Analyzable Architectural Models of Service-Based Embedded Systems

Software Engineering Institute
Carnegie Mellon University
Pittsburgh, PA 15213

Presenter: John J. Hudak
Co-Authors: Lutz Wrage, Jorgen Hansson
Date: June 27, 2008

Motivation

SOA is emerging as a paradigm for achieving maximum reuse and minimum redundancy of services via complex, multi-platform distributed environments.

SOA's are being considered embedded real-time systems

Real-time embedded systems have the following QoS of interest:

- Performance goals (latency requirements, throughput)
- Dependability, fault tolerance
- Efficient use of resources (e.g. service, power)

How can SOA's be analyzed across multiple Quality of Service (QoS) attributes early and throughout the lifecycle to ensure quality attributes are achievable?

- Predictability through design

Outline

Themes: Architecture modeling, evaluation, Device Profile for Web Services (DPWS) as an example

This talk presents an model-base approach that analysis results

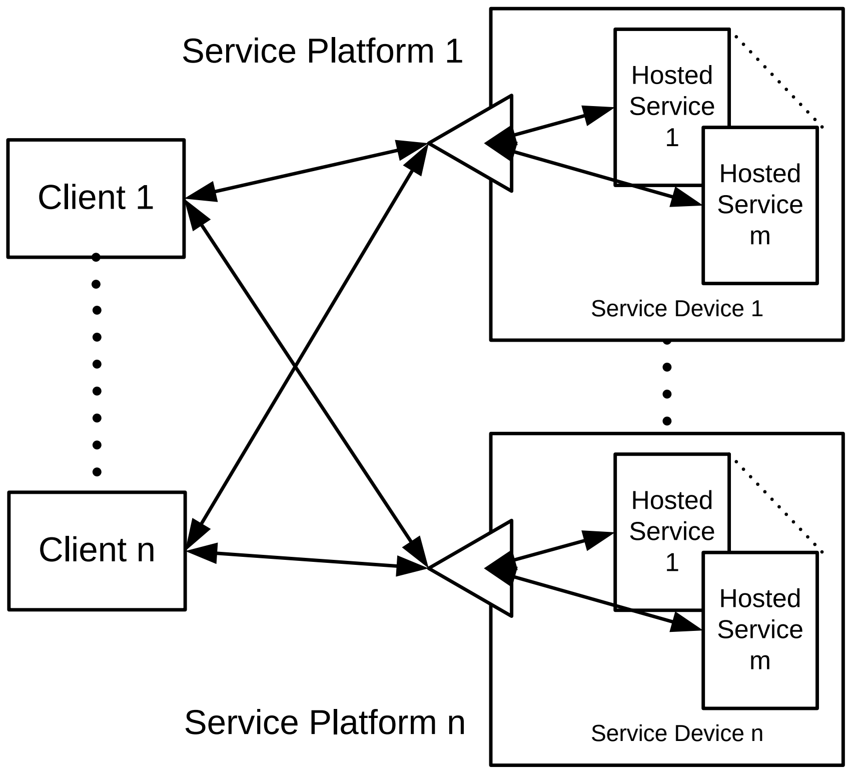
- Device Profile for Web Services – the concepts
- Architecture Analysis and Design Language (AADL) overview
- Modeling example: DPWS architecture – automobile navigation system
- Perspectives of DPWS modeling – logical communication, data flow, thread representation, cpu binding/resource utilization.

Device Profile for Web Services (DPWS)

DPWS specification that allows clients to discover services that when used together, can allow the client to perform a computational task.

