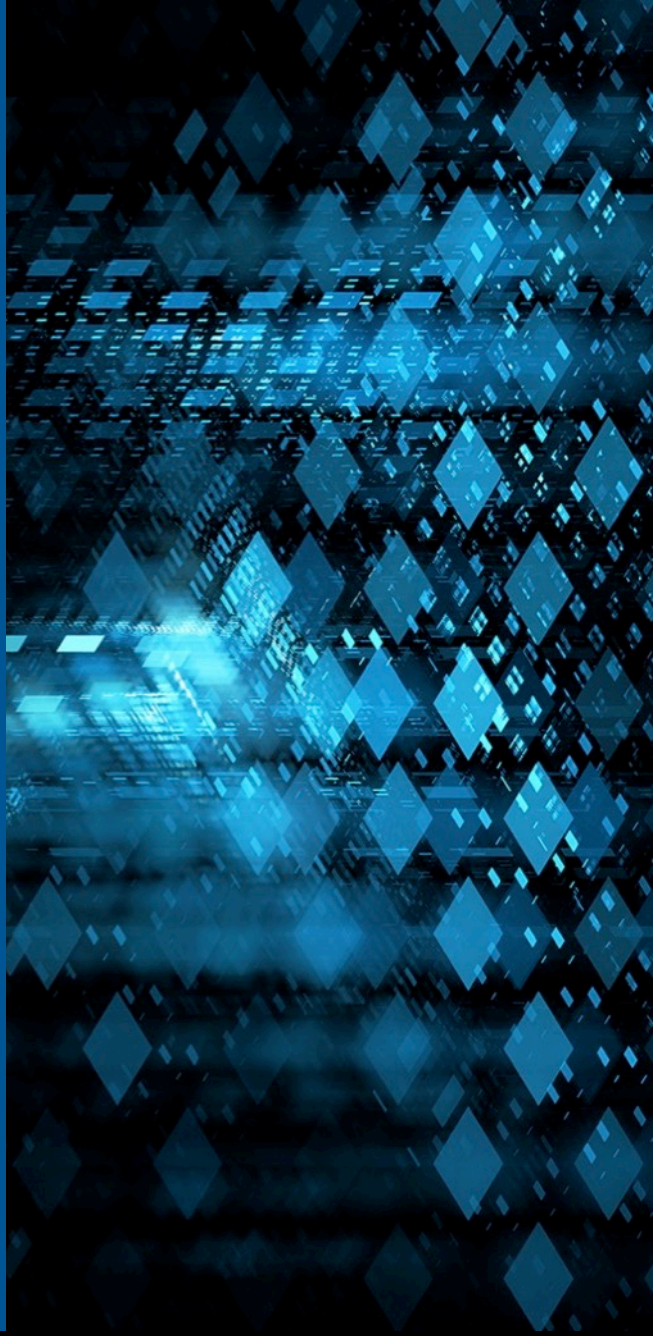
AADL v3

Jerome Hugues

Software Engineering Institute
Carnegie Mellon University
Pittsburgh, PA 15213



This material is based upon work funded and supported by the Department of Defense under Contract No. FA8702-15-D-0002 with Carnegie Mellon University for the operation of the Software Engineering Institute, a federally funded research and development center.

DM19-0625

AADLv3

Table of Contents

AADLv3 table of contents

Goal: Provide an incremental view of AADL concepts

Objective is 4 – x ? Parts, < 60 pages each

Introduction to AADL, Lead: P. Feiler

Part 1: Syntax and general concepts, Lead: P. Feiler

Part 2: Static semantics, Lead: J. Hugues

Part 3: Dynamic semantics, Lead: J. Hugues

Part 4: Property notation and Type System Lead: L. Wrage

Part 5: Property sets & MoC configurations

Allow per part ballot to allow for early prototyping

AADLv3 table of contents

Part 1: Syntax and general concepts

BNF for the core language

AADL concepts: component category, types, implementation, features, bindings, flows, modes…

Type system (L. Wrage)

Property language definition

AADLv3 table of contents

Part 2: Static semantics, i.e.

