partitioned
  - Dynamically configurable
- Software task and communication architectures
  - Component interface and structure, behavior, properties
- Bound to
  - Distributed multiple processor, integrated hardware architectures
- Fields of application
  - Avionics, Automotive, Aerospace, Autonomous systems, …
- Context and vocabulary for the integration of System Eng Technology
  - Capture of Architecture (& driving requirements), Analysis of Integration Impact (through model checking), Automated Integration to specification.

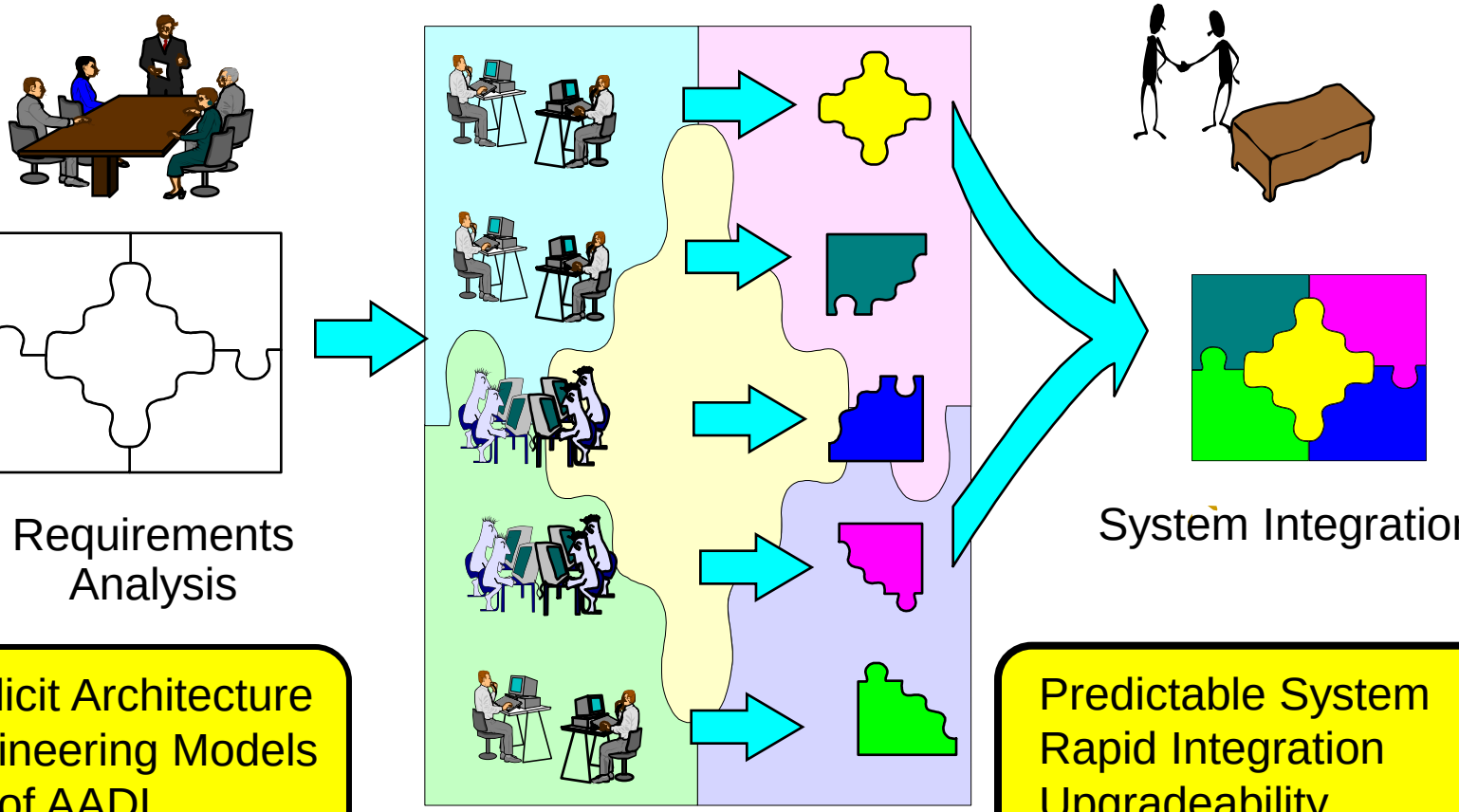
# Typical Software Development Process




manual, paper intensive, error prone, resistant to change

# Model-Based System Engineering

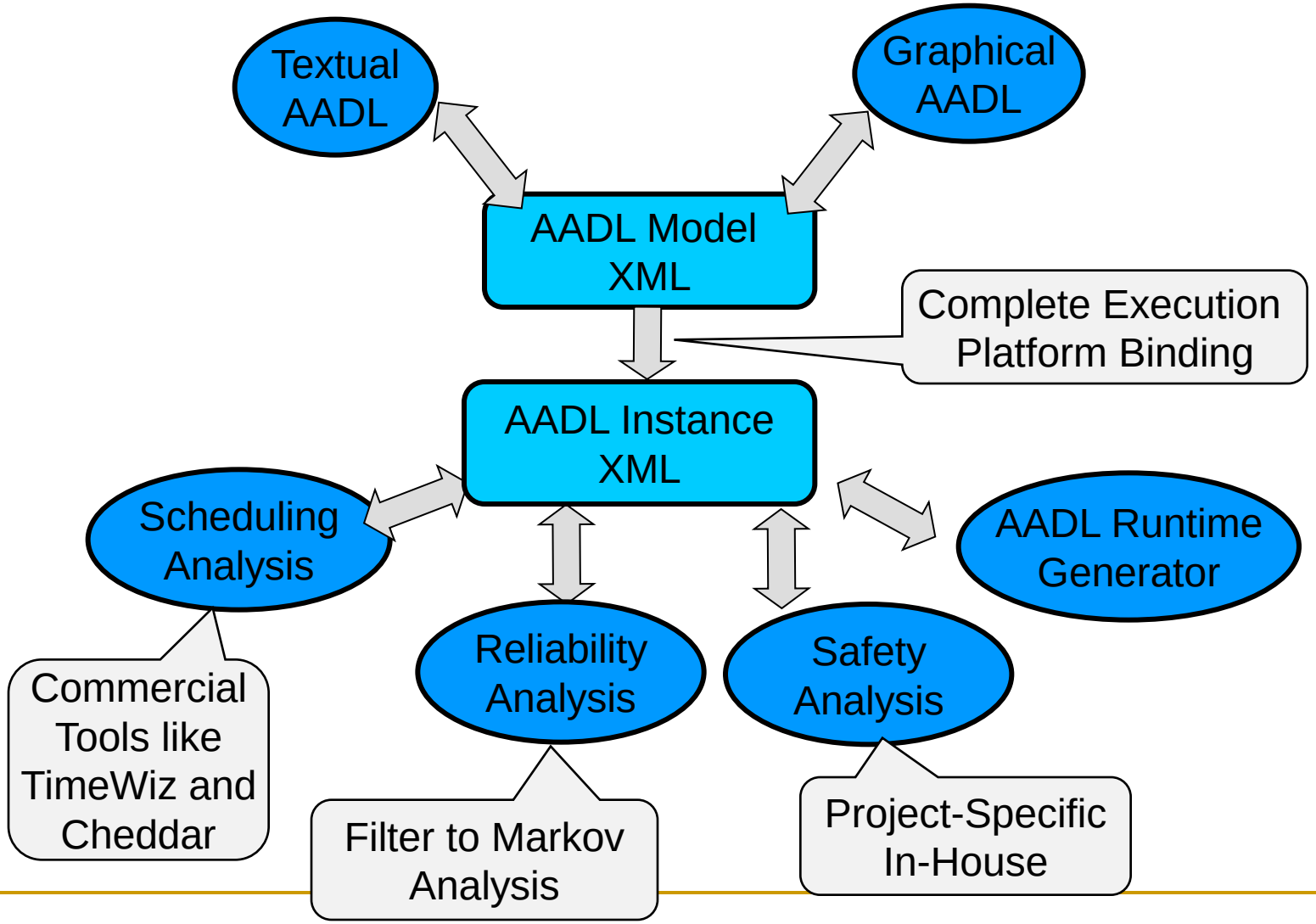
