Metropolis Metro II: a design framework based on composition of heterogeneous models
Outline

• Background on Metropolis
  – Semantics
  – Modeling Architecture

• Metropolis II infrastructure
  – Import
  – Execution semantics
  – Applications
    • UMTS
    • Energy Efficient Building
Platform-based Design

- A platform consists of a set of services
  - Associated semantics
- Functionality
  - Uses services
- Architecture
  - Provides services at some cost
Metropolis Design Framework

- **Functionality**
  - What does it do?

- **Constraints**
  - How should it do it?

- **Architecture Platform**
  - How is it done?
  - At what cost?

- **Mapping**
  - Binding between the two

---

MetaModel language

Metamodel Compiler

Abstract syntax trees

Metropolis Interactive Shell

- Back end₁
- Back end₂
- ... Back endₙ

- Simulator tool
- Verification tool
- ... tool
Metropolis Objects

- Metropolis elements adhere to a “separation of concerns” point of view.

- **Processes (Computation)**
  - 
  - 

- **Media (Communication)**
  - 
  - 

- **Quantity Managers (Coordination)**
  - 
  -
Key Modeling Concepts

• An **event** is the fundamental concept in the framework
  – Represents a transition in the *action automata* of an object
  – An event is owned by the object that exports it
  – During simulation, generated events are termed as *event instances*
  – Events can be annotated with any number of quantities
  – Events can partially expose the state around them, constraints can then reference or influence this state

• A **service** corresponds to a set of **sequences of events**
  – All elements in the set have a common begin event and a common end event
  – A service may be parameterized with arguments

Action Automata

• Processes take *actions*.
  – statements and some expressions, e.g.
    \[ y = z + \text{port.f}();, \quad z + \text{port.f}(), \quad \text{port.f}(), \quad i < 10, \ldots \]
  – only calls to media functions are *observable actions*

• An *execution* of a given netlist is a sequence of vectors of *events*.
  – *event* : the beginning of an action, e.g. \( B(\text{port.f}()) \),
    the end of an action, e.g. \( E(\text{port.f}()) \), or null \( N \)
  – the \( i \)-th component of a vector is an event of the \( i \)-th process

• An execution is *legal* if
  – it satisfies all coordination constraints, and
  – it is accepted by all action automata.
Semantics summary

- Processes run sequential code concurrently, each at its own arbitrary pace.
- Progress may block at synchronization points
  - awaits
  - function calls and labels to which awaits or constraints refer.
- The legal behavior of a netlist is given by a set of sequences of event vectors.
  - multiple sequences reflect the non-determinism of the semantics:
    concurrency, synchronization (awaits and constraints)
An architecture component specifies *services*, i.e.

- what it *can* do
- how much it *costs*

**Nexperia™ Hardware Architecture**

- **MIPS**
  - MIPS CPU
  - Device I/O Block
  - Device I/O Block
  - Device I/O Block
  - DVP System Silicon

- **TriMedia**
  - TriMedia CPU
  - Device I/O Block
  - Device I/O Block
  - Device I/O Block

**Library of Device Blocks**
- Image coprocessors
- DSPs
- UART
- 1394
- USB

**VLIW Media Processor**
- 100 to 300+ MHz
- 32-bit or 64-bit

**Nexperia System Buses**
- 32-128 bit
Meta-model: architecture components

An architecture component specifies services, i.e.