DPWS enables:

- Description of web services
- Dynamic discovery of service
- Receiving and subscription to a (web) service
- Sending secure messages to/from a (web) service



Analysis Perspectives of SOA

Performance

- **Latency**
  - Service establishment
  - Timely response from services (hard, soft real-time)

Resource Consumption

- Communication bandwidth
- **CPU utilization – resource mapping**
- Power Consumption

Dependability

- Fault detection – enumerate fault type
- Redundancy – show replication, control mechanisms
- Modal operation – degraded operational and failover state

Common

Model

Approach

AADL Overview

The Architecture Analysis and Design Language (AADL)

It is an SAE Avionics Standard- AS-5506

A formal modeling language for describing software and hardware system architecture

Based on the component-connector paradigm

Textual and graphical notations that allows for:

- Precise execution semantics for modeling of hardware and software components & interactions
  - Thread, process, data, subprogram, system, processor, memory, bus, device, *abstract component, virtual processor, virtual bus*
- Continuous control & event response processing
  - Data and event flow, synchronous call/return, shared access
  - End-to-End flow specifications
- Operational modes & fault tolerant configurations
  - Modes & mode transition

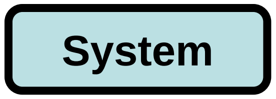
Core Standard (AS5506) published 2004, Annexes 2006, V2 currently being balloted

- www.aadl.info

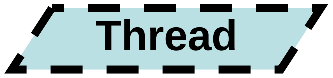
Overview of AADL Language Application Components

Components and their descriptions used in *our modeling*:

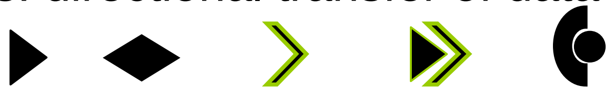
- System: hierarchical organization of components



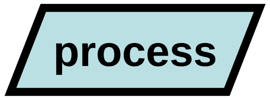
- Thread: a schedulable unit of concurrent execution



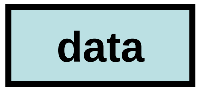
- Ports: directional transfer of data & control



