APECS: An AADL and Polychrony based Embedded Computing System Design Environment with an Elevator Control Case Study

Matthew Anderson and Sandeep K. Shukla
Virginia Tech.

Partially funded by US Air Force Research Labs
Problem and Motivation

Approach

Current Project Status

Related Work

Conclusion

Outline of the talk

1. Problem and Motivation
2. Approach
3. Current Project Status
4. Related Work
5. Conclusion
Presentation Outline

1. Problem and Motivation
2. Approach
3. Current Project Status
4. Related Work
5. Conclusion
Architecture Analysis and Design Language (AADL)

SAE International standard and framework for model based software systems
Architecture Analysis and Design Language (AADL)

**SAE International standard and framework for model based software systems**

AADL and its accompanying tool suites provide a modeling environment that allow for modeling:

- Target hardware platforms
- Complete software hierarchies
- Static and Dynamic system properties
  - Real-Time Schedulability
  - Resource Utilization
Ocarina

An AADL tool suite developed in Ada to support building Distributed Real-Time Embedded Systems
Ocarina

An AADL tool suite developed in Ada to support building Distributed Real-Time Embedded Systems

- Syntactic and Semantic Analysis
- Automated Code Generation
- Supports a variety of software languages
  - C/C++
  - Ada
  - Lustre and Esterel
Ocarina

An AADL tool suite developed in Ada to support building Distributed Real-Time Embedded Systems

- Syntactic and Semantic Analysis
- Automated Code Generation
- Supports a variety of software languages
  - C/C++
  - Ada
  - Lustre and Esterel

So, what’s missing?
Motivations for Polychrony

- A lack of formal semantics in C/Ada make verification difficult
Motivations for Polychrony

- A lack of formal semantics in C/Ada make verification difficult
- Lustre and Esterel have fully synchronous specifications.
Motivations for Polychrony

- A lack of formal semantics in C/Ada make verification difficult.
- Lustre and Esterel have fully synchronous specifications.
- Software specifications still require a fine granularity, at the thread or function level.
Motivations for Polychrony

- A lack of formal semantics in C/Ada make verification difficult.
- Lustre and Esterel have fully synchronous specifications.
- Software specifications still require a fine granularity, at the thread or function level.
- Thread synchronization
  - Oversynchronization
  - Deadlock
Motivations for Polychrony

- A lack of formal semantics in C/Ada make verification difficult.
- Lustre and Esterel have fully synchronous specifications.
- Software specifications still require a fine granularity, at the thread or function level.
- Thread synchronization
  - Oversynchronization
  - Deadlock

What can be done?

→ Extend the backend of Ocarina with support for polychronous specifications from MRICDF sources.
Presentation Outline

1. Problem and Motivation
2. Approach
3. Current Project Status
4. Related Work
5. Conclusion
Elevator Design Study

APECS Design Approach

Top-down system model design illustrated using an Elevator Case Study
APECS Design Approach

Top-down system model design illustrated using an Elevator Case Study

- Physical Requirements
  - Centralized Control
  - Bank of four elevators
  - Service five floors
Elevator Design Study

APECS Design Approach

Top-down system model design illustrated using an Elevator Case Study

- Physical Requirements
  - Centralized Control
  - Bank of four elevators
  - Service five floors

- Safety and Behavior
  - Timely Operation
  - Safe door operation
  - Fire emergency response
Top-Level Systems

- Control Process
- Central Controller (CPU)
- RAM
- Floor 1
- Floor 2
- Floor 3
- Floor 4
- Elevator Bank
- E1
- E2
- E3
- E4

Matthew Anderson and Sandeep K. Shukla

AADL and Polychrony based Embedded Computing System Design Environment
Refining Systems
Refining Systems Cont’d
Door Behavior

- **State Table**

<table>
<thead>
<tr>
<th>Event</th>
<th>Conditions</th>
<th>Outputs</th>
</tr>
</thead>
<tbody>
<tr>
<td>E1</td>
<td>OpenRequest</td>
<td>Motor &lt;= open and Timer &lt;= start</td>
</tr>
<tr>
<td>E2</td>
<td>Obstruction</td>
<td>Motor &lt;= open and Timer &lt;= start</td>
</tr>
<tr>
<td>E3</td>
<td>CloseRequest</td>
<td>Motor &lt;= close</td>
</tr>
<tr>
<td>E4</td>
<td>Timeout</td>
<td>Motor &lt;= close</td>
</tr>
</tbody>
</table>