- what it can do:
  - interfaces, methods, coordination (awaits, constraints), netlists
- how much it costs:
  - quantities, annotated with events, related over a set of events

```java
interface BusMasterService extends Port {
    update void busRead(String dest, int size);
    update void busWrite(String dest, int size);
}

medium Bus implements BusMasterService {
    port BusArbiterService Arb;
    port MemService Mem; ...
    update void busRead(String dest, int size) {
        if(dest== ... ) Mem.memRead(size);
    }
    ...
}```
Meta-model: architecture components

- This modeling mechanism is generic, independent of services and cost specified.
- Which levels of abstraction, what kind of quantities, what kind of cost constraints should be used to capture architecture components?
  - depends on applications

**Transaction:**
- Services:
  - fuzzy instruction set for SW, execute() for HW
  - bounded FIFO (point-to-point)
- Quantities:
  - #reads, #writes, token size, context switches

**Virtual BUS:**
- Services:
  - data decomposition/composition
  - address (internal v.s. external)
- Quantities: same as above, different weights

**Physical:**
- Services: full characterization
- Quantities: time
Outline

• Background on Metropolis
  – Semantics
  – Modeling Architecture

• Metropolis II infrastructure
  – Import
  – Execution semantics

• Applications
  • UMTS
  • Energy Efficient Building
Heterogeneous IP Import in Metropolis

- Excessive time spent in design import
  - Redefining and implementing classes and methods
  - Memory allocation, data types, templates, etc
- Challenges in Infineon case study
  - 802.11a on MuSIC (multiple SIMD core) architecture
Heterogeneous IP Import in Metro II

• Pros
  – Framework easier to develop and maintain
  – Leverage existing compilers/debuggers
  – Quicker import for most IP

• Cons
  – Framework has limited visibility
Components, Ports, and Connections

- IP is wrapped to expose framework-compatible interface
- Components encapsulate wrapped IP

- **Ports**
  - Coordination: provided, required
  - View ports

- **Connections**
  - Each method in interface for provided-required connection associated with begin and end events
Behavior-Performance Separation in Metropolis

- Processes make explicit requests for annotation
- Annotation/scheduling are intertwined
  - Iteration between multiple quantity managers
- Challenges in GM case study
  - Vehicle stability application on distributed CAN architecture
  - Interactions between global time QM and resource QM difficult to debug
Events

• An event is the fundamental concept in the framework

• Fields:
  – Process: Generator of event
  – Value Set: Variables exposed along with event
  – Tag Set: Quantity annotations

• Tagged Signal Model*

\[ E = \langle p, V, T \rangle \]

3 Phase Execution

1. Base
   - Each process proposes events and suspends
   - Multiple events can be proposed simultaneously by one process

2. Annotation
   - Tag proposed events with quantities

3. Scheduling
   - Rejection of some proposed events
   - At most 1 enabled event per process
Phases and Events

- Each phase is allowed to interact with events in a limited way
  - Keep responsibilities separate

<table>
<thead>
<tr>
<th>Phase</th>
<th>Events</th>
<th>Tags</th>
<th>Values</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Propose</td>
<td>Disable</td>
<td>Read</td>
</tr>
<tr>
<td>Base</td>