- Process: protected address space



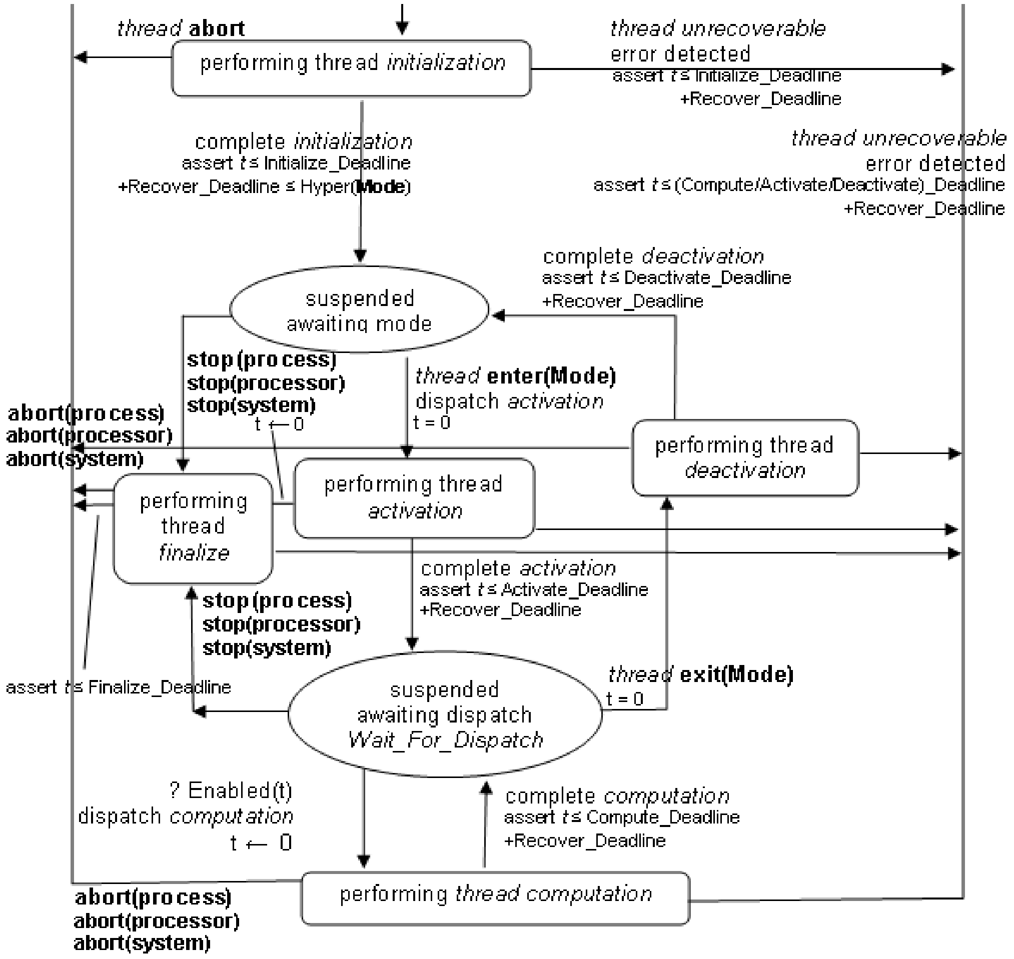
- Data: potentially sharable data



Well-Defined Architecture Execution Semantics

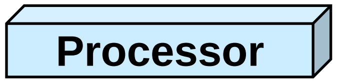
Thread Example – precise specification

- **Nominal execution entry point**
- **Fault handling entry point**
- **Resource locking**
- **Mode switching**
- **Initialization entry point**
- **Finalization exit point**

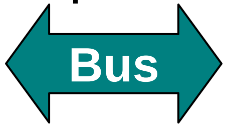

thread abort
performing thread initialization
thread unrecoverable error detected
assert t ≤ Initialize_Deadline +Recover_Deadline
complete initialization
assert t ≤ Initialize_Deadline
+Recover_Deadline ≤ Hyper(Mode)
thread unrecoverable error detected
assert t ≤ (Compute/Activate/Deactivate)_Deadline
+Recover_Deadline
complete deactivation
assert t ≤ Deactivate_Deadline
+Recover_Deadline
suspended awaiting mode
stop(process)
stop(processor)
stop(system)
t ← 0
thread enter(Mode)
dispatch activation
t = 0
abort(process)
abort(processor)
abort(system)
performing thread deactivation
performing thread finalize
performing thread activation
complete activation
assert t ≤ Activate_Deadline
+Recover_Deadline
stop(process)
stop(processor)
stop(system)
thread exit(Mode)
t = 0
assert t ≤ Finalize_Deadline
suspended awaiting dispatch Wait_For_Dispatch
? Enabled(t)
dispatch computation
t ← 0
complete computation
assert t ≤ Compute_Deadline
+Recover_Deadline
performing thread computation
abort(process)
abort(processor)
abort(system)

Overview of AADL language-Hardware Components

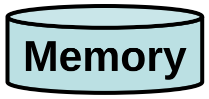
- Processor – provides thread scheduling and execution services



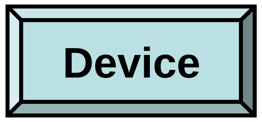
- Bus – provides physical connectivity between execution platform components