Goal is to simply present the static structure of an AADL model

Component categories description

New component category: virtual memory

Architecture modeling through containments, connections and bindings

AADLv3 table of contents

Part 3: Dynamic semantics

Default semantics of component category and features

Separates the canonical property-less semantics (P3) from the parametric property-driven semantics (P4)

Rationale is: better to define a per-objective semantics using AADLv3 configuration sets than a long collection of properties that are hard to relate.

AADLv3 table of contents

Part 4 : Property sets & MoC configurations

Catalogue of AADL properties

Parametrization of default semantics through properties

Presentation of specific Model of Computation (e.g. scheduler), or support for verification objectives (e.g. flow latency)

Use v3 configuration sets for specific Model of Computation

Walkthrough: an example

Question: how do I model for scheduling analysis ?

Part 1 defines the general syntax of AADL core

Part 2 defines concepts e.g. threads, virtual processors

Part 3 defines the general state machine of threads, ports

Part 4 introduces properties for scheduling and introduces specific task models as configuration set

If you're familiar with AADL, Part 4 should provide all required information and examples.

Note: to some extend, Part 4 could be read as a collection of annex document, covering flow analysis, scheduling, memory, data modeling, etc.

AADLv3

Part 2 – Static Semantics

About Part 2 semantics

Goal is to present the static structure of an AADL model

Architecture modeling through containments, connections and bindings

Component categories description

New component category: virtual memory

Textual description + legality rules + default interpretation of legality rules

Rule of thumb: any AADL tool **must** support these rules: they are the core of the AST analysis and name resolution mechanisms

Part 2 organization

For each component category

- Design intent: purpose of this component category, interaction with other categories
- Legality rules: allowed features categories, subclauses, etc.
- Semantics: text-based interpretation of the legality rules *only*

No section on properties -> moved to P4 for clarity

If list of properties per category required, could be provided as appendix to the document to facilitate cross-referencing

Component category

4 categories

- Software: `thread, thread group, process, data, subprogram, subprogram group;`
- Execution platform: `memory, virtual memory, processor, virtual processor, bus, virtual bus, device`
- Generic: `abstract`
- Composite: `system`

Change from v2:

- Suppression of call sequences (use subcomponents instead)
- Introduction of virtual memory
- Update to match AADLv3 new concepts for features, bindings

On model completeness

Must be clear on the requirements on model completeness, and prevent overspecification

> e.g. in v2, 13.3 (5) “All software components for a process must be bound to memory components that are all accessible from every processor to which any thread contained in the process is bound. “

For code generation, this brings no value for some RTOS (e.g. RTEMS, FreeRTOS), but is required for others (e.g. ARINC653)

$\Rightarrow$ Model completeness is a per-objective issue

Objective: what is my objective when modeling? What type of credits to I want to gain from this modeling activity?

On component categories

Question: should we split Execution Platforms (virtual bus/processor) from Hardware elements?

- Rationale: VB/VP/VM control executions of software, using resources provided by hardware elements, required by software elements.
- They act as a resource broker
  - Virtual Bus: communication protocols, message arbitration
  - Virtual Processor: scheduling policy
  - Virtual Memory: ? Security e.g. red/write/execute

Memory vs Virtual Memory (June 16)

- Separation of physical and logical concerns
  - Binding of logical to physical
- Memory
  - Storage with binding points
  - Need for representing different section of memory addresses: binding points have properties to indicate base address and range (size)
  - Memory binding point on devices can model device registers without requiring memory subcomponents.
- Virtual memory roles
  - Represent logical addresses that are mapped to addresses in different components in the platform
  - Logical resource with capacity/budget
  - Logical containment regions or segments of address space
- Memory as system (platform) subcomponents
  - Subcomponents as binding points
  - Memory system architecture with connectivity via bus
  - Platform with memory and processor

Virtual Memory component category

Question: what is the static semantics of virtual memory ?

- Define the logical view of memory, a process perspective ?
- Binding VM to device/system to capture flash storage?

Or is data/process sufficient to achieve the same goal?