Model-Based & Architecture-Driven



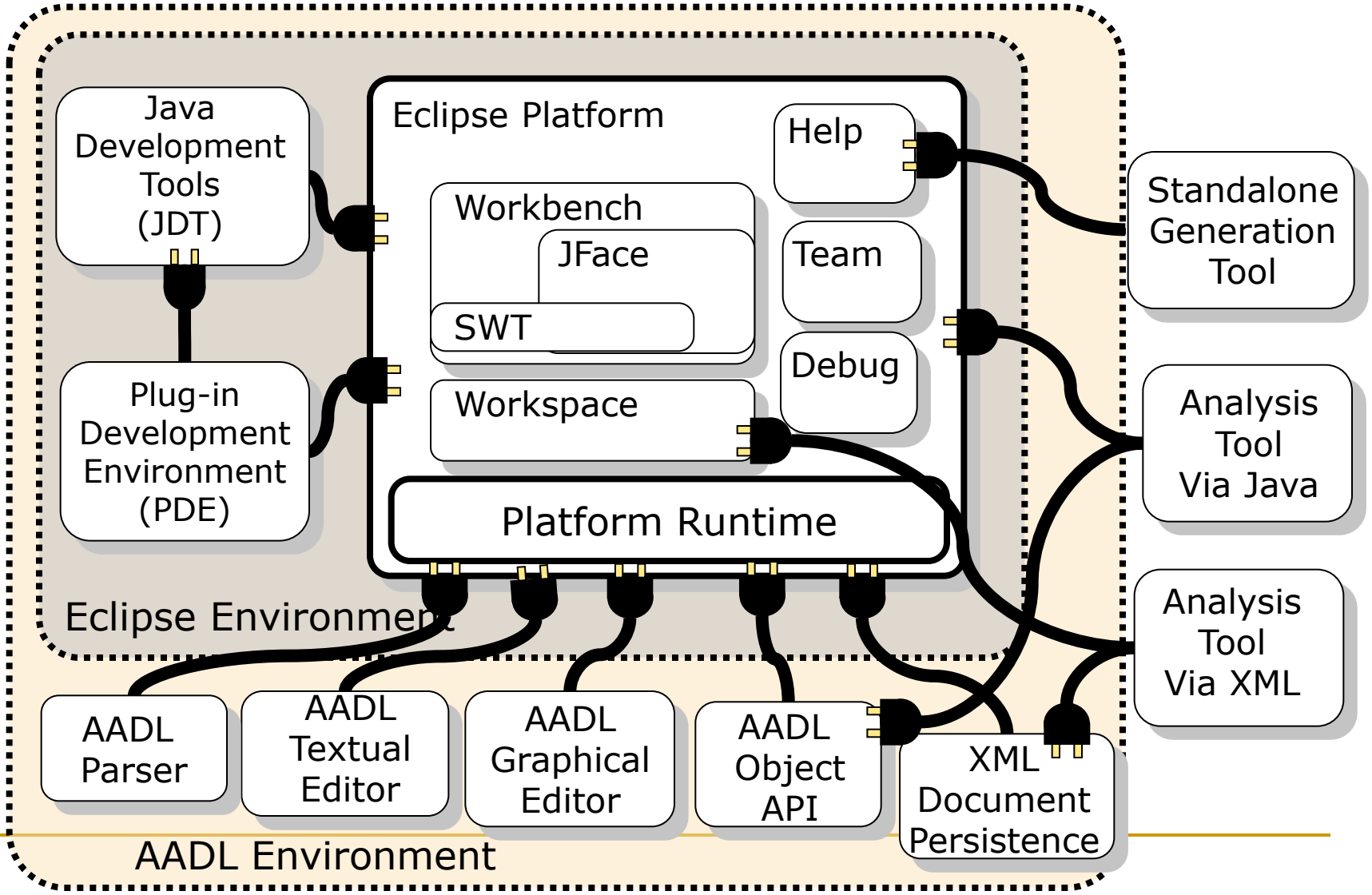

Explicit Architecture Engineering Models
Use of AADL

Predictable System
Rapid Integration
Upgradeability

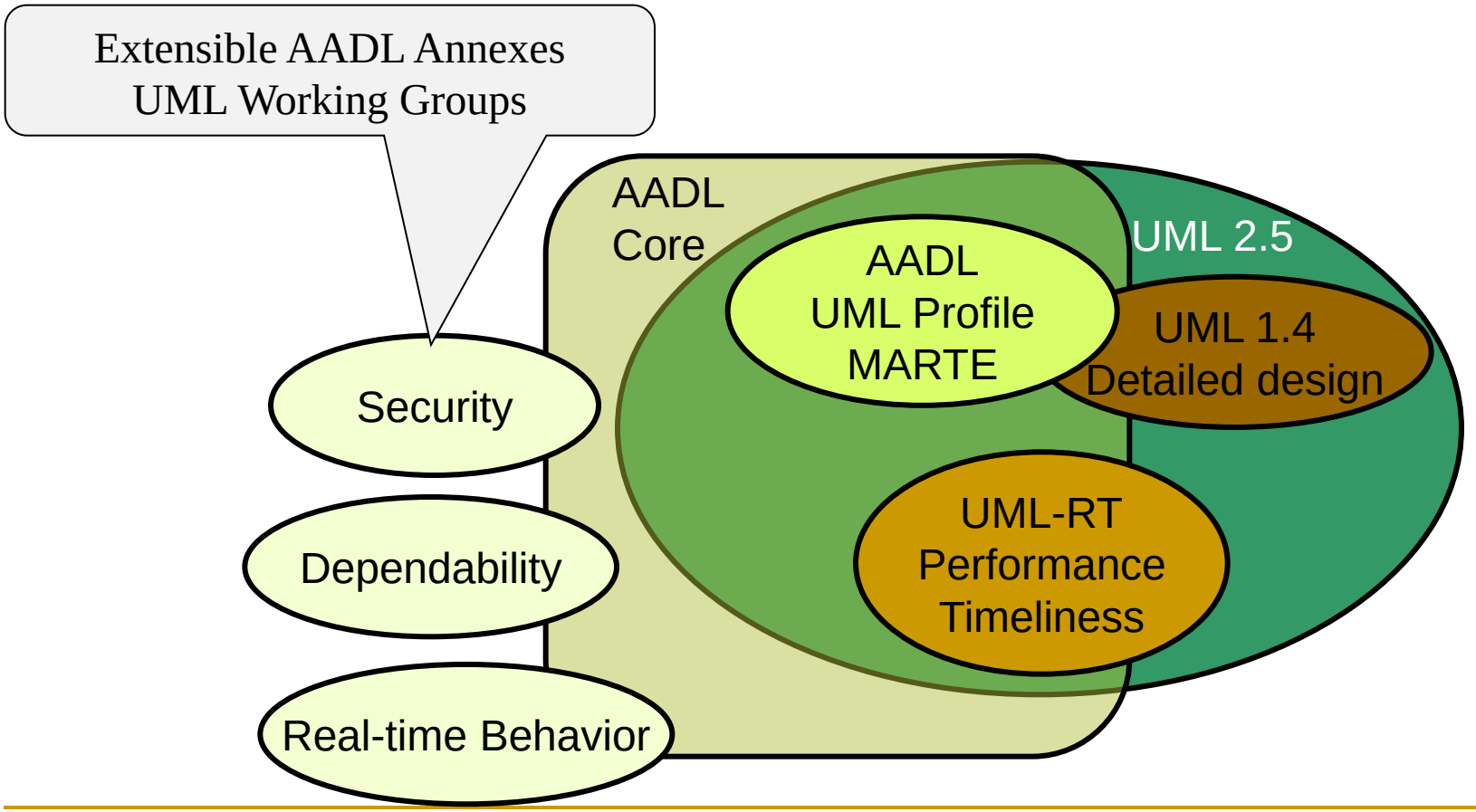
# An XML-Based AADL Tool Strategy



# OCASE: Open Source AADL Environment



# AADL/UML Relationship

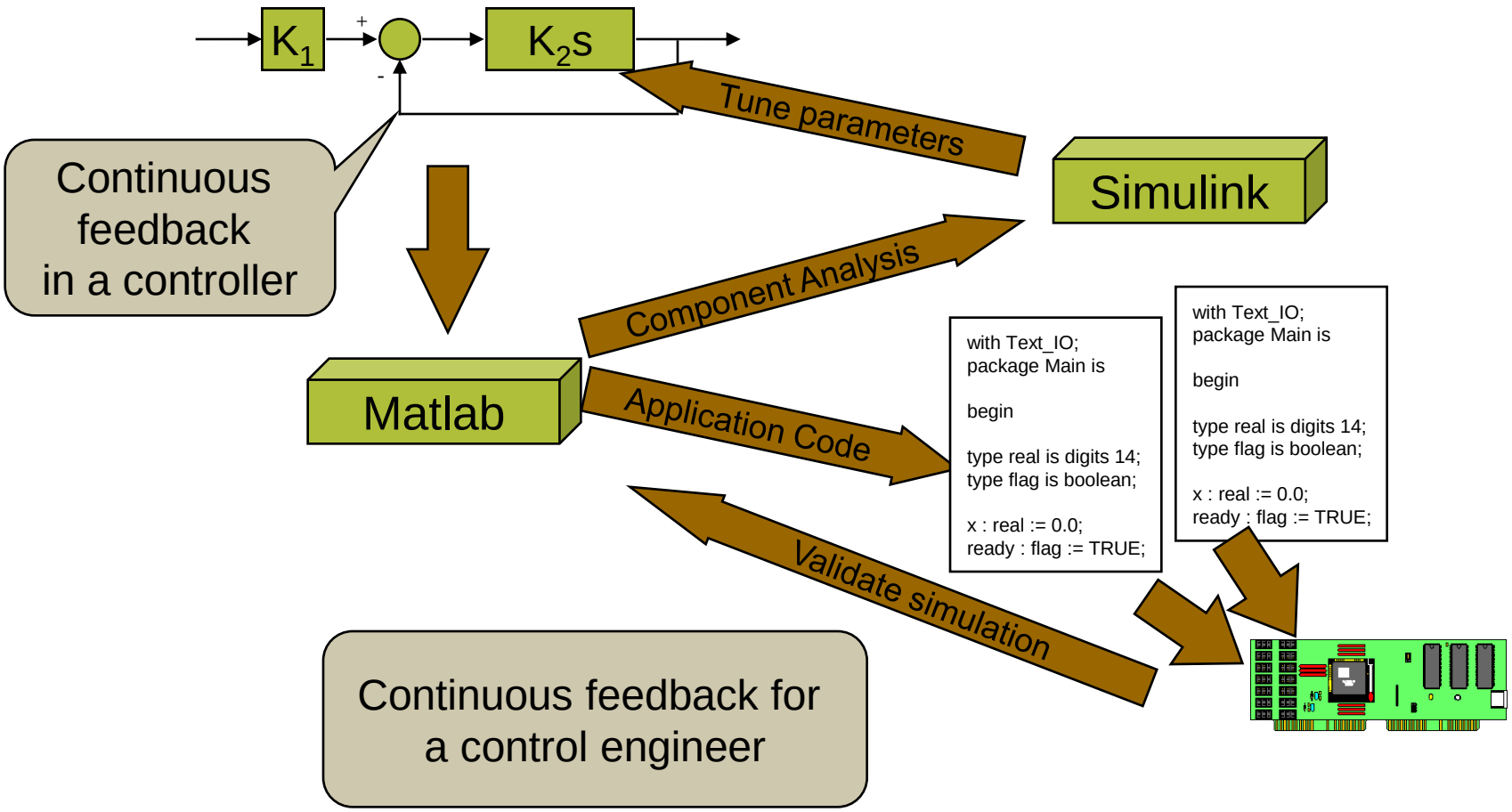


# What Is Involved In Using The AADL?

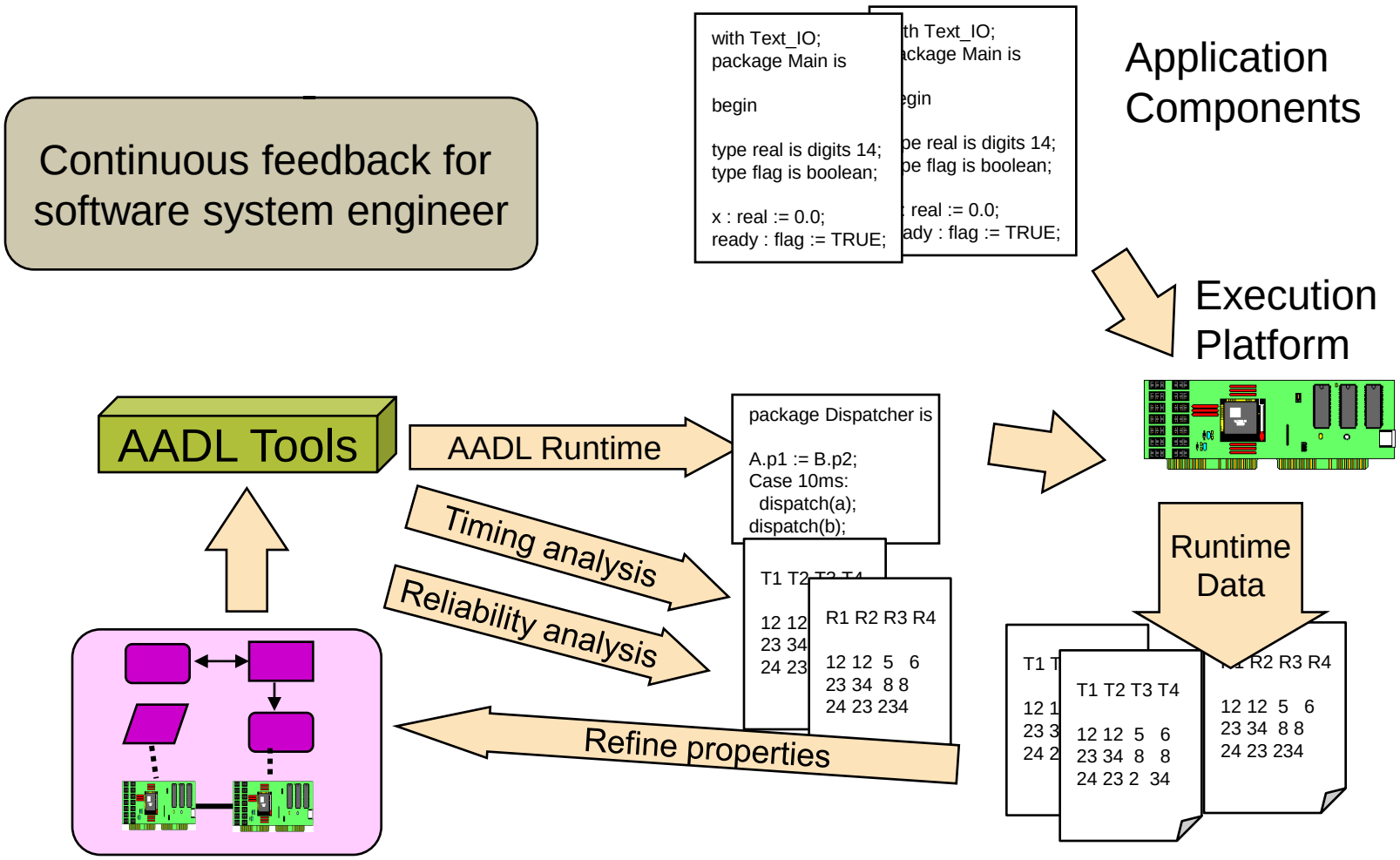
- Specify software & hardware system architectures
- Specify component interfaces and implementation properties
- Analyze system timing, reliability, partition isolation
- Tool-supported software and system integration
- Verify source code compliance & middleware behavior