<td>Yes</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Annotation</td>
<td></td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>Scheduling</td>
<td></td>
<td>Yes</td>
<td>Yes</td>
</tr>
</tbody>
</table>
Behavior-Performance Separation in Metro II

• Pros
  – Phase 1 objects no longer explicitly request annotation
  – Separation of quantity managers into annotators and schedulers
    • “Global time” separates into physical time (annotation) and logical time (scheduling)

• Cons (?)
  – Additional phase introduced into execution model
Mappers

- Mappers are objects that help specify the mapping
  - Bridge syntactic gaps only

- Mapping occurs at the component level
  - Between components with compatible interfaces
  - Possibly many functional components mapped to a single architectural component
Summary: Features for Metro II

- Import heterogeneous IP
  - Different languages
  - Different models of computation
- Behavior-Performance Separation
  - No explicit requests for annotation
  - Annotation separated from scheduling
- Operational/Denotational Separation
  - Restricted access to events and values
  - Mapping carried out at component level
Outline

• Background on Metropolis
  – Semantics
  – Modeling Architecture
• Metropolis II infrastructure
  – Import
  – Execution semantics
• Applications
  • UMTS
  • Energy Efficient Building
Design Activity: UMTS Case Study

- **UMTS is a mobile communication protocol standard**
  - Universal Mobile Telecommunications System
  - 3G cell phone technology
  - Often used in Software Defined Radio (SDR)

- **Started with C and SystemC models as baseline**
  - Source of Metro II functional models
  - Profiling to use in architecture models
  - Comparisons for Metro II simulation results

- **Have both DLL and PHY level SystemC models**
  - Converted only data link layer to Metro II
Metro II UMTS Models

Focused on the DLL layer

Initial SystemC model was converted to Metro II

Two Models:
- Pure functional model with blocking read and write semantics.
- Timed model with a scheduler and preemption.

Timer communicates with scheduler to keep track of the passage of time. Components run in “zero time” (when given permission) but are told to post their output data at times which reflect their profiled execution time.
Synchronization Mechanisms

UMTS example exposed two approaches to synchronization in Metro II:

Explicit Synchronization:
- Use the underlying simulation framework directly, i.e. SystemC “or/and” waits.

Constraints:
- Move synchronization from phase 1 to phase 3 completely.

Option #1:
- Imperative code written to enforce operational semantics.

Option #2:
- Declarative constraints enforce semantics.
Metro II Mapping Semantics

Mapping allows both functional and architectural progress “simultaneously” via rendezvous type constraints.
Metro II: Service Modeling

- Two basic architecture modeling styles: cycle accurate runtime analysis vs. off line, pre-profiled approach.
Architecture Model Overview

Tasks for mapping 1-to-1 with functional components

RTOS for scheduling events from N tasks to M processing elements

Three scheduling policies:
- Round Robin
- Fixed Priority
- FCFS

Numerous configurations of processing elements (48 chosen)
Metro II Complete System

Phase 1

- FC1
- M1
- T1
- Sparc1

- FCN
- M2
- T2
- ARM7
- OS
- uB

Phase 2

- MNN
- TN

- Annotator

Phase 3

- Mapping Constraint Solver