- Data being structured memory, process being protected memory

Most of the semantics of VM seems driven by properties

- Stack/BSS/Heap/Code memory only ?
- Read/write/execute ?

Use cases:

- Binding Virtual Memory to Memory (Hardware) as a deployment
- Binding Process/Thread/Data/Subprogram to VM ?
- VM inside VM ?

Virtual memory and composition

Ultimately:

- A process (e.g. software partition) is bound to both VM and VP
- VP is then bound to a processor, VM to a memory
- And finally processor and memory are connected

Otherwise, architecture is inconsistent

But if some of these bindings are not done, model is “just” incomplete w.r.t. some verification objectives

Role split between virtual processor and processor (June 16)

Virtual Processor :- OS + scheduler configuration, security policies, health monitoring, contribution to fault propagation, etc.

Processor :- physical CPU (chip, core, etc.), contributes to fault propagation, ?

For code generation purpose, having only a virtual processor makes sense: you target an OS, the actual physical CPU is irrelevant, and deployment to this physical one depend on the deployment strategy

For safety analysis, one may want to have threads (functions) and processors. Virtual processors may or may not be required depending on its contribution to errors propagation/containment.

Definition of Virtual Processor

AADLv2.2 6.2 (1)

"A virtual processor represents a logical resource that is capable of scheduling and executing threads and other virtual processors bound to them. Virtual processors can be declared as a subcomponent ~~of a processor~~ or another virtual processor, i.e., they are implicitly bound to the ~~processor or~~ virtual processor they are contained in. Virtual processors can also be declared separately, that is as a subcomponent of a system component, and explicitly bound to a processor or virtual processor. "

Processor, device as systems

System are composite elements.

Processors and devices are also systems when we open the box.

Refining a processor (black box) into system (white box) violates type system (no type promotion), natural modeling step that must be supported

Using Implemented_As does not work: no feature connection, breaks mode or fault propagation

Processor, device as systems

Proposal to extend the definition of processor

AADLv2.2 6.1 (1)

"A processor is an abstraction of hardware and software that is responsible for scheduling and executing threads and virtual processors that are bound to it. A processor also may execute driver software that is declared as part of devices that can be accessed from that processor. Processors may contain memories and may access memories and devices via buses."

AADLv3

"A processor is a hardware system that is responsible for providing access to resources it controls to threads and virtual processors. A processor may control and give access to subcomponents: internal processing cores, memory elements (e.g. cache, flash storage) or internal devices."

Containment clarifies propagation of faults, mode changes, etc (loss of processor -> loss of all subcomponents, loss of subcomponents degrade processor capability), preserves modularity and type system

Processor, device as systems

Similarly, for devices

AADLv2.2 6,6 (1)

"A device component represents dedicated hardware within the system, entities in the external environment, or entities that interface with the external environment. [..] Devices may internally have a processor, memory and software ~~that can be modeled in a separate system declaration and associated with the device through the Implemented_As property~~."

AADLv3

"A device component represents dedicated hardware within the system, entities in the external environment, or entities that interface with the external environment. [..] Devices may internally have a processor, memory and software **modeled as subcomponents**"

Rationale similar to processor: use containment to ease failure analysis, preserve types, etc.

Virtual bus/bus

No major change for the moment

Wait for security discussion to see if improvements are needed

e.g. composition of protocols (cyphering, authentication)

Patterns to capture execution of protocol code ?

- Attach an entrypoint on the bus to capture the code to run ? .. but not sufficient if producer?/consumer uses different APIs
- Use subprogram access so that a VB can requires subprograms provided by a VP (OS/library) ?