Model and analyze early and throughout the product life cycle

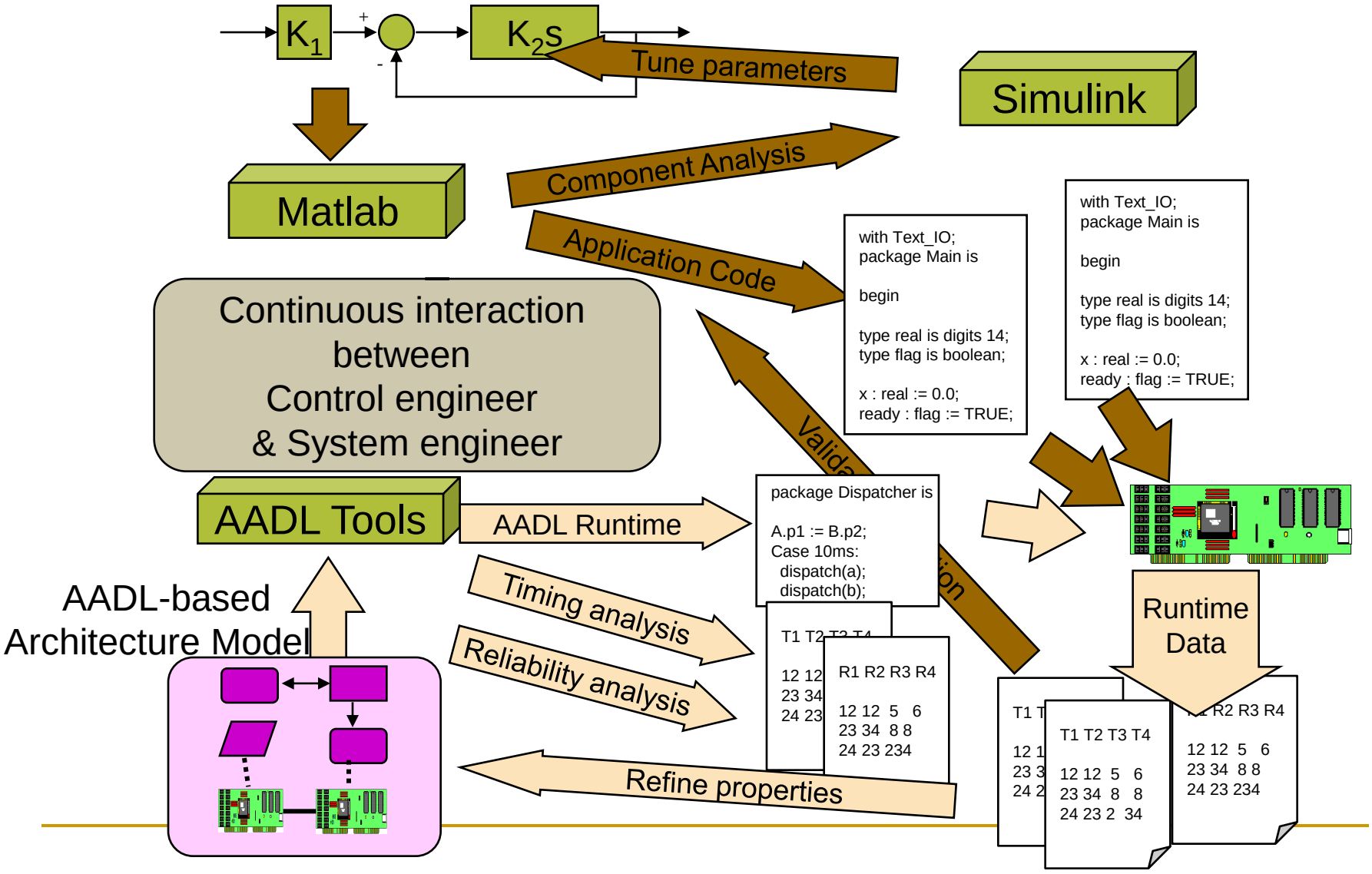
# A Control Engineer Perspective



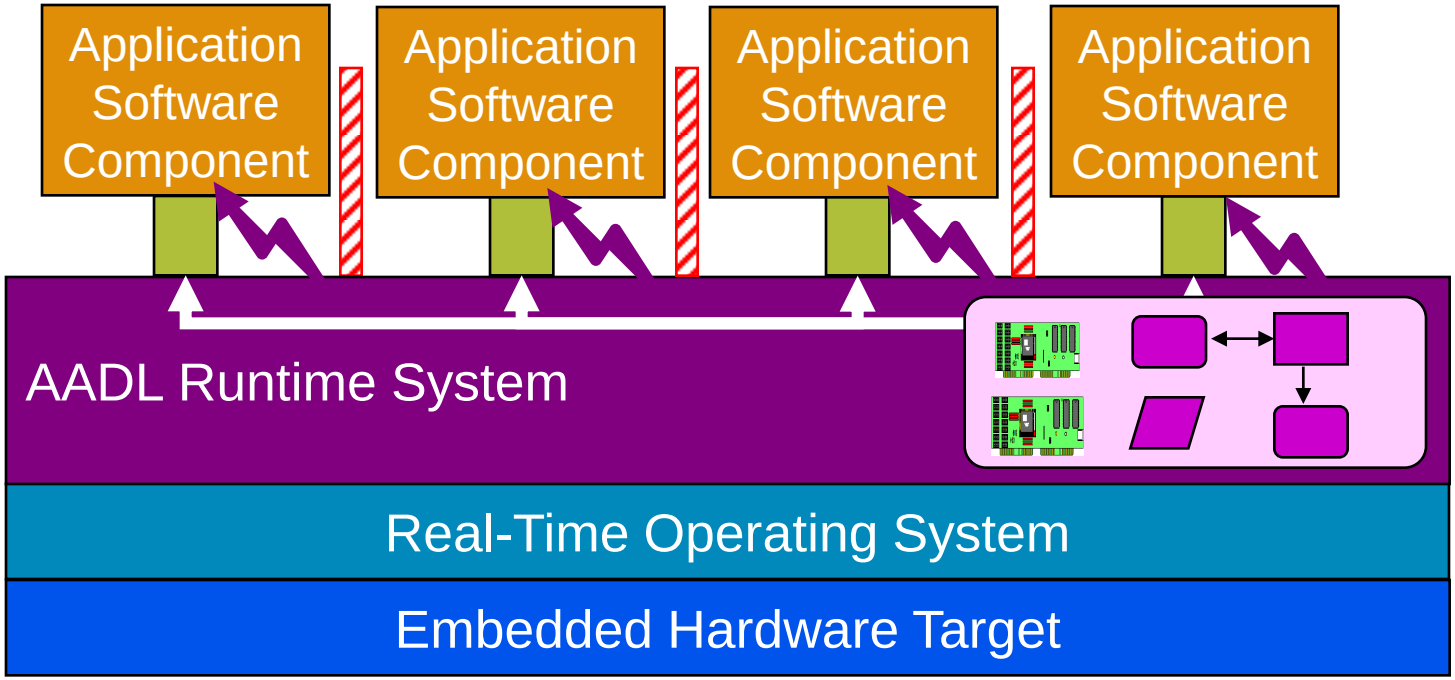
# A Software System Engineer Perspective



# A Combined Perspective



# Application Components as Plug-ins




## Strong Partitioning

- Timing Protection
- OS Call Restrictions
- Memory Protection

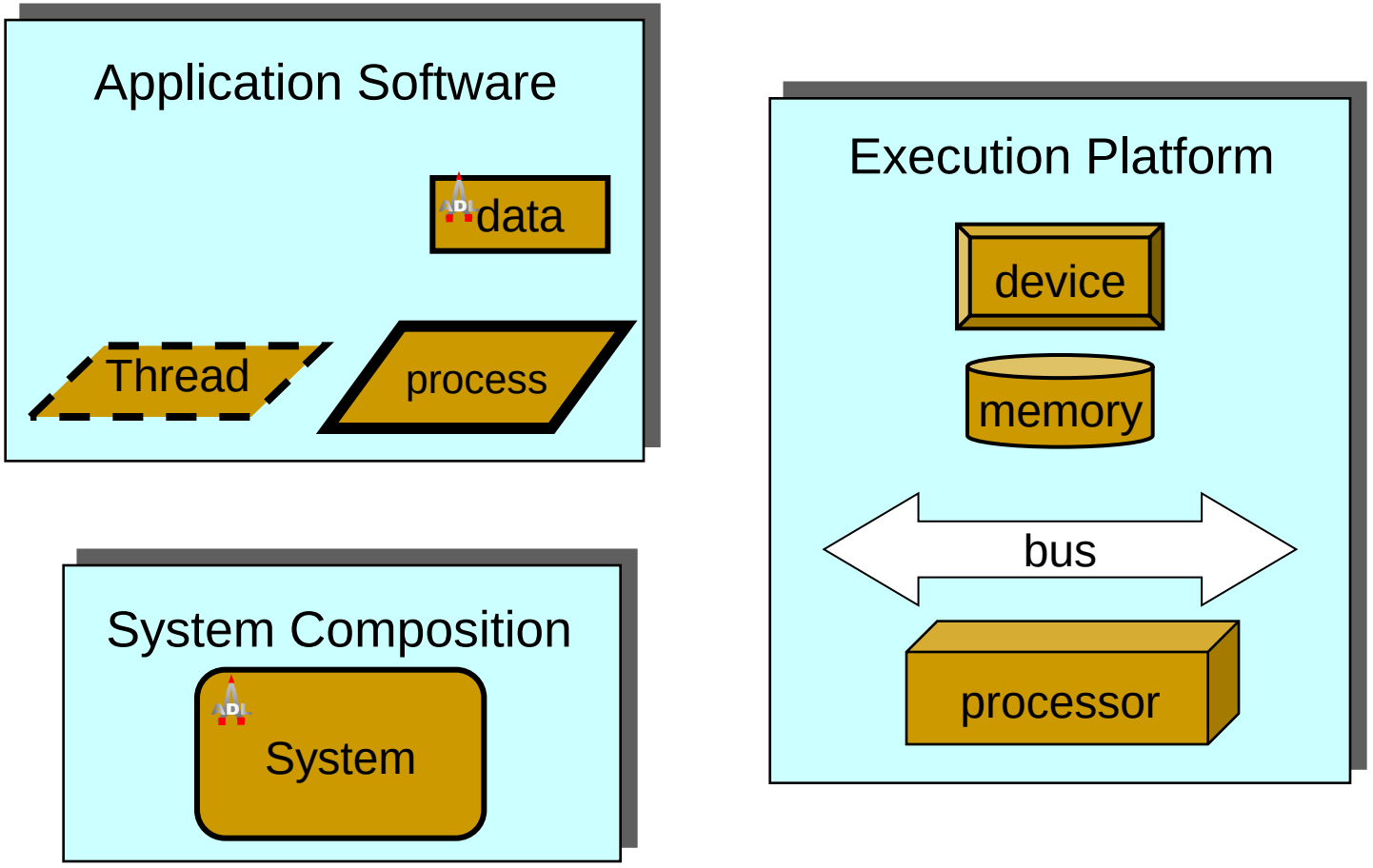
## Interoperability/Portability

- Tailored Runtime Executive
- Standard RTOS API
- Application Components

# Predictable System Integration

- Required, predicted, and actual runtime properties
- Application components designed against functional and non-functional properties
- Application code separated from task dispatch & communication code
- Consistency between task & communication model and implementation through generation
- Feedback into model parameters: refinement of estimated performance values

# AADL Components - Graphical



# Modeling Vocabulary

- Application System
  - Thread
  - Thread Group
  - Process
  - System
  - Package
  - Subprogram
  - Data (shared/message)
  - Data Port
  - Event
  - Event Port
  - Event Data Port
  - Connection
  - Mode
- Execution Platform
  - Processor
  - Memory
  - Device
  - Bus
  - System
- Extension
  - Inheritance
  - Properties
  - Sublanguages (safety, flow, user defined, component behavior ….)
  - Domain Specific Annexes

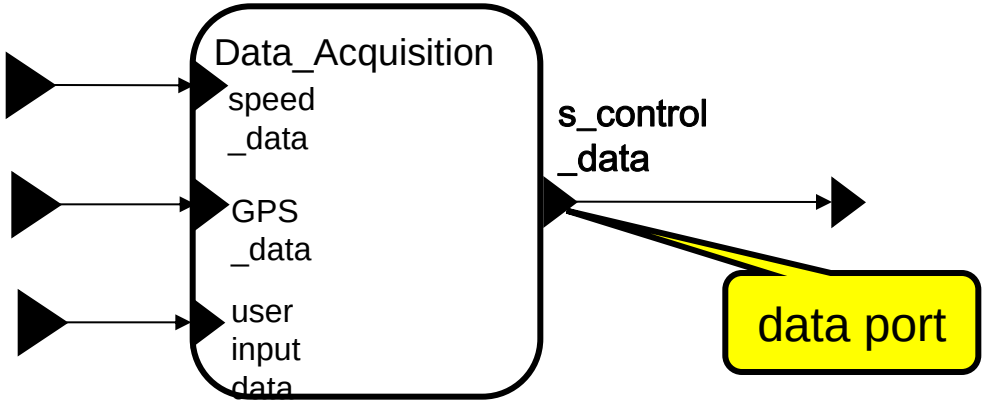
# Graphical and Textual Notation

```
system Data_Acquisition
provides
speed_data: in data metric_speed;
GPS_data: in data position_carthesian;
user_input_data: in data user_input;
s_control_data: out data state_control;
end Data_Acquisition;
```

