Bridging Functional and Architectural Aspects of Controller Design in Cyber-Physical System

Liangpeng Guo

EECS Department, UC Berkeley

6/7/2013
Gap between Function and Architecture

• Function: describe the functionality of the design
  – Behaviors that the system should implement

• Architecture: provides various platform targets (configurations of resources that implement certain functions)

• The functionality is modeled on high abstraction level with heterogeneous MoCs
  – High level abstraction without implementation details

Correctness of the CPS is sensitive to the...
A Functional Model without Implementation Details

DE Director

Generator Regulator with Overvoltage Protection

stopTime: 60.0

overVoltageThreshold: 119.0

SingleEvent

time: 15.0

At time 15, turn on a load.

Supervisor

MicrostepDelay

GeneratorContactorLoad

LoadImpedance: 1.0
voltageSamplePeriod: 1.0
generatorTimeConstant: 5.0
generatorOutputImpedance: 1.0

DesiredVoltage
value: 110.0

AddSubtract

PID

Kp: 2.0
Ki: 1.0

Expression

voltage >= overVoltageThreshold

TimedPlotter

legend: Voltage, Drive

_flipPortsHorizontal: true
Functional Behaviors are Affected by Implementation Details
Need New Methods & Tools for Bridging Functional & Architectural Aspects

• Approach 1: refine functional model by adding implementation details
  – E.g. add bus model
• Potential problems:
  – The model becomes complicated to understand and modify
  – Environment/language for modeling function is not suitable for architecture.
  – Semantic restriction: difficult to model complex architecture in functional environment

• Approach 2: construct the function using architectural primitives
  – E.g. programming language
• Potential problems:
Need New Methods & Tools for Bridging Functional & Architectural Aspects

• Approach 3: make assumptions (contracts) that decouple the function and architecture
  – E.g. assume a bounded communication delay

• Potential problems:
  – Making ‘appropriate’ assumptions is hard
  – Assumptions of the functional model about the architecture significantly impact the Software/Hardware implementation
  – Inappropriate assumptions may restrict the design choices and lead to costly or infeasible architecture
Aspect-Oriented Modeling

Joint modeling of implementation architecture and functional design to enable effective architecture exploration and assessment of the behavioral consequences of architectural choices.

– What is the performance? How this modifies the behavior? Does this satisfy the vertical contract?

– Investigate different design choices for the same functional model
Aspect-Oriented Modeling

- Map functional and architectural models without significant changes
- Enable co-simulation and performance estimation of the mapped model
- Provide interfaces to explore design and performance trade-offs

Simulation of Mapped Model

Functional Model

Architectural Candidates

tasks, signals

Sensors, Processors, Messages on buses
Example 1: Simplified BCU Controller

- Decide the power sources for AC buses
  - Input: fault signals of power sources and contactors
  - Output: control signals for contactors
  - Constraint: power sources are never paralleled

Generate control signals for contactors

Decide the power source of the left bus

Decide the power source of the right bus
Functional Model in Ptolemy

- Model each part as a FSM
- Compose FSMs into a Synchronous Reactive (SR) Model
- SR Director is modified to make the model mappable
Map to Arch Model in SystemC

Scheduler

Task1
Notify Metro II events

Task2
Notify SystemC events

Task3
Notify systemC events

Named Pipess

Propose Metro II events for each task in Ptolemy II

SystemC Architectural Model

Ptolemy II Functional Model

Arrange LeftPath

Arrange RightPath

Control SignalGen

in

out
Preliminary Co-simulation Results

- **Candidate 1**
  - High-speed single processor

- **Candidate 2**
  - Slow-speed dual processors
  - Low synchronization overhead

- **Candidate 3**
  - Medium speed dual processors
  - High synchronization overhead