- **State Machine**
MRICDF Door Control Process
Clock Tree

```
  /
 /\  
/  \ /  
\ /\ \ /
 X1 [X1] X2 [X2]
  \  \  /
   \ /\ /
    X X2
```

```
  /
 /\  
/  \ /  
\ /\ \ /
 O [O] O [O]
  \  \  /
   \ /\ /
    O O
```

```
  /
 /\  
/  \ /  
\ /\ \ /
 C [C] C [C]
  \  \  /
   \ /\ /
    C C
```
MRICDF Door Control Process

Thread - Open_Door
- openReq
- obstruction
- X1
- True
- Y1
- openDoor
- StartTimer

Thread - Close_Door
- closeReq
- X2
- timeOut
- Y2
- closeDoor

Thread - Open_Event
- isOpened
- Y3
- opened

Thread - Close_Event
- isClosed
- Y4
- closed
Associating Software Behavior with AADL

```plaintext
process controller
Features
    ...
    ...
end controller

process implementation controller.c
Properties
    ...
    source_name => "MRICDF_Controller";
    source_language => MRICDF;
    source_location => "../PATH";
end controller.c
```
Ocarina Generation Frontend

- Frontend (AADL v 1.0 & v 2.0)
  - Parse AADL files
  - Syntactic Analysis
  - Semantic Analysis
Ocarina Generation Backend

- Backend (Source Language)
- Expansion
- Intermediate Tree
- Code Generation

![Diagram showing the process of Ocarina Generation Backend.]

Matthew Anderson and Sandeep K. Shukla

AADL and Polychrony based Embedded Computing System Design Environment
Dynamically Adding Threads

- Root
- Process 1
- Process N

Dispatch_Protocol => ...
Source => ...
Source_Language => MRICDF
Dynamically Adding Threads Cont’d

![Diagram of computational components]

- **Root**
  - **Process 1**
    - **Thread 1**
      - **Subprogram 1**
        - **Dispatch_Protocol** => ...
        - **Source** => “block1”
        - **Source_Language** => MRICDF
        - **Source_File** => ...
        - **Source_Language** => MRICDF

- **Process N**

Matthew Anderson and Sandeep K. Shukla

AADL and Polychrony based Embedded Computing System Design Environment
Dynamically Adding Threads Cont’d
Presentation Outline

1. Problem and Motivation
2. Approach
3. Current Project Status
4. Related Work
5. Conclusion
An extension has been written for Ocarina to accept code generated by MRICDF.

A new code generation option has been added for dynamically parsing and adding threads.
An extension has been written for Ocarina to accept code generated by MRICDF.

A new code generation option has been added for dynamically parsing and adding threads.

**Planned Additions:**

- Updates to MRICDF to streamline code generation
- Inclusion of analysis for Real-time properties and code verification
- Generate bus interface for distributed communication
Presentation Outline

1. Problem and Motivation
2. Approach
3. Current Project Status
4. Related Work
5. Conclusion
Related AADL tools

A lot of work has been done on creating tools for AADL development:

- **OSATE** as an environment for textual and graphical development
- **Ocarina** supports code generation from models
- **Cheddar** performs software schedulability analysis
- A number of model verification and analysis tools: **COMPASS** and **BIP**
AADL to Signal

"Toward polychronous analysis and validation for timed software architectures in AADL"

Y. Ma, H. Yu, T. Gautier, P. Le Guernic, J.P. Talpin, L. Besnard, and M. Heitz

- Translates an entire AADL model into Signal
- Formally analyze the translated model
- Simulate the behavior of the translated model
AADL to Signal

"Toward polychronous analysis and validation for timed software architectures in AADL"

Y. Ma, H. Yu, T. Gautier, P. Le Guernic, J.P. Talpin, L. Besnard, and M. Heitz

- Translates an entire AADL model into Signal
- Formally analyze the translated model
- Simulate the behavior of the translated model

Instead of APECS:
- Acts as an extension of the AADL development environment
- Is aimed at generating executables for the targeted physical platform
The APECS framework is based on the AADL development tools Osate / Ocarina and the polychronous tool EmCodeSyn.

APECS provides an environment for modeling end to end systems.

Software behavior is explicitly specified and formally verifiable:
- As opposed to informal specifications in C or Ada
- Free from additional synchronization constraints of Esterel/Lustre

We illustrate this with the Elevator System.
Any Questions?