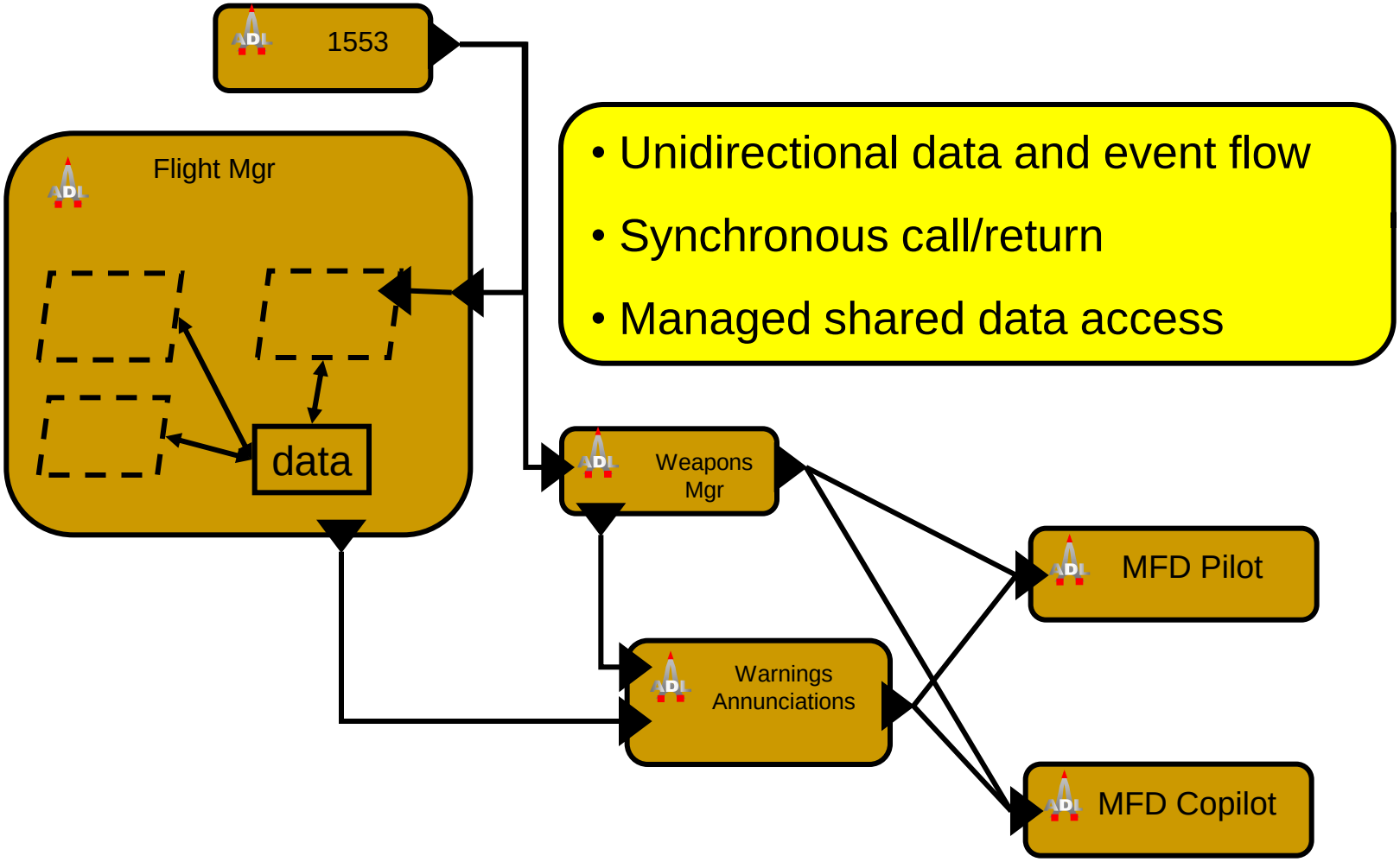
data type of port

data port

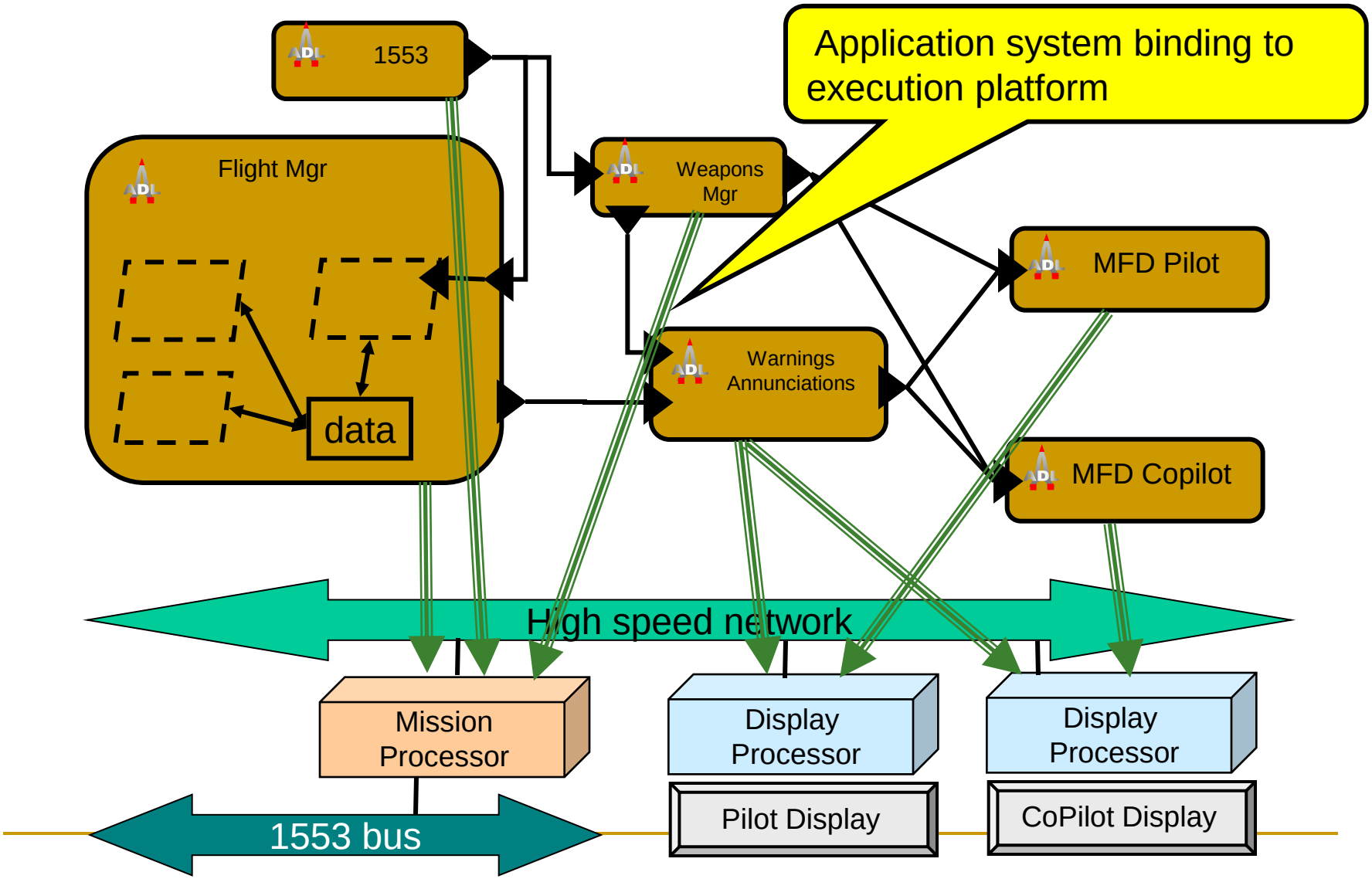
Right-click + View Diagram:



# AADL Component Interaction



# Application System and Execution Platform



# Thread Properties

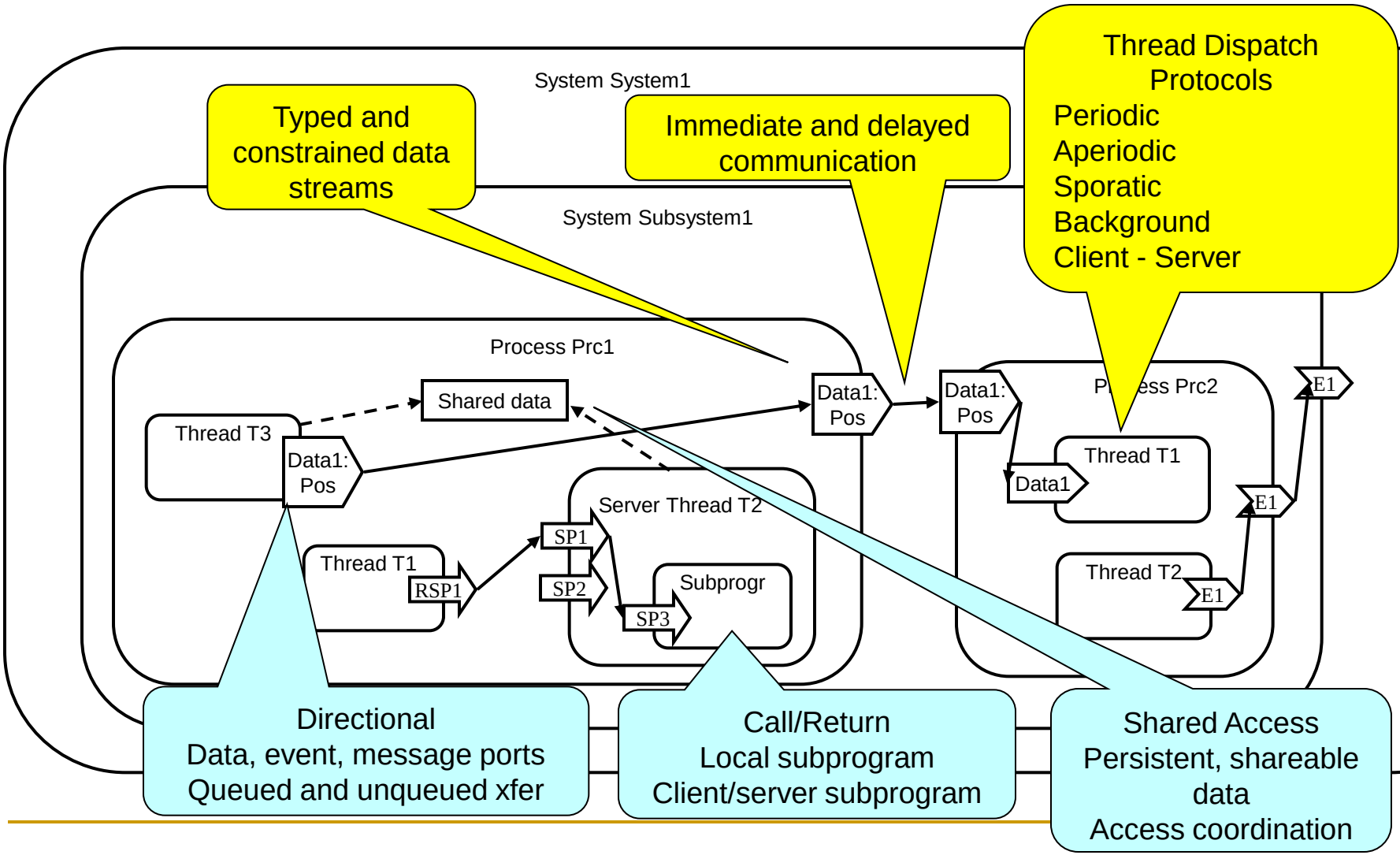
- Dispatch_Protocol => Periodic;
- Period => 100 ms;
- Compute_Deadline => Period;
- Compute_Execution_Time => 20 ms;
- Initialize_Deadline => 10 ms;
- Initialize_Execution_Time => 1 ms;
- Compute_Entrypoint => “Calculate_Trajectory”;
- Source_Text => “waypoint.java”;
- Source_Code_Size => 1.2 KB;
- Source_Data_Size => .5 KB;

Dispatch execution properties

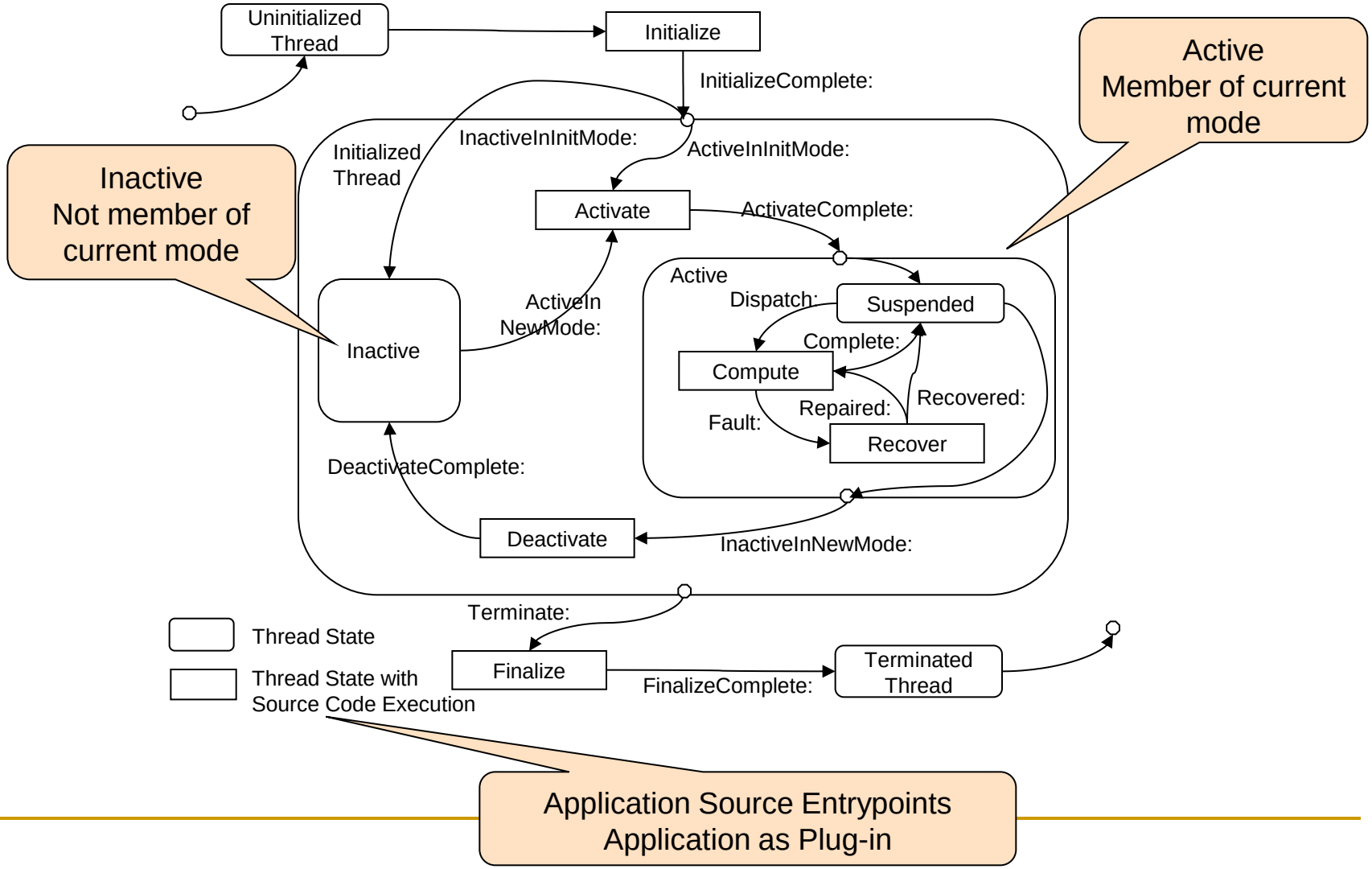
Code function to be executed on dispatch