- Memory – provides storage for data and source code



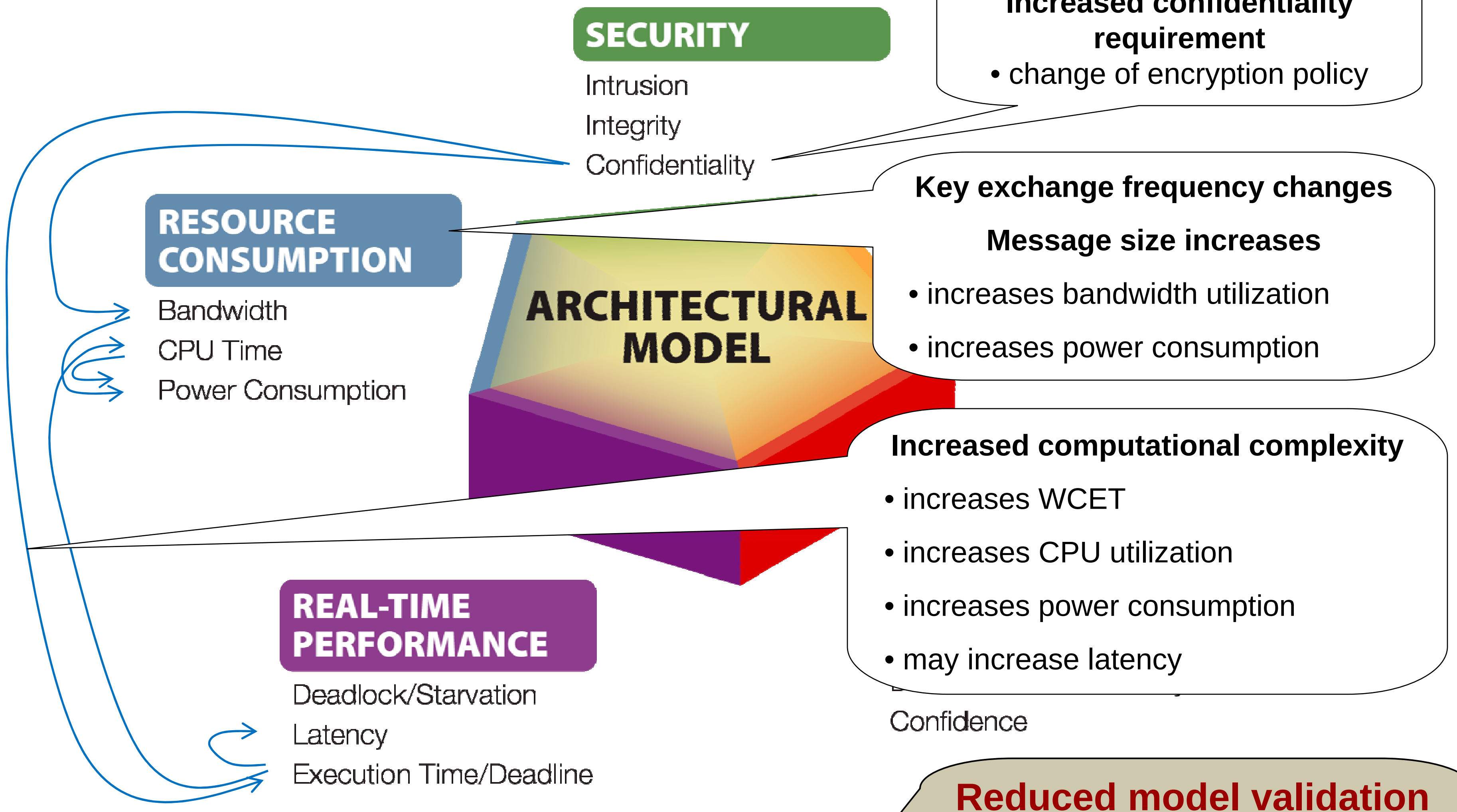
- Device – interface to external environment