About data types

(Pending text on types system)

About call sequences

Change highlight: suppression of call sequence

Rationale: CS breaks symmetry in components hierarchy

```
--  AADLv2
thread implementation P.Impl
calls
   Mycalls: { S : subprogram Spg; };
connections
   parameter S.I -> I;
end P.Impl
```

```
--  AADLv3
thread implementation P.Impl
subcomponents
   S : subprogram Spg;
connections
   parameter S.I -> I;
end P.Impl
```

Directly reuse rules for subcomponents

Subprogram instances reside in attached process memory space

A Method of Implementation may decide to duplicate this subprogram (inlining) or have one instance per process

General updates

Other updates (in progress)

Features category to be reflected (pending completion of P1)

Link with types (e.g. parameter)

Access to data

Subprograms as accessors to data component types as in v2

AADIv3

Part 3 – Dynamic semantics

AADLv3 table of contents

Part 3: Dynamic semantics

Default semantics of component category and features

They provide high-level rules for processing an AADL model and interpret its behavior

This behavior is later refined on a per-property basis in P4

Rule of thumb: an AADL tool **must** support these rules in the case they process the AADL model for one verification objective

Status update June 2019

Document split from AADLv2.2 document

Must revisit the definition of the hybrid automata

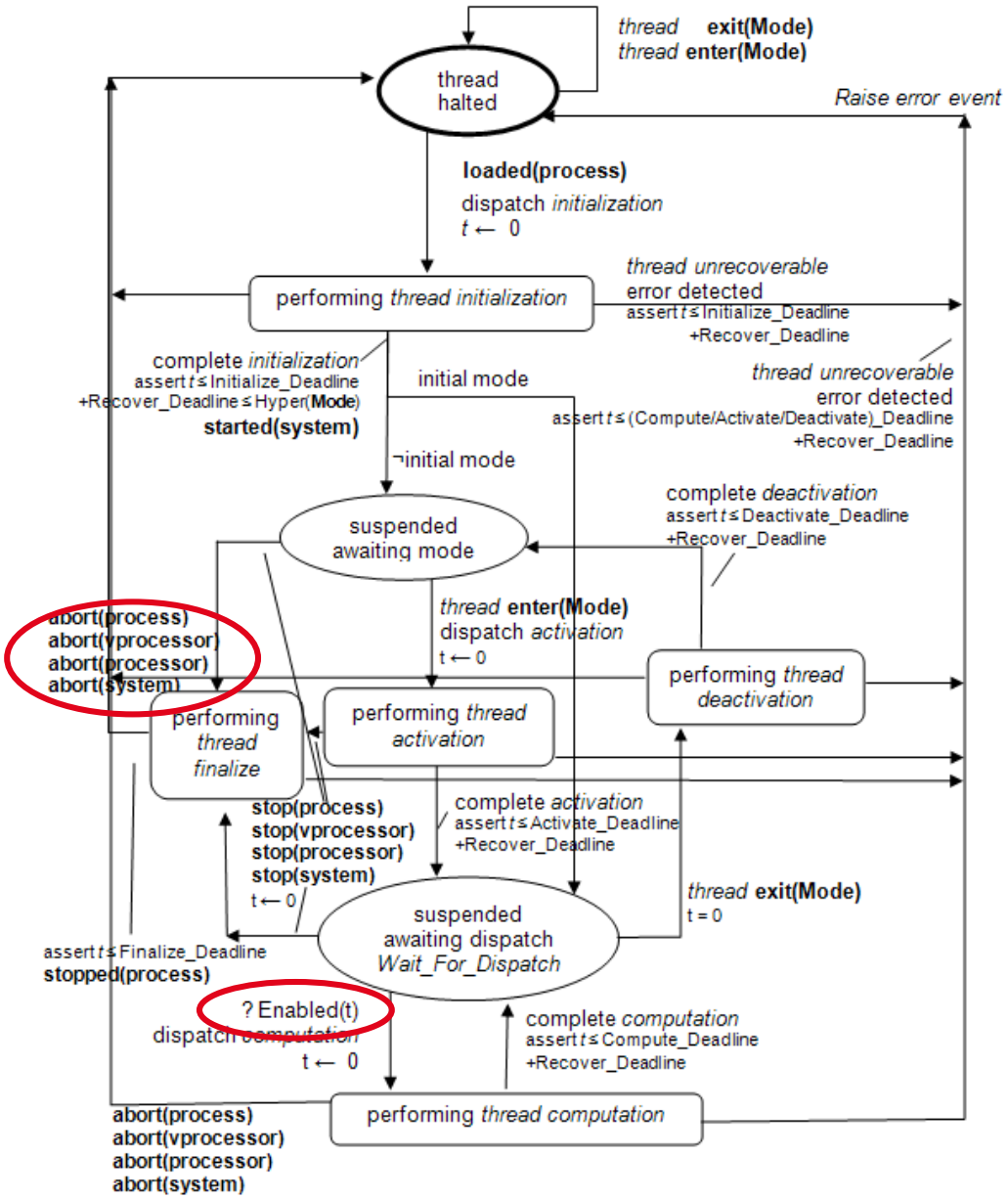
For the moment,

- informal states combined with RTS call (abort, await_dispatch, etc.)
- No clear rules for composing automata, it is implicit that automata at one level “controls” subcomponents
  - How to clarify this part?

Thread automata

Most complex one

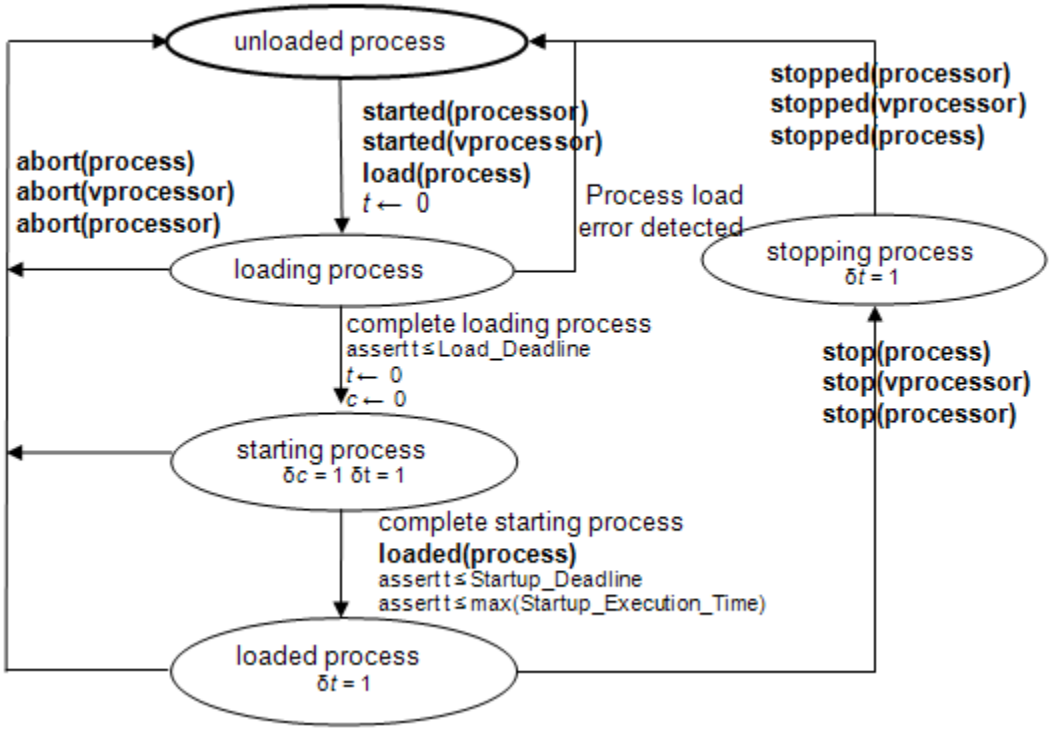
- RTS calls emerge from thread, control other automata
- *but* ?Enabled and dispatch ‘implemented’ by (virtual) processor scheduler
- State when thread blocked on resources as part of “performing thread computation” ?



Automata

Process on top of OS abstraction.