File containing the application code

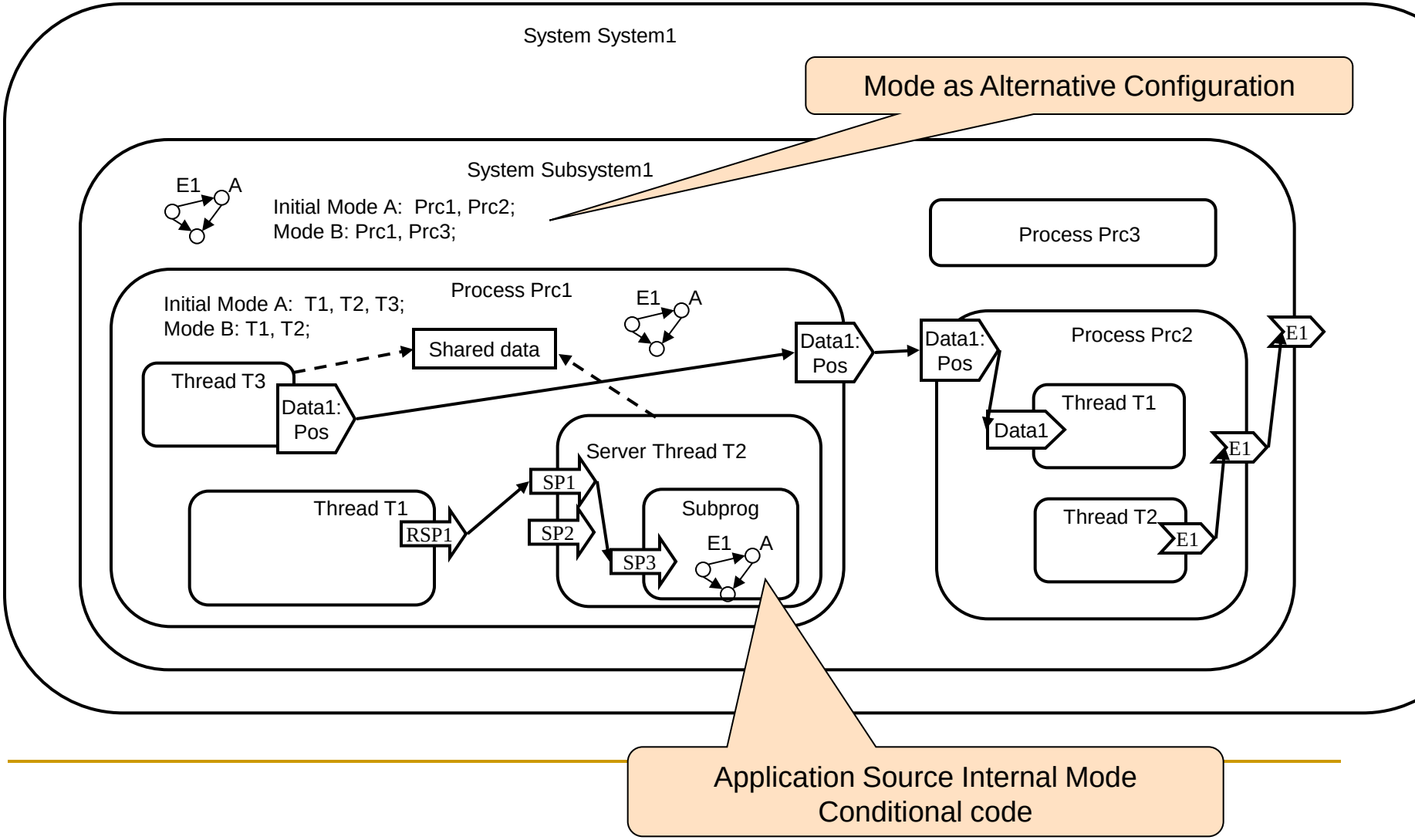
# Task and Interaction Architecture



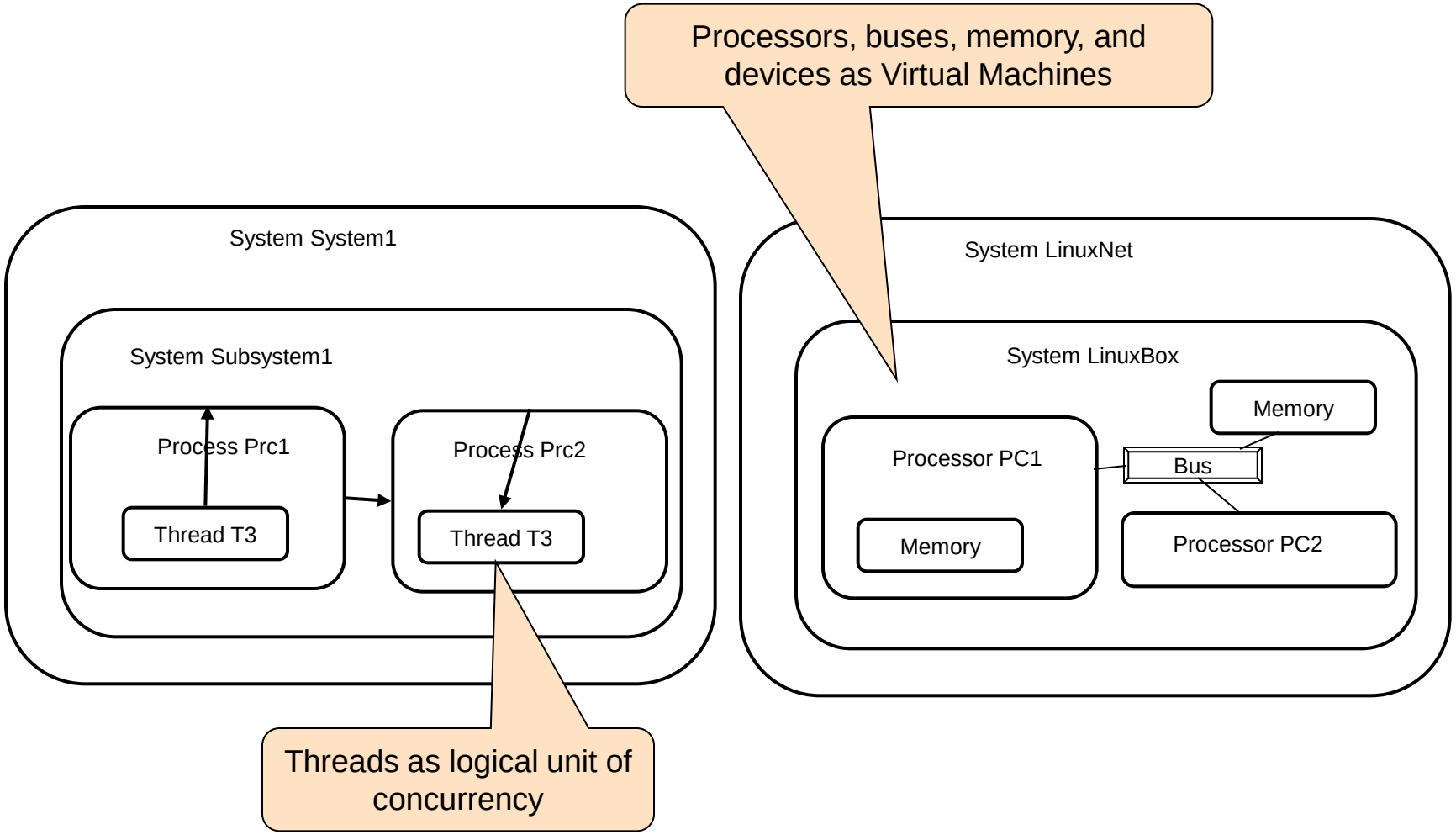
# Thread States with Multiple Modes Possible



# Hierarchical Modes



# Systems and Execution Platforms



# AADL and Scheduling

- AADL provides precise dispatch and communication semantics via hybrid automata
- AADL task and communication abstraction does not prescribe a particular scheduling protocol
  - A cyclic executive can be supported
- Specific scheduling protocols may require additional properties
- Predefined properties support rate-monotonic, fixed priority, preemptive scheduling

This scheduling protocol is analyzable, requires small runtime footprint, provides flexible runtime architecture

# Faults and Modes

- AADL provides a fault handling framework with precisely defined actions
- AADL supports runtime changes to task and communication configurations
- AADL defines timing semantics for task coordination on mode switching
- AADL supports specification of mode transition actions
- System initialization and termination are explicitly modeled

# System Safety Engineering

Capture the results of

- *hazard analysis*
- *component failure modes & effects analysis*

Specify and analyze

- *fault trees*
- Markov models
- partition isolation/event independence

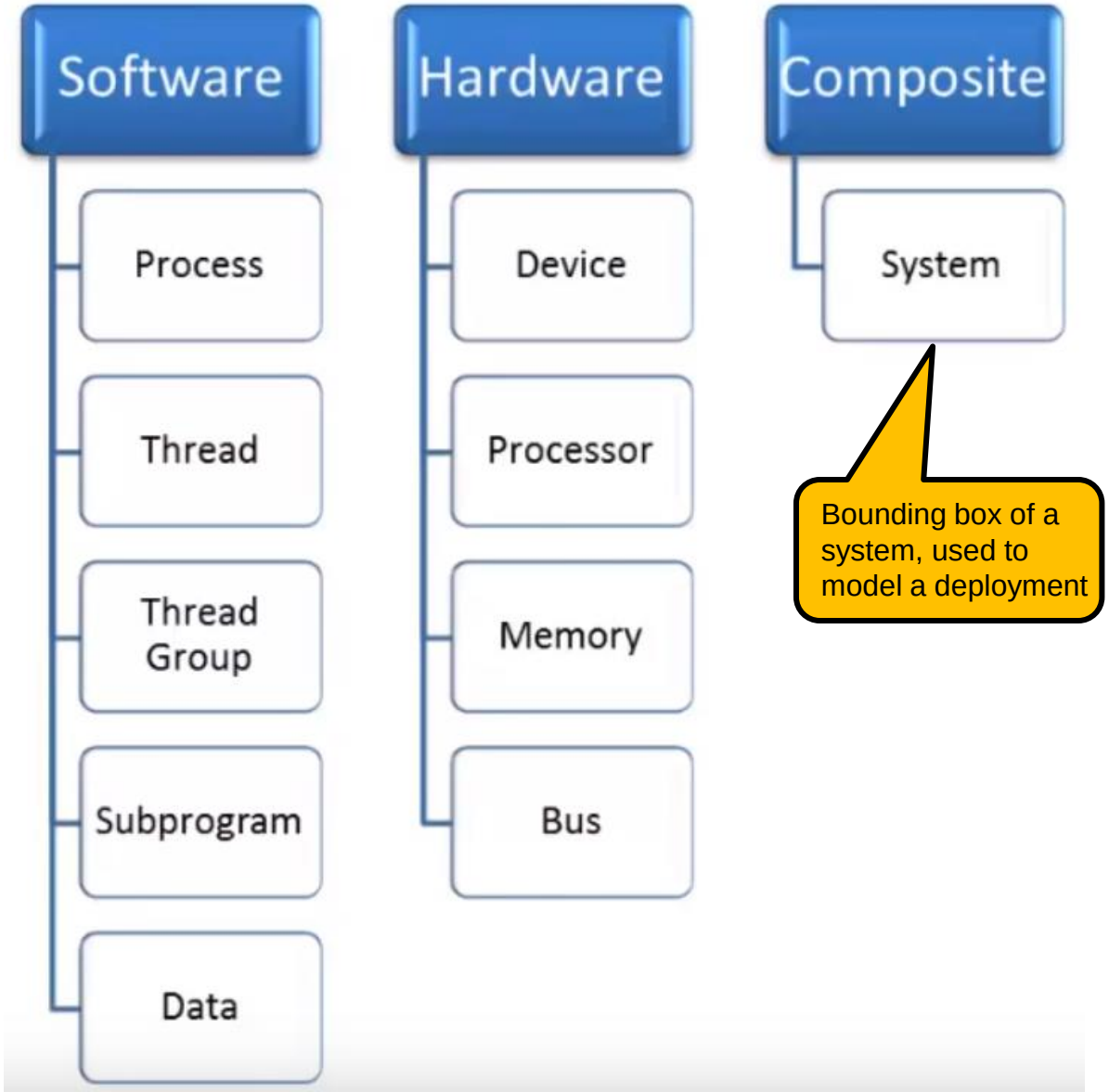
Supported by Error Model Annex

Integration of system safety with architectural design

- enables cross-checking between models
- insures safety models and design architecture are consistent
- reduces specification and verification effort

