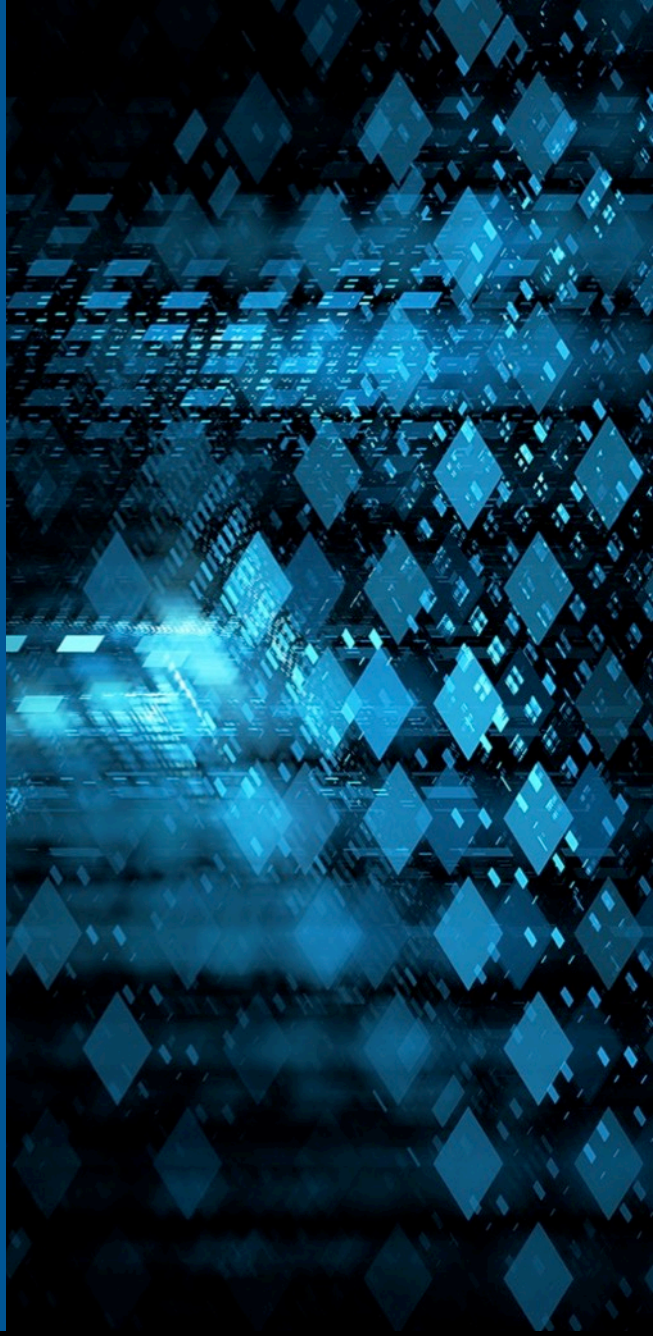
AADL: Bindings and Resources

Peter Feiler

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

Bindings between System Hierarchies

AADL supports a (primary) containment hierarchy

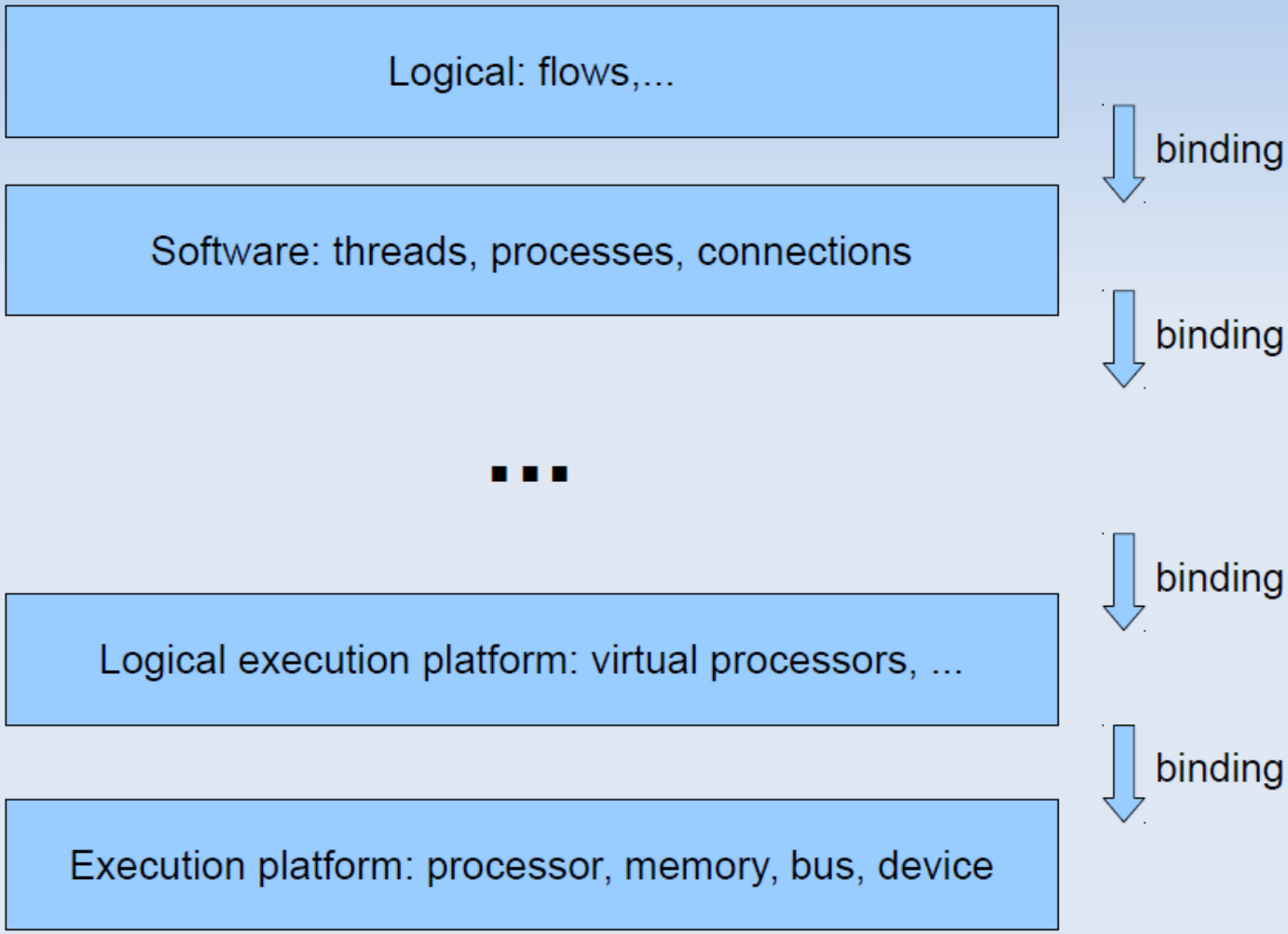
Semantic connections represent flow between and within subtrees

- Managed interaction complexity by requiring connections up and down the hierarchy to restrict arbitrary connectivity
- Note: for subprogram calls we offer both a connection and a mapping specification

Deployment bindings (aka. allocations) are a mapping from elements of one subtree to elements of another subtree

- The subtrees represent different virtual machine layers with the lower layer typically representing resources to the higher layer
- Bindings represent resource allocation

Multi Layer Architecture



Issues in Current Binding Approach

Bindings are currently expressed by properties

Binding related properties are not distinguishable from others

- Properties that express bindings
- Properties that relate to bindings

EMV2 propagation paths are derived from bindings

- binding points currently are identified by special keyword

Binding properties reach down the instance containment hierarchy

- A primary driver for introducing contained property associations

Resource Flow and Resource Allocation

Resource flow within an architecture: follows interface rules

- Continuous: Electricity, fluid flow, … Discrete: data samples, messages
- Directional flow with continuous characteristics: producer –> consumer
- Resource type represented as type (or abstract component type)
  - Annotations for discrete or continuous flow
- Fan out/in of flow “volume”
  - multiple features & multiple connections from one feature

Resource allocation/binding: Across different architecture layers

- Resource usage that needs to be allocated/scheduled
- SW to computer platform
- Logical to physical
- Resource capacity and demand as provides and requires features
  - Feature acts as binding point: resource type as classifier
  - Multiple