Model-based Engineering

A holistic approach provides insight via architectural analysis

Model-based Engineering: *Single-Model, Multi-Dimensional Analysis*



Increased confidentiality requirement

- change of encryption policy

Key exchange frequency changes

Message size increases

- increases bandwidth utilization
- increases power consumption

Increased computational complexity

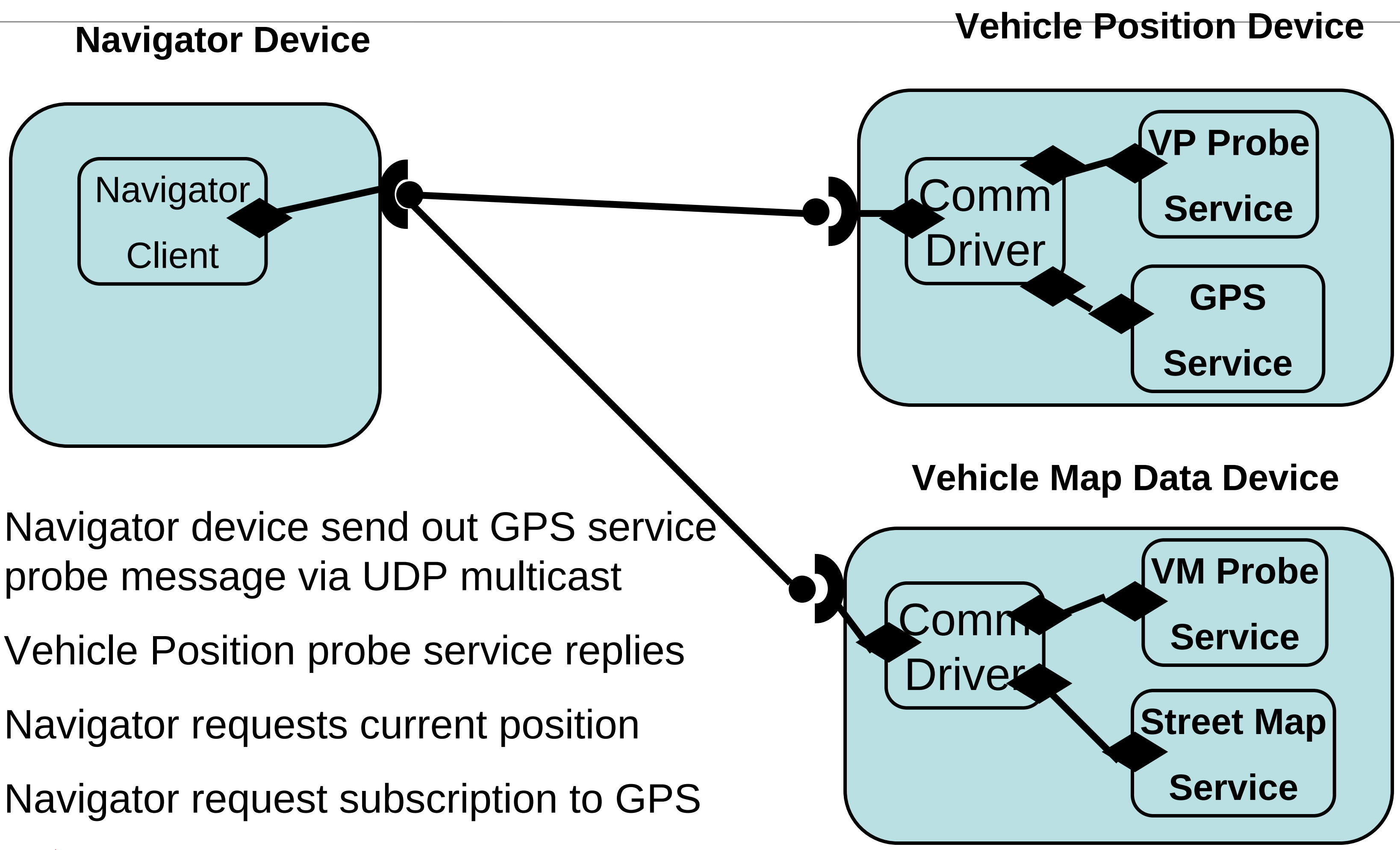
- increases WCET
- increases CPU utilization
- increases power consumption
- may increase latency

Reduced model validation cost due to single source model

High-Level Models

Insight into desired information flow

Vehicle Navigator SOA – Logical Data Paths



1. Navigator device send out GPS service probe message via UDP multicast
2. Vehicle Position probe service replies
3. Navigator requests current position
4. Navigator request subscription to GPS

➡ Establish Latency requirements

Latency Analysis

Establish the latency requirement (e.g. flow specification)

- Modeling the logical flow of data, latency property values

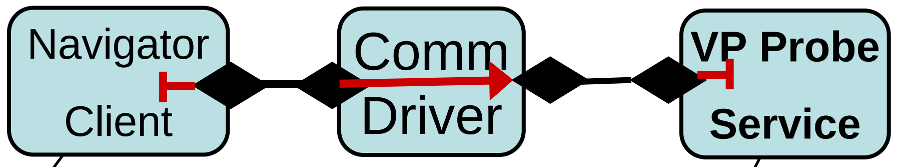
Determine actual latency of application components