# Types of AADL Components

- Software
- Hardware
- Composite



# Component Definition

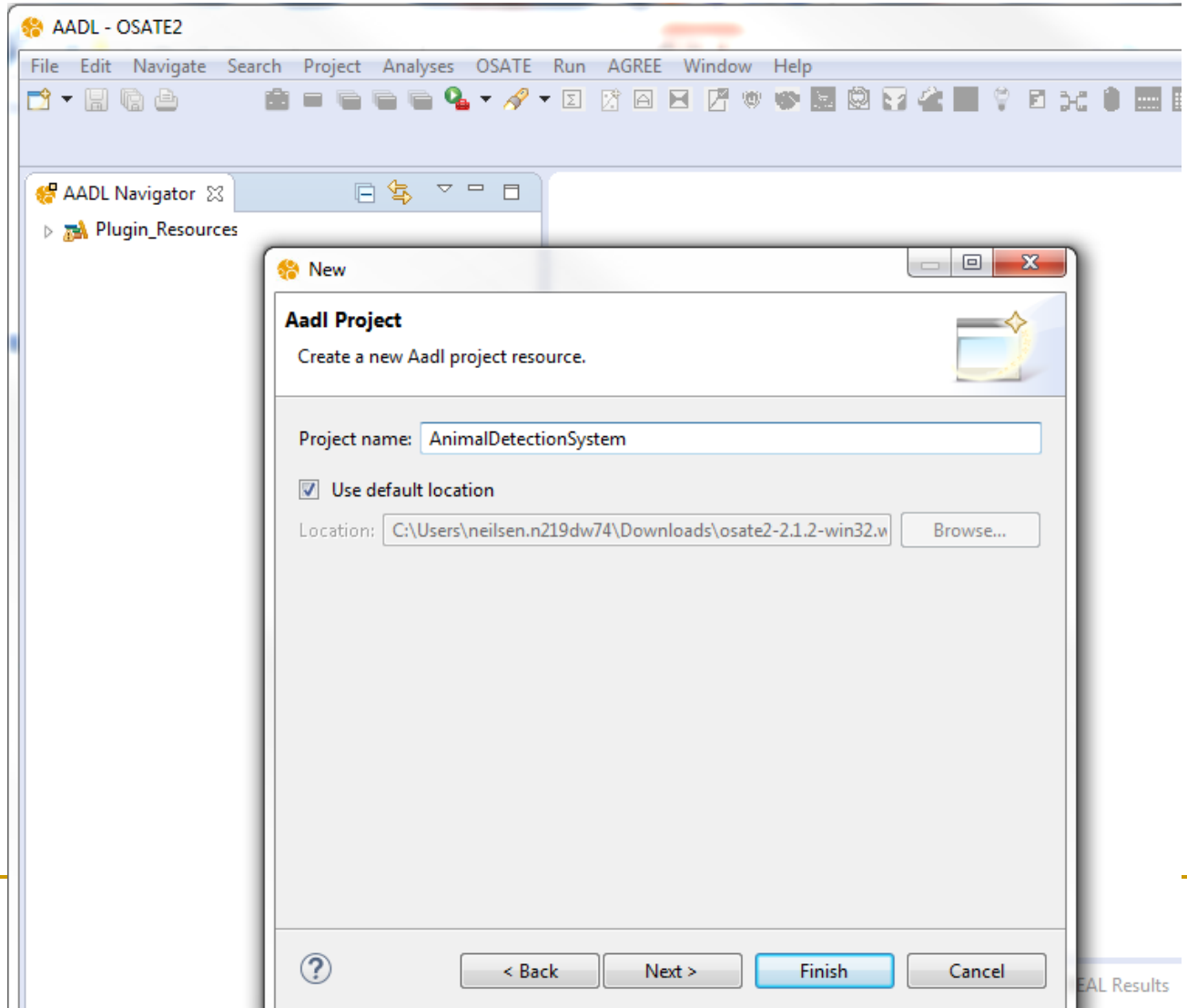
- Type – external view of the component
- Implementation – internal structure or description of a component type

# Example: Animal Detection System

- Build an animal detection device to be used remotely in the field, consisting of a **system unit** which contains software, a processor, and memory (Raspberry Pi).
- The **system unit** receives input from a **heat detecting camera** and a **sound sensor**.
- When an animal is detected via heat or sound, the system sends a signal to a **camera**, which starts recording.

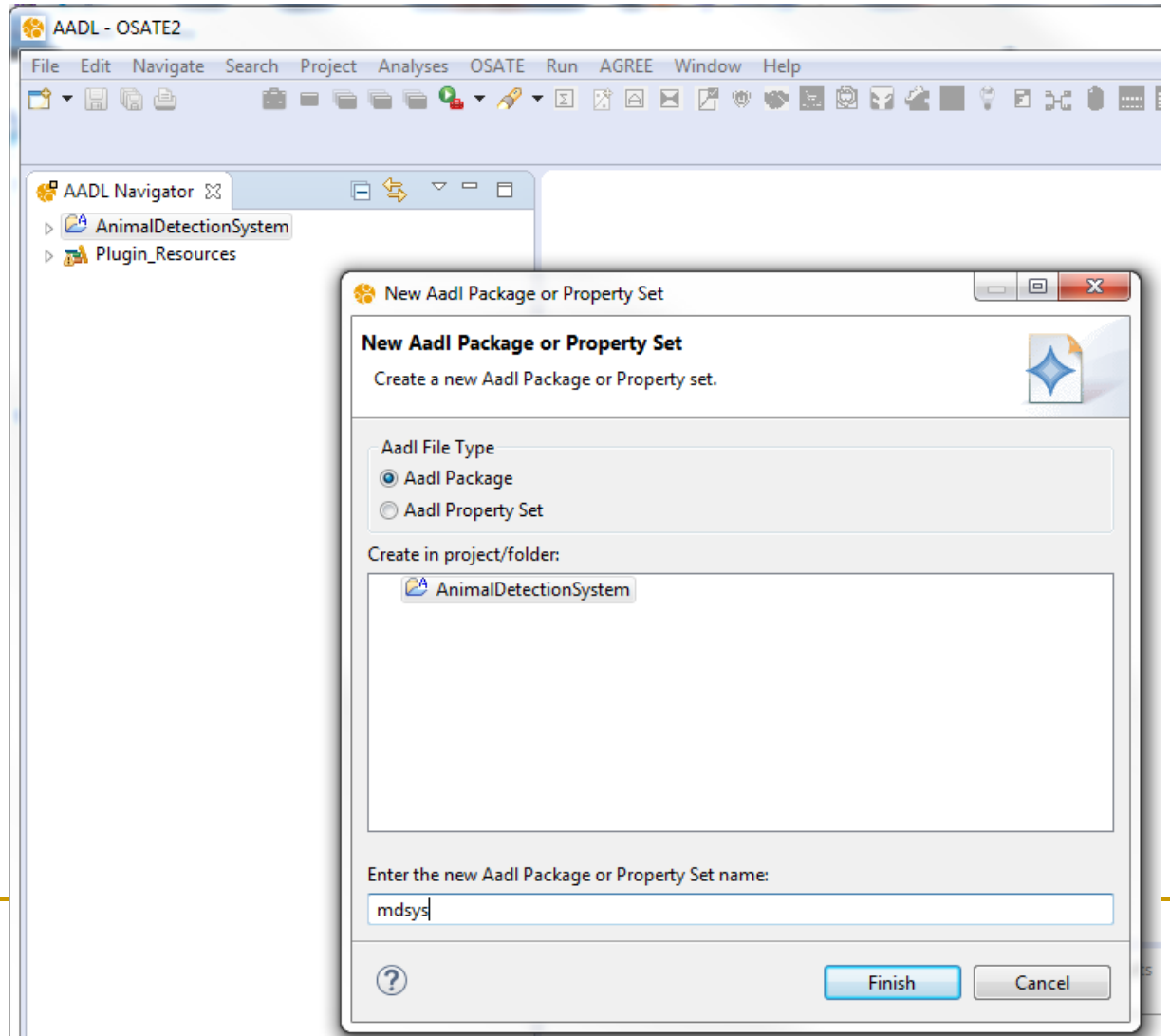
# New AADL Project

- In OSATE2, create a new AADL Project:



# New AADL Model in Project

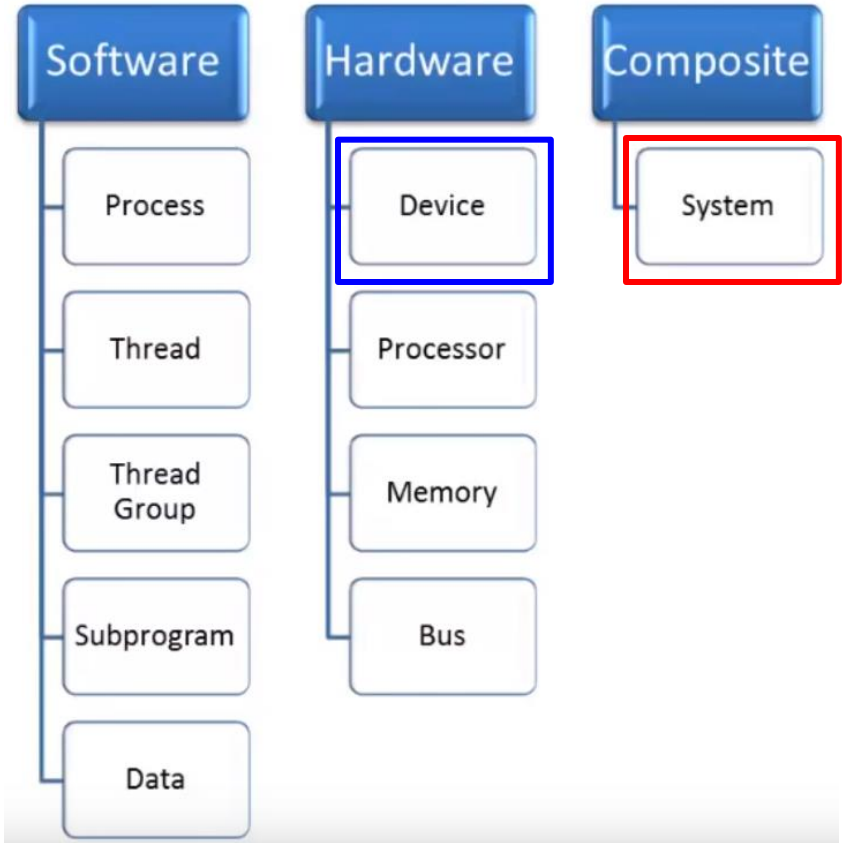
- Enter an AADL package name for a new AADL Model in the project.



# Motion Detection System

*mdsys.aadl

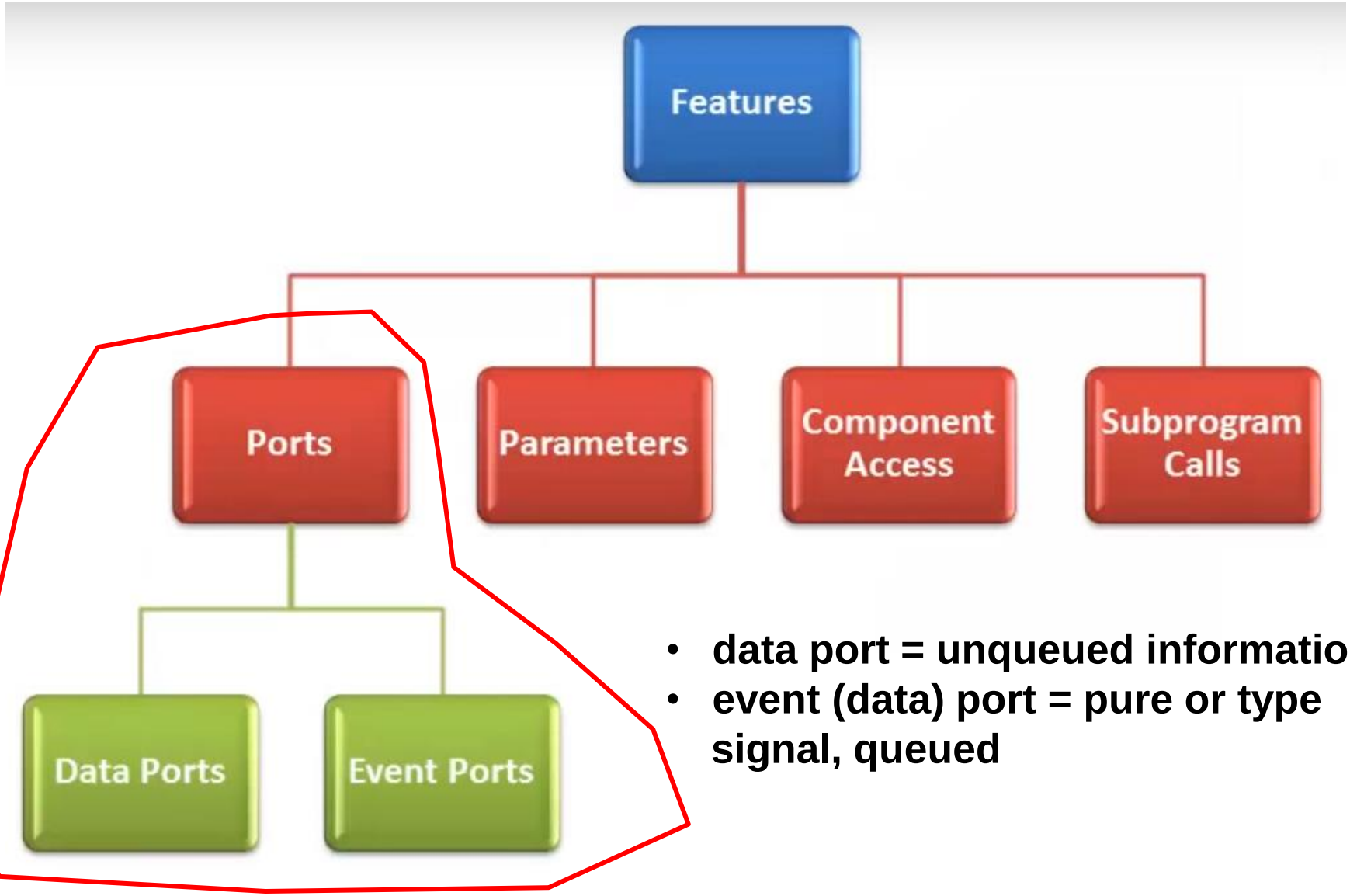
```
package mdsys
public

system AnimalDetectionSystem

end AnimalDetectionSystem;

system implementation AnimalDetectionSystem.impl

end AnimalDetectionSystem.impl;

device HeatDetectionSensor

end HeatDetectionSensor;

device implementation HeatDetectionSensor.impl

end HeatDetectionSensor.impl;

device SoundSensor

end SoundSensor;

device implementation SoundSensor.impl

end SoundSensor.impl;

device Camera

end Camera;

device implementation Camera.impl

end Camera.impl;


end mdsys;
```