<table>
<thead>
<tr>
<th>Parameters</th>
<th>Candidate</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>1</td>
</tr>
<tr>
<td>Scheduling overhead (ns)</td>
<td>10</td>
</tr>
<tr>
<td>Execution Time (ns)</td>
<td></td>
</tr>
<tr>
<td>ALP</td>
<td>40</td>
</tr>
<tr>
<td>ARP</td>
<td>45</td>
</tr>
<tr>
<td>CSG</td>
<td>25</td>
</tr>
<tr>
<td>Synchronization overhead (ns)</td>
<td>0</td>
</tr>
<tr>
<td>Parallelization of ALP and ARP</td>
<td>No</td>
</tr>
<tr>
<td>Total execution time (ns)</td>
<td>775</td>
</tr>
</tbody>
</table>

ALP: left controller
ARP: right controller
CSG: control signal generator
Example 2: printing press controller

Physical part: Continuous Models

Electrical part: Controllers

```cpp
#include <iostream>
#include <systemc.h>
#include "writer.h"
#include "Task.h"
#include "metroII.h"

using namespace std;

int sc_main(int argc, char* argv[]) {
    m2_params params;
    params.manager = m2_params::EXTERNAL_MANAGER;
    params.temp_path = "$METRO_TEMP/";
    params.debug_level = 2;

    m2_init(params);

    Task task1("Task1");

    m2_start();

    cout << "simulation completed." << endl;
    return 0;
}
The controller is a PID controller that minimizes the error between the target velocity profile and the measured velocity. The register actor is required since the sensor sampling is not necessarily at the same frequency as the frequency of receipt of profile information.
Mapping Configuration

Task 0

Task 1

Task 2
Round-Robin Scheduling

• Architectural model
  – Designed in a language and environment that are comfortable for architecture designer

• Example 1: RR scheduling

```c
Task* OS::next_task_to_run() {
    if (_ready_task_list.empty()) {
        return NULL;
    }

    Task* next_task = _ready_task_list.front();
    _ready_task_list.pop_front();
    return next_task;
}
```

| 500 ms: Task0.Register begins |
| 500005 us: Task0.Register ends |
| 500005 us: Task2.Register begins |
| 500010 us: Task2.Register ends |
| 500010 us: Task0.AddSubtract begins |
| 500012 us: Task0.AddSubtract ends |
| 500012 us: Task2.AddSubtract begins |
| 500014 us: Task2.AddSubtract ends |
| 500014 us: Task0.PID begins |
| 500029 us: Task0.PID ends |
| 500029 us: Task2.Register begins |
| 500034 us: Task2.Register ends |
| 500034 us: Task2.AddSubtract begins |
| 500036 us: Task2.AddSubtract ends |
| 500036 us: Task2.PID begins |
| 500051 us: Task2.PID ends |
| 500051 us: Task2.Const begins |
| 500053 us: Task2.Const ends |

| 800 ms: Task1.AddSubtract begins |
| 800002 us: Task1.AddSubtract ends |
| 800002 us: Task2.Register begins |
| 800007 us: Task2.Register ends |
| 800007 us: Task0.Register begins |
| 800012 us: Task0.Register ends |
| 800012 us: Task1.PID begins |
| 800027 us: Task1.PID ends |
| 800100 us: Task2.Register begins |
| 800105 us: Task2.Register ends |
Priority-based Scheduling

• Example 1: priority-based scheduling

```cpp
Task* OS::next_task_to_run() {
  if (_ready_task_list.empty()) {
    return NULL;
  }
  // Task* next_task = _ready_task_list.front();
  // _ready_task_list.pop_front();

  list<Task*>::iterator it_highest_priority = _ready_task_list.begin();
  for (list<Task*>::iterator it = _ready_task_list.begin();
      it != _ready_task_list.end();
      ++it) {
    if ((*it)->priority() > (*it_highest_priority)->priority()) {
      it_highest_priority = it;
    }
  }
  Task* next_task = *it_highest_priority;
  _ready_task_list.erase(it_highest_priority);
  return next_task;
}
```
Performance Annotation

- The basic performance annotation can be customized in a csv file.
Summary

• A simulation-based architecture exploration framework
  – Separate function/architecture models
  – Change mapping without significantly modifying models
  – Evaluate performance,
  – Explore architectural candidates and ‘discover’ new architectures
THANK YOU