- Measuring component execution time, actual latency property values

Perform latency analysis

- Summation of required latency values :: Summation actual latency values over specified flows

Flow Specification for DWPS



```
system nav_client
features
        veh_position: inout data port;
flows
        Nav_client_f1: flow source veh_position_probe{later
       {actual_latency 40ms};
end nav_client;
system VP_comm_drv
features
        nav_request: inout data port my_data;
        vp_request: inout data port my_data;
flows
       VP_comm_f1: flow path nav_request -> vp_request
        {latency:50 ms}, {actual_latency:40ms};
end VP_comm_drv;
system VP_Probe_Service
Features
         vp_request: inout data port float_type;
flows
         VP_probe_f1: flow sink VP_service_request
          {latency: 50 ms}, {actual_latency:40ms};
end VP_Probe_Service;
```

-Completeness of flow

-Hardware abstraction

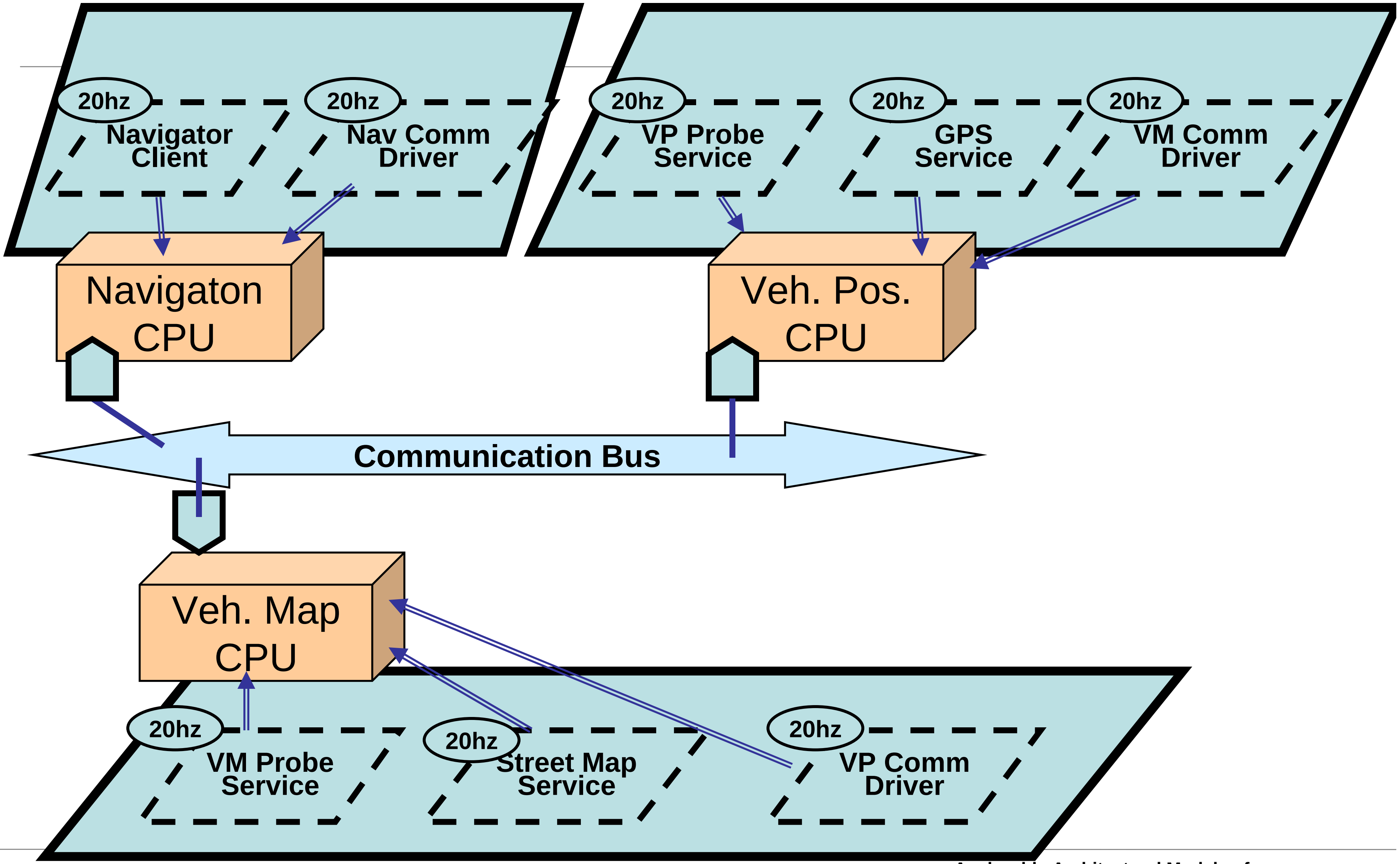
-Hierarchical decomposition

```
VP_probe_svc_flow: Nav_client_f1->C1->VP_comm_f1->C2->VP_probe_f1;{latency:50ms}
```