# Add Controller Process to System

```
system AnimalDetectionSystem

end AnimalDetectionSystem;

system implementation AnimalDetectionSystem.impl
    subcomponents
        this_heat_detection_sensor : device HeatDetectionSensor;
        this_sound_sensor : device SoundSensor;
        this_camera : device Camera;
        this_controller : process Controller;
end AnimalDetectionSystem.impl;

process Controller

end Controller;

process implementation Controller.impl

end Controller.impl;

device HeatDetectionSensor

end HeatDetectionSensor;
```

# Add Port Features




- **data port = unqueued information**
- **event (data) port = pure or type signal, queued**

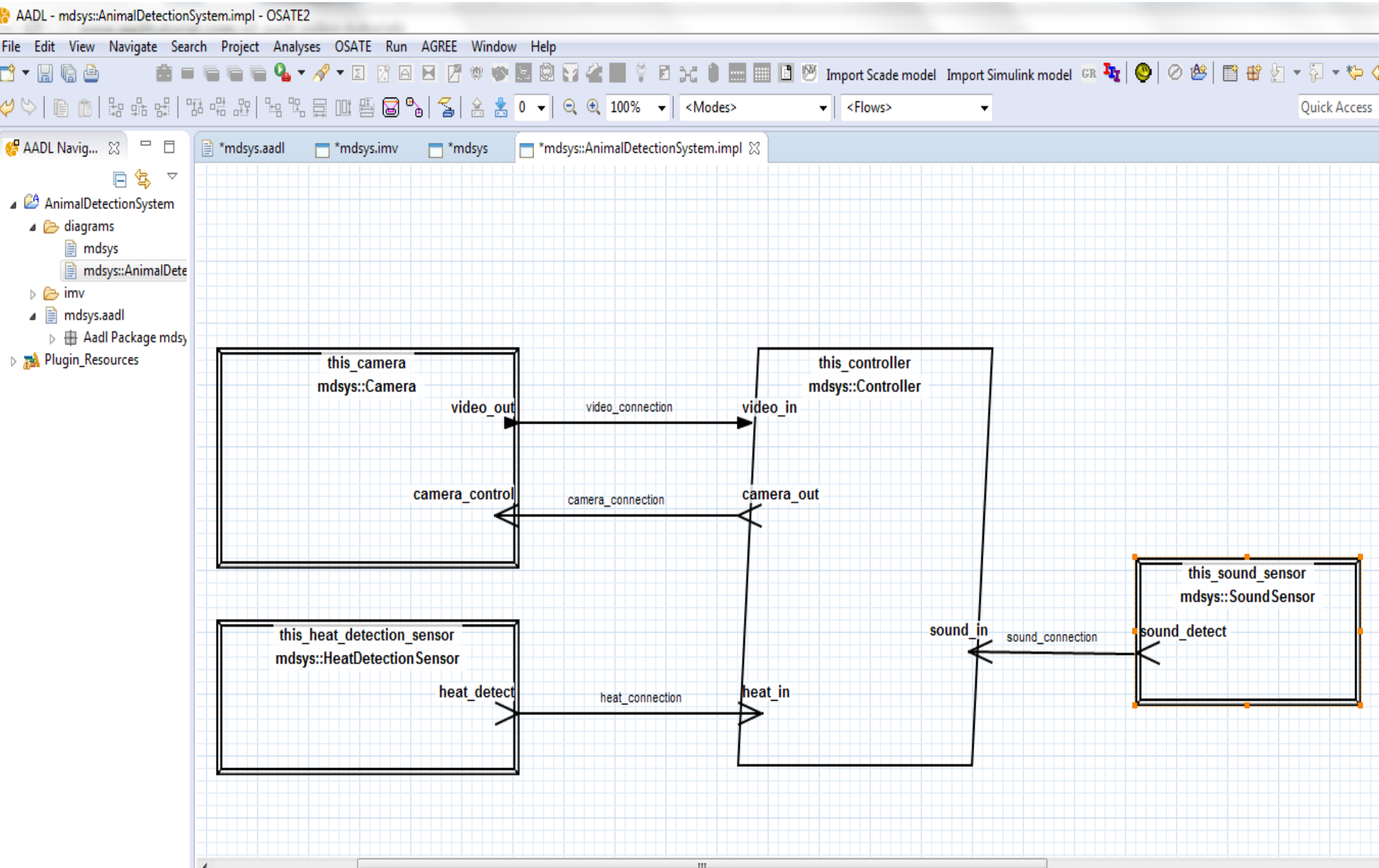
# Add Ports

```
system implementation AnimalDetectionSystem.impl
    subcomponents
        this_heat_detection_sensor : device HeatDetectionSensor;
        this_sound_sensor : device SoundSensor;
        this_camera : device Camera;
        this_controller : process Controller;
    connections
        sound_connection : port this_sound_sensor.sound_detect -> this_controller.sound_in;
        heat_connection : port this_heat_detection_sensor.heat_detect -> this_controller.heat_in;
        camera_connection : port this_controller.camera_out -> this_camera.camera_control;
        video_connection : port this_camera.video_out -> this_controller.video_in;
end AnimalDetectionSystem.impl;

process Controller
    features
        heat_in : in event port;
        sound_in : in event port;
        camera_out : out event port;
        video_in : in data port;
end Controller;

process implementation Controller.impl

end Controller.impl;

device HeatDetectionSensor
    features
        heat_detect : out event port;
end HeatDetectionSensor;
```

# Add Connections between Ports

```
system implementation AnimalDetectionSystem.impl
    subcomponents
        this_heat_detection_sensor : device HeatDetectionSensor;
        this_sound_sensor : device SoundSensor;
        this_camera : device Camera;
        this_controller : process Controller;
    connections
        sound_connection : port this_sound_sensor.sound_detect -> this_controller.sound_in;
        heat_connection : port this_heat_detection_sensor.heat_detect -> this_controller.heat_in;
        camera_connection : port this_controller.camera_out -> this_camera.camera_control;
        video_connection : port this_camera.video_out -> this_controller.video_in;
end AnimalDetectionSystem.impl;

process Controller
    features
        heat_in : in event port;
        sound_in : in event port;
        camera_out : out event port;
        video_in : in data port;
end Controller;

process implementation Controller.impl

end Controller.impl;

device HeatDetectionSensor
    features
        heat_detect : out event port;
end HeatDetectionSensor;
```

# View Diagram with Ports and Connections



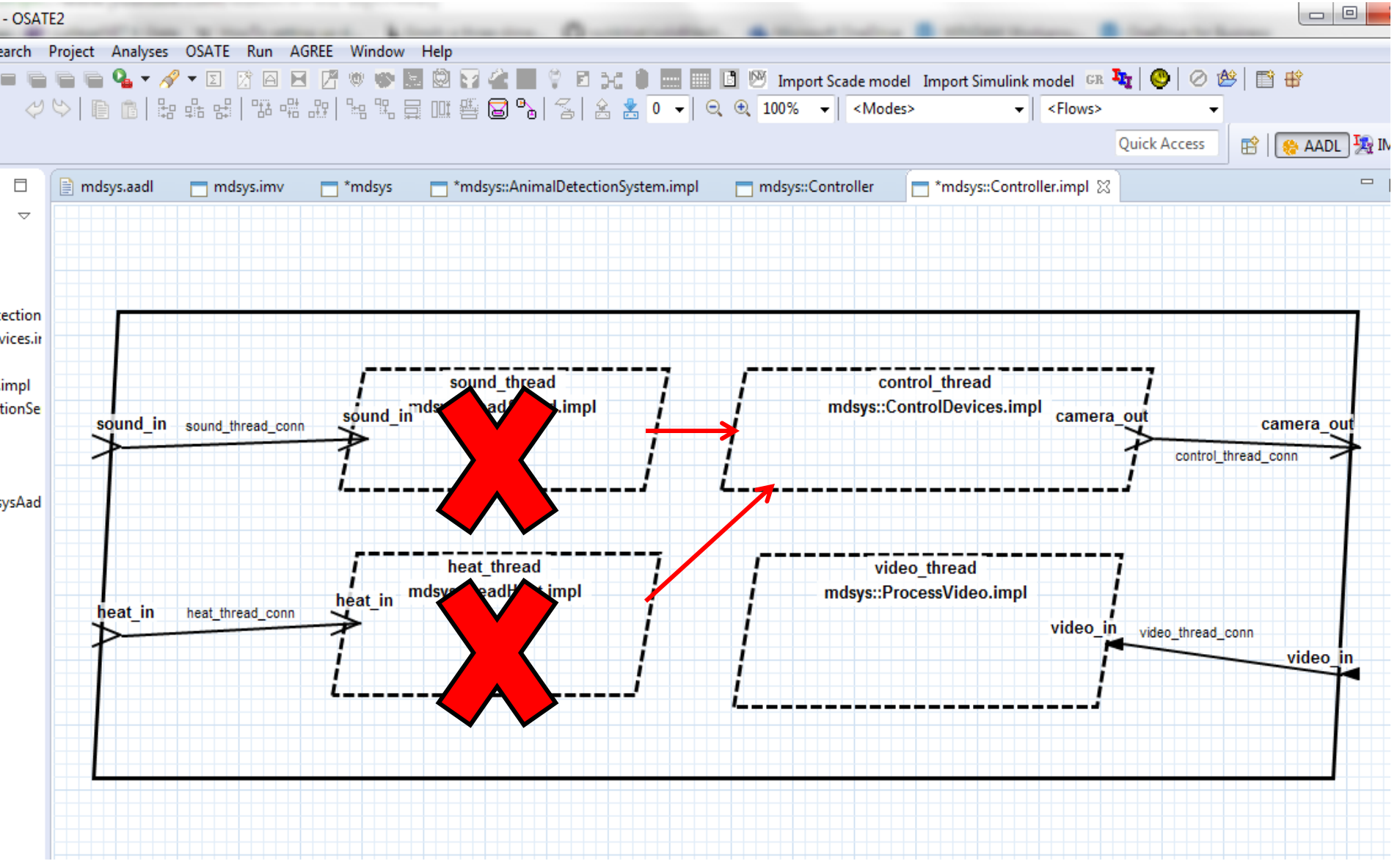
# Add Threads

```
process Controller
    features
        heat_in : in event port;
        sound_in : in event port;
        camera_out : out event port;
        video_in : in data port;
end Controller;

process implementation Controller.impl
    subcomponents
        heat_thread: thread ReadHeat.impl;
        sound_thread: thread ReadSound.impl;
        control_thread: thread ControlDevices.impl;
        video_thread: thread ProcessVideo.impl;
end Controller.impl;

thread ReadHeat
    features
        heat_in : in event port;
end ReadHeat;

thread implementation ReadHeat.impl

end ReadHeat.impl;

thread ReadSound
    features
        sound_in : in event port;
end ReadSound;

thread implementation ReadSound.impl

end ReadSound.impl;
```