- Logical Time Scheduler
UMTS Case Study Outcome

- Processing elements include
  - ARM7, ARM9: GPP profiling approach
  - Microblaze: Programmable platform flow
  - SPARC: runtime profiling with C code snippets of core routines

- 48 different mappings explored
  - 11 PEs, 1 PE, combinations of 4 PEs broken down by RLC tx/rx and MAC tx/rx
  - 9 classes within the 48 mappings
## 48 Mappings

<table>
<thead>
<tr>
<th>#</th>
<th>Type</th>
<th>Partition</th>
<th>#</th>
<th>Type</th>
<th>Partition</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>RTP</td>
<td>11 Sp</td>
<td>17</td>
<td>PP</td>
<td>2 μB (2), 2 A9 (3)</td>
</tr>
<tr>
<td>2</td>
<td>PP</td>
<td>11 μB</td>
<td>18</td>
<td>PP</td>
<td>2 A9 (2), 2 μB (3)</td>
</tr>
<tr>
<td>3</td>
<td>PP</td>
<td>11 A7</td>
<td>19</td>
<td>PP</td>
<td>2 A7 (2), 2 A9 (3)</td>
</tr>
<tr>
<td>4</td>
<td>PP</td>
<td>11 A9</td>
<td>20</td>
<td>PP</td>
<td>2 A9 (2), 2 A7 (3)</td>
</tr>
<tr>
<td>5</td>
<td>RTP</td>
<td>4 Sp (1)</td>
<td>21</td>
<td>MIX</td>
<td>Sp (4), μB (5), A7 (6), A9 (7)</td>
</tr>
<tr>
<td>6</td>
<td>PP</td>
<td>4 μB (1)</td>
<td>22</td>
<td>MIX</td>
<td>Sp (4), μB (5), A9 (6), A7 (7)</td>
</tr>
<tr>
<td>7</td>
<td>PP</td>
<td>4 A7 (1)</td>
<td>23</td>
<td>MIX</td>
<td>Sp (4), A7 (5), μB (6), A9 (7)</td>
</tr>
<tr>
<td>8</td>
<td>PP</td>
<td>4 A9 (1)</td>
<td>24</td>
<td>MIX</td>
<td>Sp (4), A7 (5), A9 (6), μB(7)</td>
</tr>
<tr>
<td>9</td>
<td>MIX</td>
<td>2 Sp (2), 2 μB (3)</td>
<td>25</td>
<td>MIX</td>
<td>Sp (4), A9 (5), A7 (6), μB(7)</td>
</tr>
<tr>
<td>10</td>
<td>MIX</td>
<td>2 μB (2), 2 Sp (3)</td>
<td>26</td>
<td>MIX</td>
<td>Sp (4), A9 (5), μB (6), A7 (7)</td>
</tr>
<tr>
<td>11</td>
<td>MIX</td>
<td>2 Sp (2), 2 A7 (3)</td>
<td>27</td>
<td>MIX</td>
<td>μB (4), Sp (5), A7 (6), A9 (7)</td>
</tr>
<tr>
<td>12</td>
<td>MIX</td>
<td>2 A7 (2), 2 Sp (3)</td>
<td>28</td>
<td>MIX</td>
<td>μB (4), Sp (5), A9 (6), A7 (7)</td>
</tr>
<tr>
<td>13</td>
<td>MIX</td>
<td>2 Sp (2), 2 A9 (3)</td>
<td>29</td>
<td>MIX</td>
<td>μB (4), A7 (5), Sp (6), A9 (7)</td>
</tr>
<tr>
<td>14</td>
<td>MIX</td>
<td>2 A9 (2), 2 Sp (3)</td>
<td>30</td>
<td>MIX</td>
<td>μB (4), A7 (5), A9 (6), Sp (7)</td>
</tr>
<tr>
<td>15</td>
<td>PP</td>
<td>2 μB (2), 2 A7 (3)</td>
<td>31</td>
<td>MIX</td>
<td>μB (4), A9 (5), A7 (6), Sp (7)</td>
</tr>
<tr>
<td>16</td>
<td>PP</td>
<td>2 A7 (2), 2 μB (3)</td>
<td>32</td>
<td>MIX</td>
<td>μB (4), A9 (5), Sp (6), A7 (7)</td>
</tr>
</tbody>
</table>

(1 = Rx MAC, Tx MAC, Rx RLC, Tx RLC), (2 = Rx MAC, Rx RLC), (3 = Tx MAC, Tx RLC), (4 = Rx MAC), (5)(Rx RLC), (6)(Tx MAC), (7 = Tx RLC) (Sp = Sparc, μB = Microblaze, A7 = ARM7, A9 = ARM9)
Execution Time and Utilization Analysis

- **Round Robin**
  - Mapping #1 (fastest, 11 SPARC#s) and #46 (slowest, 1 uBlaze) had a 2,167% difference

- **Priority**
  - Avg. execution time reduced by 13% over round robin
  - Avg. utilization decreases by 2%

- **FCFS**
  - Avg. execution time reduced by 7%
  - Avg. utilization increases by 27%
SystemC vs. Metro II

- Metro II timed functional model has a 7.4% increase in runtime over SystemC timed functional model
- Mapped Metro II model is 54.8% faster than timed SystemC model
  - Metro II phases 2 and 3 have significantly less overhead than the timer-and-scheduler based system required by the SystemC timed functional model
- In a comparison of the Metro II timed model running without constraints and one running with them, the average runtime decrease was 25%