Execution Platform Mapping

Another perspective

Thread view


20hz
Navigator Client
20hz
Nav Comm Driver
20hz
VP Probe Service
20hz
GPS Service
20hz
VM Comm Driver
Navigaton CPU
Veh. Pos. CPU
Communication Bus
Veh. Map CPU
20hz
VM Probe Service
20hz
Street Map Service
20hz
VP Comm Driver

GPS Service Thread Model Example

```
thread GPS_service
features
  GPS_data: inout data port GPS_data.raw;

properties
  Dispatch_Protocol => Periodic;
  Period => 20 ms;
  Compute_Deadline => value(Period);
  Compute_Execution_Time => 20..20 ms;
  Initialize_Deadline => 10 ms;
  Initialize_Execution_Time => 1..1 ms;
end GPS_service;
```

Typed data ports

Dispatch execution properties

Value of other property

Minimum & maximum execution time

Analysis / Synthesis of Binding

Binding Decisions can be analyzed

- Do we have enough hardware capacity?
  - Structured properly?
- How is performance affected given different scheduling protocols?
  - `Supported_Scheduling_Protocols` **`=> type enumeration`**`( Fixed_Priority_Preemptive, <others> );`
  - `Scheduling_Protocols` **`=>`** `Fixed_Priority_Preemptive;`

Binding Decisions can be Automatically Optimized

- To minimize hardware needed
- Analyzed choices can be constrained
  - E.g. Due to Fault-tolerant replicas of failure independence
    - o `Not_Collocated =>` **`reference`** `replica2;`
  - Binary code compatibility
    - o `Allowed_Processor_Binding_Class => (`**`processor`**` powerpc);`

Summary

Architecture modeling can be used to create analyzable representations of certain SOA's