Initialization of OS vs. initialization of process (user code)

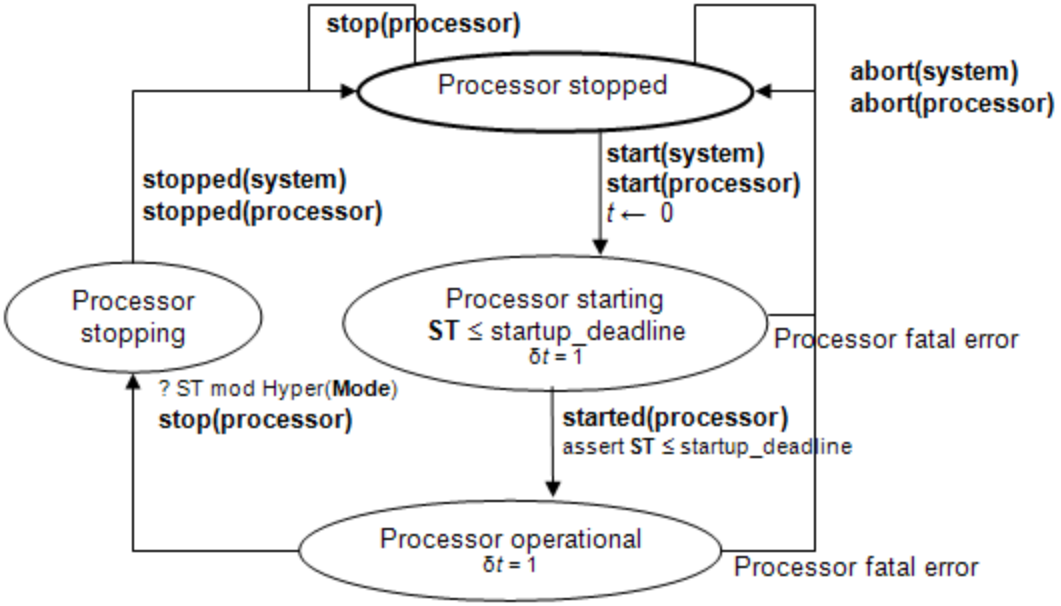


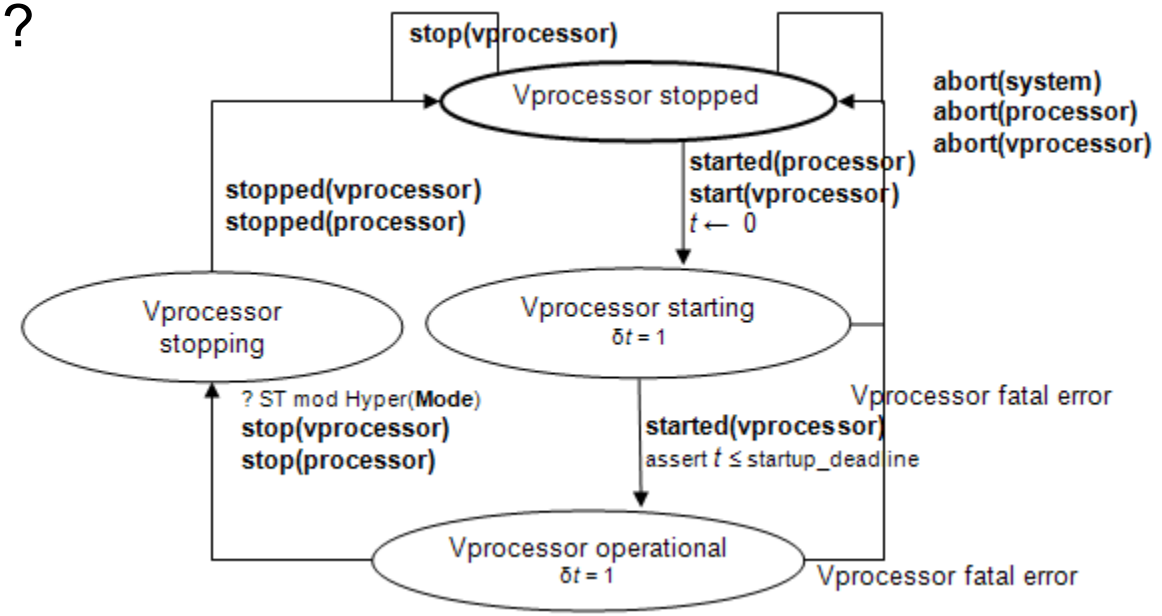
Automata

Start up of associated devices? Hidden in corresponding driver thread?