Binding Specification Proposal

Binding Types

Binding points

- Properties, constraints

Binding instances

- Specify source and target of binding

Binding Types

Define binding type

- User defined name, sets of source and target types

```
Binding ProcessorBinding : union( thread, thread_group, process) ->
union (  processor, virtual processor )
```

Predefined binding types

- *FunctionalBinding* to bind elements of a functional architecture to elements of a physical architecture.
- *ThreadBinding* to bind a thread to a processor, or to a virtual processor which in turn is the source of a processor binding to ultimately a processor.
- *CodeBinding* to bind source code associated with processes or threads to memory components.
- *DataBinding* to bind data components as well as source code data including stacks and heaps.
- *ConnectionBinding* to bind a connection to a flow sequence in hardware platform, or a virtual hardware platform whose elements are ultimately bound to a physical hardware platform.

Binding Instances

Deployment binding

- Binding source and target are components
  - Explicit binding type specification
  - Binding point identified by binding type

```
Configuration AS.deploymentconfig extends AS.systemconfig is
Binding1 : ThreadBinding binding Appsys.sub.proc.thread1 ->
platform.node.cpu1;
Binding2 : ProcessorBinding binding Appsys.sub.proc.thread2 ->
platform.node.cpu2;
```

Binding Point Specification

As **provides/requires binding** feature

- As abstraction for targets contained inside a system
- Similar to access features

One component can have multiple binding points

- Binding points of different types
  - Need/provision of different resources, e.g., at system level
- Binding points of same type
  - E.g., subsets of total resource capacity

Binding Point Specification

Explicit in features section:

- Directional features to be used as source or target of binding
- Identify type of binding
  - Binding type

```
thread task1 is
RequiredMemory: Requires binding Storage;
RequiredCycles: Requires binding Cycles;

System LinuxBox is
ProvidedDisk: Provides binding Storage;
ProvidedCycles: Provides binding Cycles;
end;

type Cycles : union (thread) -> union {processor, virtual processor };
```

Visibility of Binding Points

How far down can the allocation declaration reach

- Parameterized configuration as boundary for external use

Map binding point at configuration interface to component(s) in implementation that manage or represent resource

```
System ASplatform
ComputeCycles: provides binding #ProcessingCycles;
  Storage: provides binding #cache;
End ASPlatform;
ASplatform.boundconfig configures ASPlatform.impl {
ComputeCycles -> cpu,
Storage -> Cachememory
};
```

Connection Bindings

Currently: sequence of target elements

Connection acts as binding point

- Propagation identifies connection by name

```
connections
Conn1: port sub1.p1 -> sub2.p1 Requires XferBandwidth;
Conn2: abstract sub1.fe1 -> sub2.fe1 Requires WattsPerHour;
```

Platform flow sequence as binding target

- Expressed by end to end flow declaration
- Source and destination of ETE flow must match binding target of connection source and destination
- Each element of the flow has binding point of matching type

Virtual bus as binding target

- Virtual bus itself needs to be bound to a sequence of items => ETE flow

Flow sequence as closed platform configuration binding point

- How to expose platform internal ETEF as external binding point? => access to virtual bus that is bound to ETE flow

Qualified and Quantified Resources

Resource specification

- Via property on binding point or providing/requiring component
- Quantified via numeric type
- Qualified via enumeration type

```
thread task1
RequiredCycles: Requires binding Cycles {#Resources::ProcessingCycles => 200
MIPS;};
thread task2
RequiredInstructionSet: Requires binding IntelX86 {#Resources:: InstructionSet =>
X86;};


Processor IntelX86 is
  #Resources::ProcessingCycles => 1200 MIPS;
  #Resources:: InstructionSet => X86;

Property InstructionSet : enumeration ( X86, ARM, RISC) applies to Processor,
virtual processor, system, binding point;

Property ProcessingCycles : real units CycleUnits applies to Processor, virtual
processor, system, binding point;
```

Binding Type Multiplicity

1-to-n binding Alternatives (*):

- example – multiple cores

```
Binding MultiCoreBinding :
  union( thread, thread_group, system ) ->* union ( processor)
```

Bind to virtual processor (resource) or enclosing system:

- It schedules multiple cores