Incremental Refinement – Each abstraction can be refined

- Real-life development processes

Modeling Concerns of Software - System Integration

- Impact of Runtime architecture

For Analysis / Synthesis

- Multiple and extensible concerns

AADL + OSATE tools set provides a good 'first step' to design and analysis

- Open system, academic and industrial community support

Research issues to be explored -

- Scalability

Software Engineering Institute | Carnegie Mellon

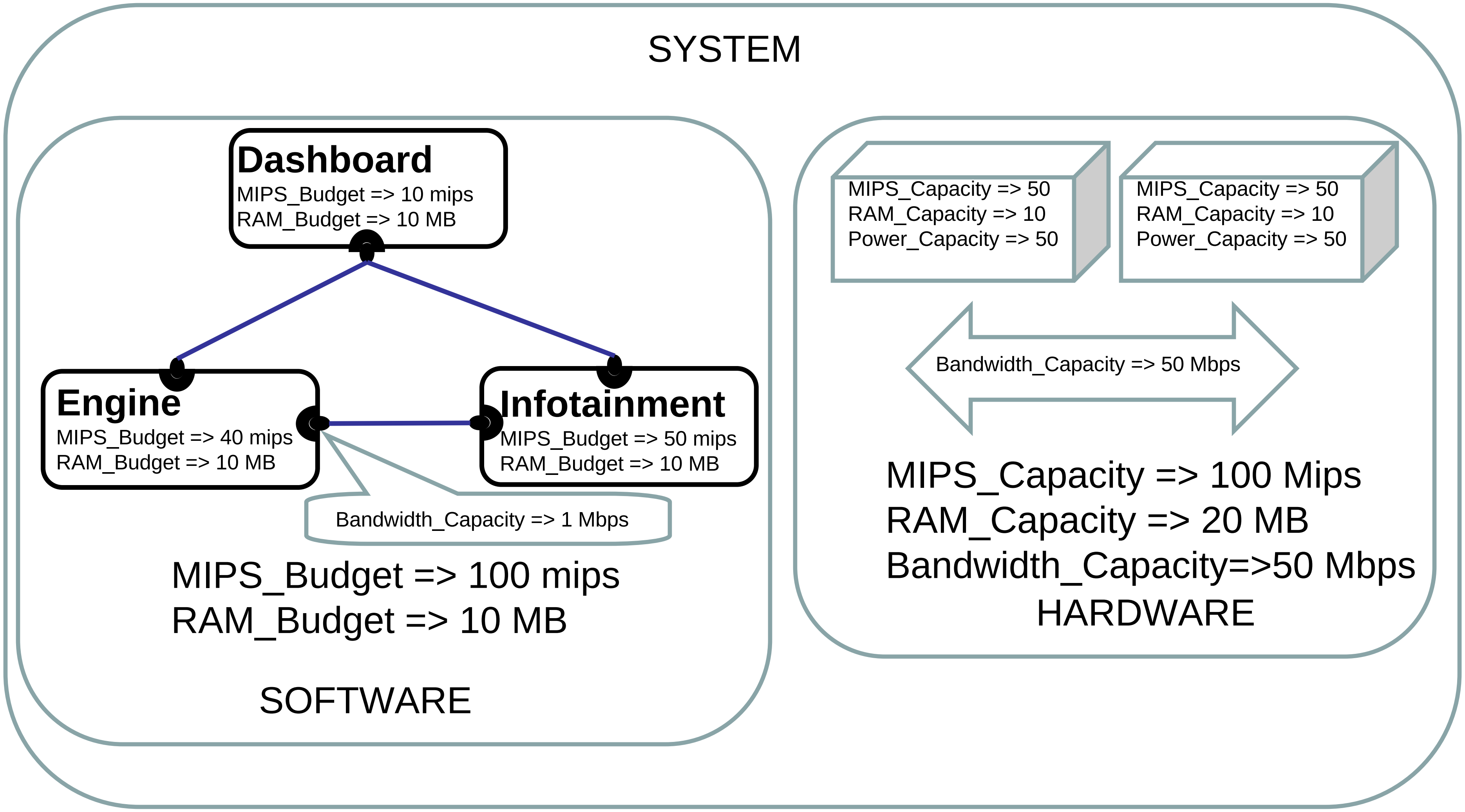
Incremental Refinement

Supports Development Process Decomposition

- Impact on Performance (Budgets)
  - Comparison of budgets and actuals
- Incremental addition of details
  - Without breaking previous decisions

At each step of refinement the model is analyzable

Incremental Refinement (Budgeting)


SYSTEM
Dashboard
MIPS_Budget => 10 mips
RAM_Budget => 10 MB
Engine
MIPS_Budget => 40 mips
RAM_Budget => 10 MB
Infotainment
MIPS_Budget => 50 mips
RAM_Budget => 10 MB
Bandwidth_Capacity => 1 Mbps
MIPS_Budget => 100 mips
RAM_Budget => 10 MB
SOFTWARE
MIPS_Capacity => 50
RAM_Capacity => 10
Power_Capacity => 50
MIPS_Capacity => 50
RAM_Capacity => 10
Power_Capacity => 50
Bandwidth_Capacity => 50 Mbps
MIPS_Capacity => 100 Mips
RAM_Capacity => 20 MB
Bandwidth_Capacity=>50 Mbps
HARDWARE