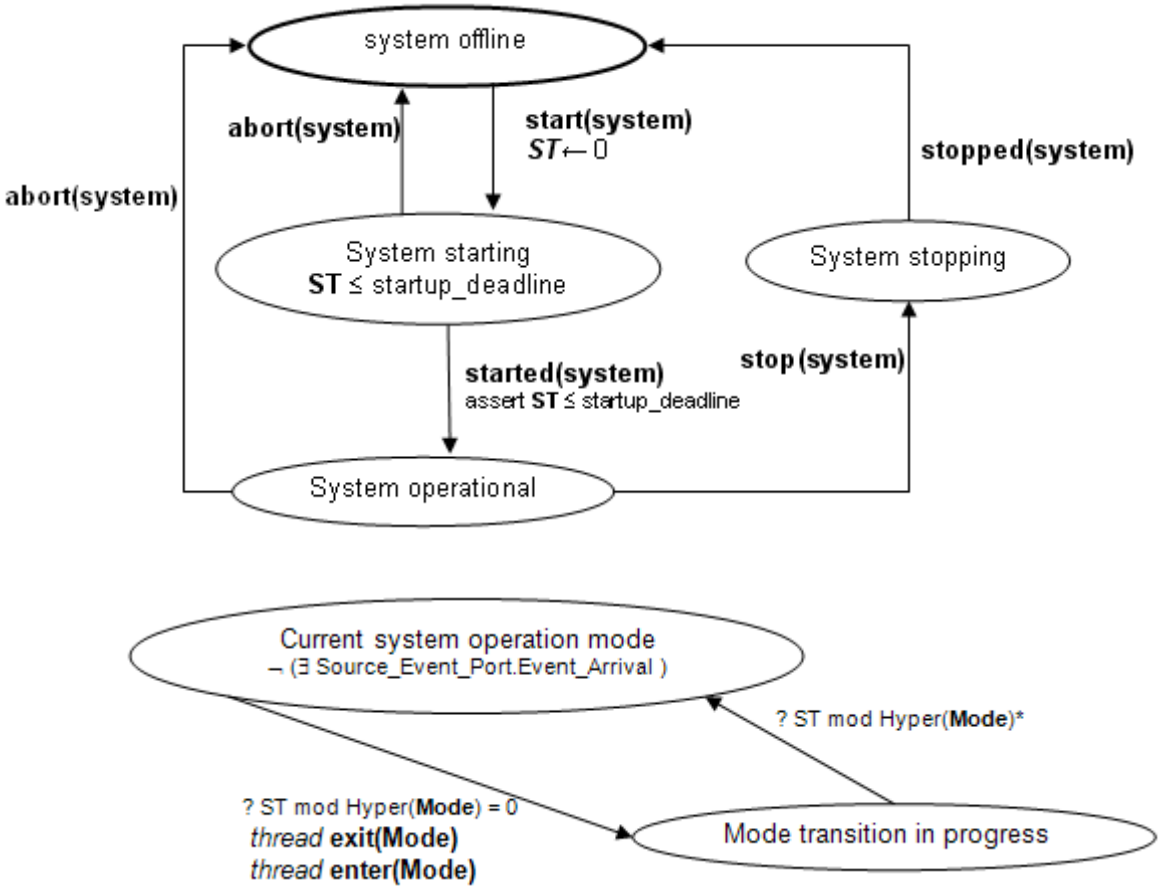
Bus initial handshaking?

How far do we want to go?





Automata



AADIv3

Part 4 – MoC

AADLv3 table of contents

Part 4 : Property sets & MoC configurations

Rationale: a MoC is a particular usage of properties e.g. Synchronous profile, mono core, multi core, etc. Goal is to document most common ones

Define relevant subsets, starting with AADLv2 property sets

Use configuration sets to compose them. Ideally, should be orthogonal