# Connect Ports to Threads

```
process Controller
    features
        heat_in : in event port;
        sound_in : in event port;
        camera_out : out event port;
        video_in : in data port;
end Controller;

process implementation Controller.impl
    subcomponents
        heat_thread: thread ReadHeat.impl;
        sound_thread: thread ReadSound.impl;
        control_thread: thread ControlDevices.impl;
        video_thread: thread ProcessVideo.impl;
    connections
        heat_thread_conn : port heat_in -> heat_thread.heat_in;
        sound_thread_conn : port sound_in -> sound_thread.sound_in;
        control_thread_conn : port control_thread.camera_out -> camera_out;
        video_thread_conn : port video_in -> video_thread.video_in;
end Controller.impl;
```

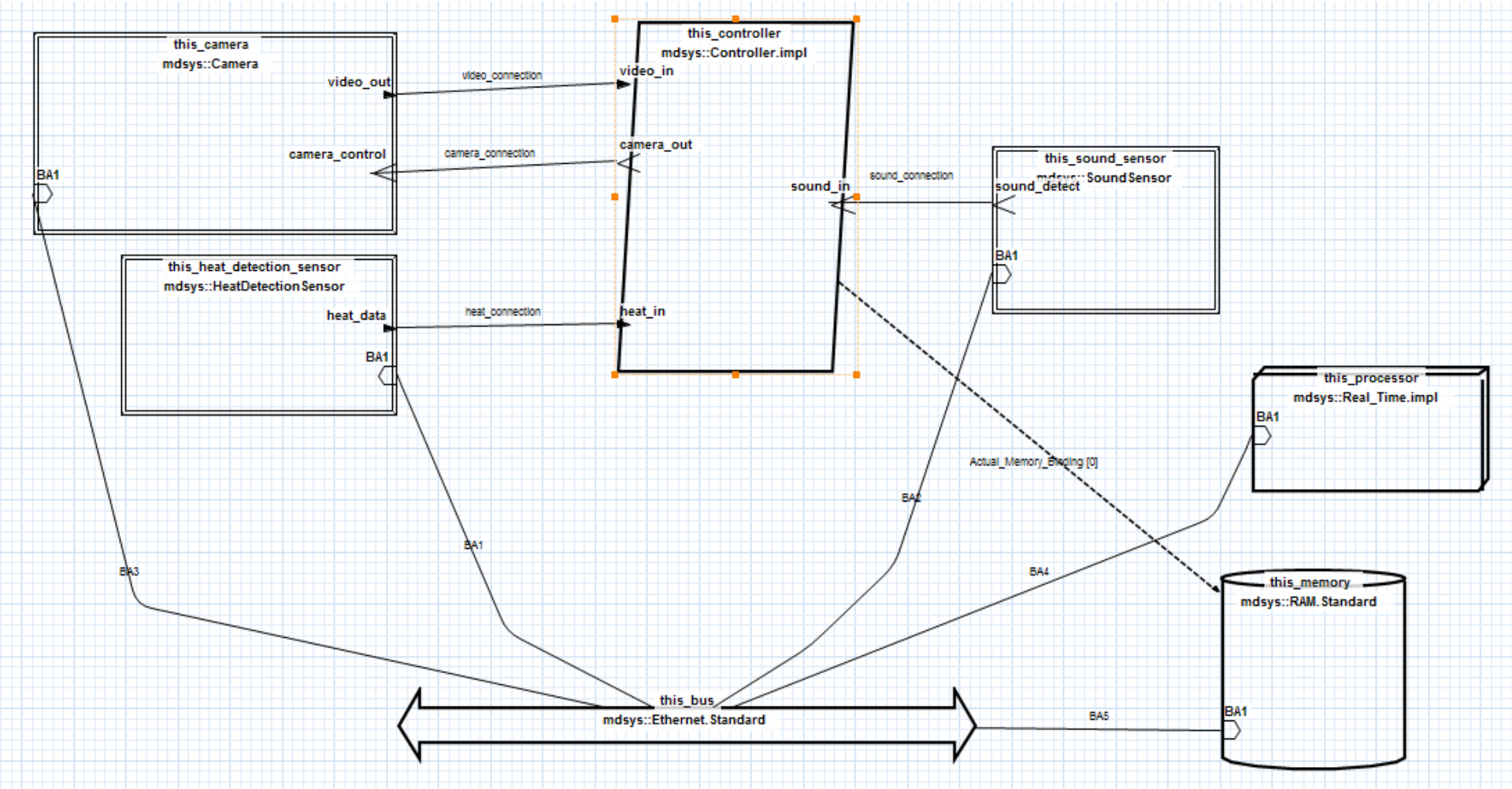
# mdsys::Controller.impl



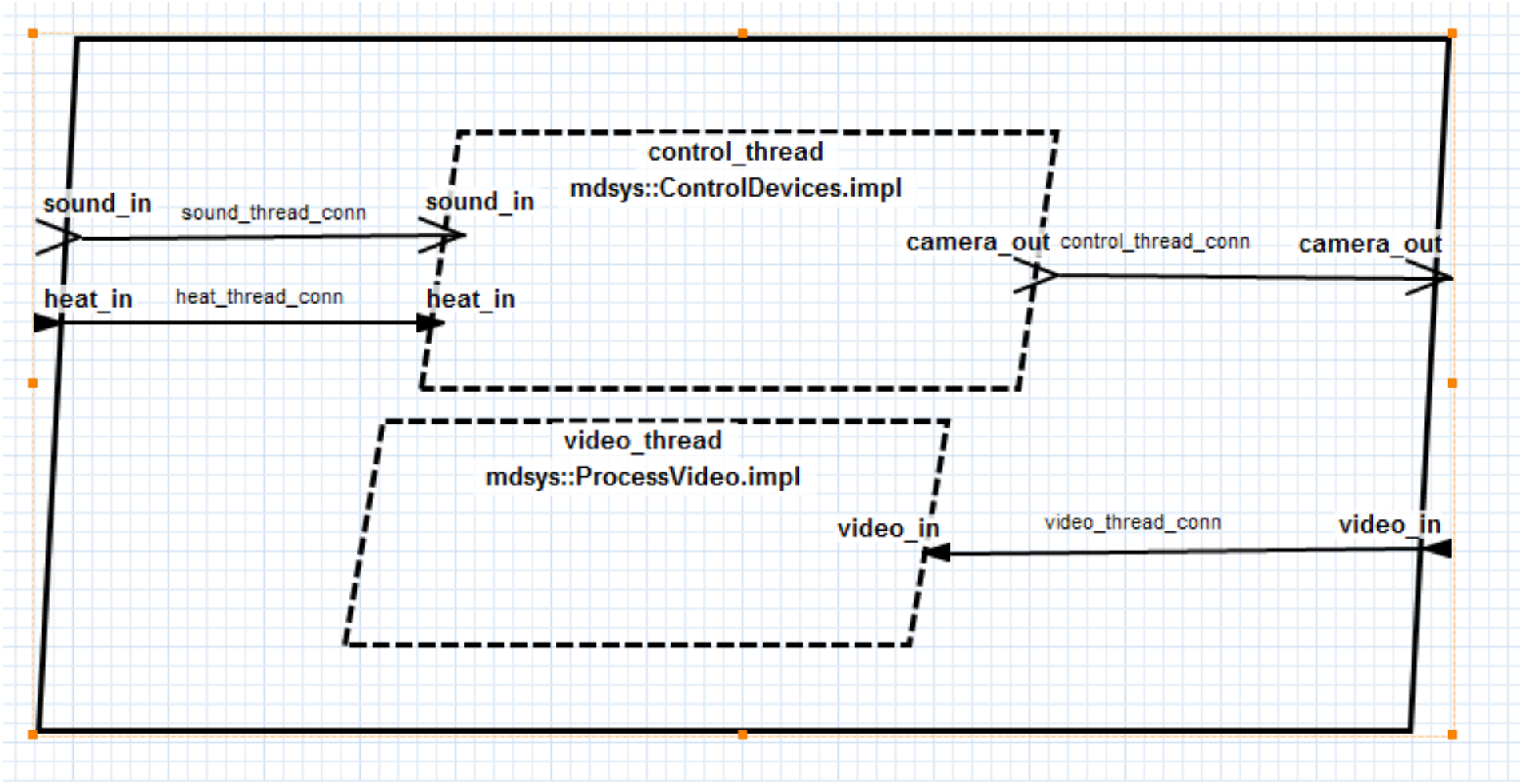
# Add Thread Temporal Requirements

```
thread ControlDevices
    features
        sound_in : in event port;
        heat_in : in data port;
        camera_out : out event port;
    properties
        Dispatch_Protocol => Periodic;
        Compute_Execution_Time => 5 ms .. 8 ms;
        Deadline => 100 ms;
        Period => 100 ms;
        Priority => 2;
end ControlDevices;
```

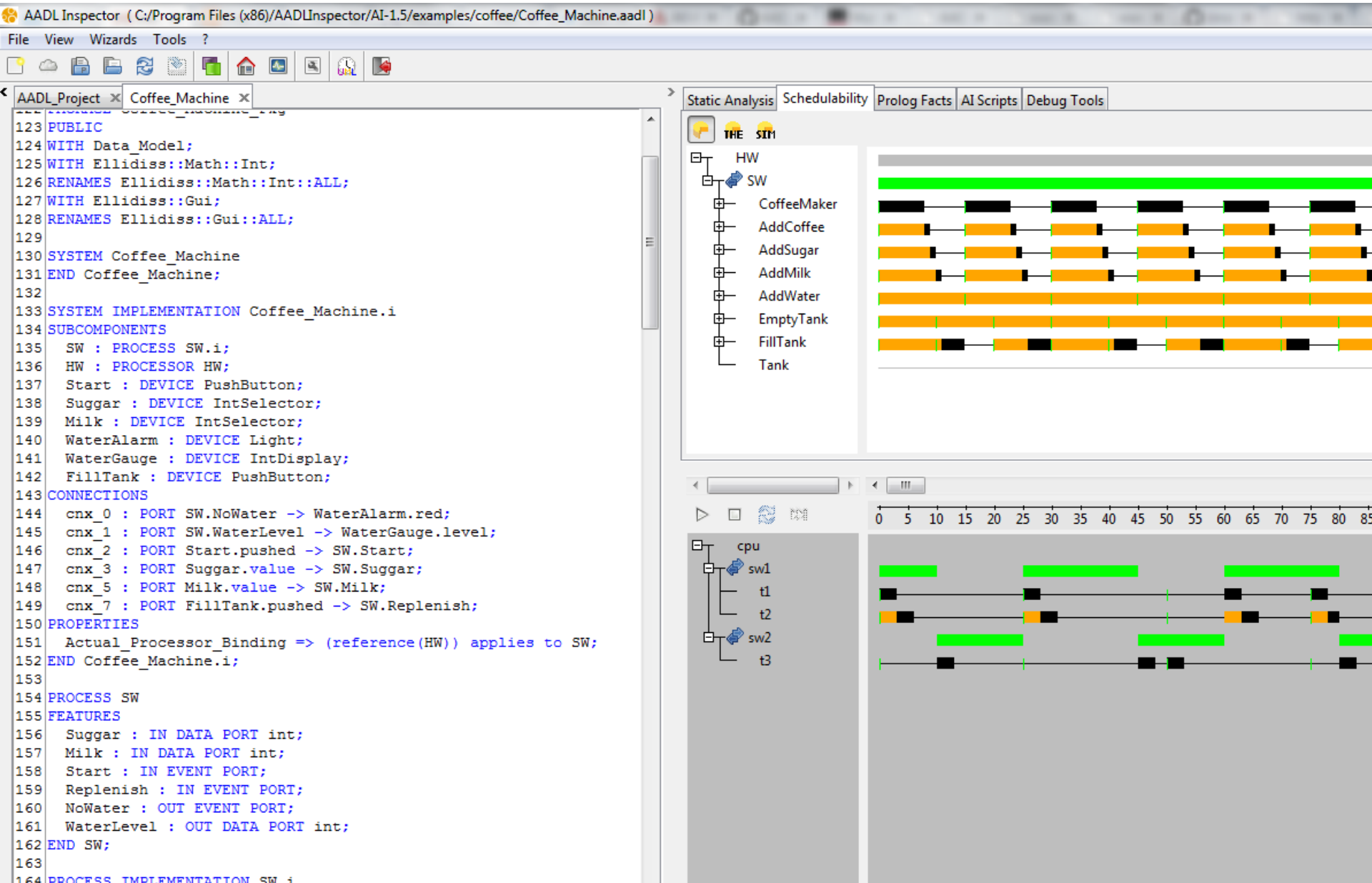
# System


this_camera
mdsys::Camera
video_out
camera_control
BA1
video_connection
camera_connection
this_controller
mdsys::Controller.impl
video_in
camera_out
sound_in
sound_connection
this_sound_sensor
sound_detect
BA1
this_heat_detection_sensor
mdsys::HeatDetectionSensor
heat_data
BA1
heat_connection
heat_in
this_processor
mdsys::Real_Time.impl
BA1
Actual_Memory_Binding [0]
BA2
BA1
BA3
BA4
this_memory
mdsys::RAM.Standard
this_bus
mdsys::Ethernet.Standard
BA5
BA1

# Controller Process



# AADL Inspector –Coffee Machine



# Next Time

- More examples
- Behavior Analysis – real-time analysis