Design Effort

• Entire design
  – 85 files
  – 8,300 LOC

• Mapping change affects only 2 files

• Metro II conversion affects 1% of lines in each file
  – 58% of these lines relate to constraint registration

• SystemC SPARC model conversion adds only 3.4% to code size (92 lines)
On-Chip Communication

Core graph + constraints

Components and models

- Synthesis
- Implementation
- Refinement and SystemC generation

Building automation networks

- Synthesis
- Implementation
- Network load
- Network cost (20yr)
- Latency and slack

<table>
<thead>
<tr>
<th>N</th>
<th>Area (um²)</th>
<th>Power (mW)</th>
<th>N-hops</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>15.6</td>
<td>8.5</td>
<td>2.2</td>
</tr>
<tr>
<td>4</td>
<td>22.6</td>
<td>4.7</td>
<td>3.1</td>
</tr>
<tr>
<td>3</td>
<td>10.9</td>
<td>1.9</td>
<td>3.1</td>
</tr>
<tr>
<td>4</td>
<td>30.1</td>
<td>13.4</td>
<td>4.6</td>
</tr>
<tr>
<td>5</td>
<td>25.1</td>
<td>10.9</td>
<td>4.1</td>
</tr>
<tr>
<td>4</td>
<td>56.0</td>
<td>18.9</td>
<td>0.4</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Vcc</th>
<th>VDD (V)</th>
<th>1.2V</th>
<th>0.9V</th>
<th>0.6V</th>
</tr>
</thead>
<tbody>
<tr>
<td>VCC (V)</td>
<td>1.8</td>
<td>1.8</td>
<td>1.8</td>
<td></td>
</tr>
<tr>
<td>VDD (V)</td>
<td>0.8</td>
<td>0.8</td>
<td>0.8</td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>4.75</td>
<td>4.75</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Np (mW/mm²)</td>
<td>0.48</td>
<td>0.48</td>
<td>0.48</td>
<td></td>
</tr>
<tr>
<td>Pp (mW/mm²)</td>
<td>1.8</td>
<td>1.8</td>
<td>1.8</td>
<td></td>
</tr>
</tbody>
</table>
Building automation

\[ t = 1 \nu, \; l = 0.5 \nu, \; b = 18, \; p = 10 \]

**Synthesis**

Implementation

Network load
Network cost (20yr)
Latency and slack
Heating and Cooling in Building Automation

Source (Temp and Pressure constant)

Room 1 (P,T)

Environment (P,T)

Sink

Door (Crack Model)

Room 2 (P,T)
Heating and Cooling Functional Model

Modelica Model

CORBA communication

OpenModelica

MetroII

Controller1

Controller2

Room Temperatures

A1 -> FIFO_a1c -> Controller1

A2 -> FIFO_a2c -> Controller2

S1 -> FIFO_s1c

S2 -> FIFO_s2c
COSI and Metro II: Design Flow

Step 1
Metro II Functional Model

Mapping

Step 2
Metro II Simulation Results

COSI Synthesis results

Step 3
Refined Metro II Architecture Model

COSI
Metro II (UCB) to ASPN (UCLA) Flow Overview

1. Develop a small library of computational and communication components which we can characterize quickly.

2. Create an architecture model and map it to existing functional model. H.264 is leading candidate.

3. Extract computation time and communication volume for the mapped model.


5. Feed ASPN to UCLA

New architecture topology

- MicroBlaze
- Sparc
- FSL
- FIFO Memory

Metro II Component Arch Library

Functional Model (e.g. H.264)

Liangpeng Guo (UCB)
Douglas Densmore (UCB)
Qi Zhu (UCB)
Karthik Gururaj (UCLA)

ASPN (XML Description)

TASK_GRAPH <id> <name>

TASK_NUMBER <tasks>

TASK <id> <name>

WORST_CASE_CYCLES

<processor_type> <integer>

WORST_CASE_CYCLES

<processor_type> <integer>

END_TASK_GRAPH

EDGE <id> <name>

PREDECESSOR_TASK <id>

SUCCESSOR_TASK <id>

COMMUNICATION_VOLUME <cycles>

END_CLUSTER

CLUSTER <id>

PROCESSOR <id>

<type>

TASK <id>

TASK <id>

TASK <id>

END_CLUSTER

END_STAGE

//All stages done

FIFO_EDGE <id>

PREDECESSOR_CLUSTER <id>

SUCCESSOR_CLUSTER <id>

FIFO_SIZE <size>

UCLA