Run-Time System and Bus Scheduling for TDL Components

Emilia Farcas
Claudiu Farcas
California Institute for Telecommunications and Information Technology

Wolfgang Pree
Christian Doppler Lab Embedded Software Systems

University of California, San Diego, USA

University of Salzburg, Salzburg, Austria
Contents

- Timing Definition Language
- Tool Chain
- Run-time system
- Bus Scheduling
- Summary
Timing Definition Language (TDL)

Vision
- Determinism, compositionality, and platform independence!

Advantages
- modularization of applications
- ECU consolidation
- transparent distribution of real-time components
  - the functional and temporal behavior of a system is the same no matter where a component is executed
  - for the developer there is no difference between local and distributed execution of components
  - the components can be developed independently
Core abstraction: Logical Execution Time (LET)

- LET means that the observable temporal behavior of a task is independent from its physical execution.
TDL Component Model - Example

Producer Module MPrd

mode1

Task1 [T=30ms] → a1
Task3 [T=20ms] → a3

mode2

Task1 [T=20ms] → a3
Task4 [T=10ms] → a4

Consumer Module MCns

modeX

Task2 [T=30ms] → a2

Import relationship
Contents

- Timing Definition Language
- Tool Chain
- Run-time system
- Bus Scheduling
- Summary
Tool Chain Overview

- .tdl
- Compiler
- .ecode
- Network config.
- Bus Scheduler plugin
- .busch
- Platform specific
- Platform plugin
- Glue code
- TDL run-time system
- Functionality code

Diagram showing the flow from .tdl to .ecode through Compiler, Bus Scheduler plugin, and Platform plugin to TDL run-time system.
Contents

- Timing Definition Language
- Tool Chain
- Run-time system
- Bus Scheduling
- Summary
Real-Time Execution Environment
TDL Scheduler

- Clock-driven scheduling
  - Off-line
  - Table based
  - Predictable but inflexible

- Priority-driven scheduling
  - On-line
  - Assigns priorities based on
    - fixed attributes: Rate Monotonic, Deadline Monotonic
    - dynamic attributes: Earliest Deadline First
  - Flexible but needs schedulability analysis
Scheduling in 2 steps:

- Schedule first the messages.
- Schedule then the tasks with deadlines constraints from messages.
Transparent Distribution

Unit of distribution: TDL module
Behavior: as if executed locally
Communication: via broadcast (bus)
Medium access control: TDMA
Cooperation model: Producer-Consumer (Push)
Transparent Distribution

![Diagram showing a network with nodes and tasks]
message sent according to bus schedule (TDMA)
Module Stubs

M2 imports M1
M4 imports M1, M3
Contents

- Timing Definition Language
- Tool Chain
- Run-time system
- **Bus Scheduling**
- Summary
Overview of Bus-Schedule Generation

Hyperperiod scheduling:
Bus period = LCM of periods of communicating tasks
Connections Identifier

- automatically detects who has to communicate with whom
- creates a data connection from a producer to all its consumers
- data may be produced, respectively read, with different frequencies in different modes
Basic Producer-Consumer Model

- 5 messages allocated to 4 frames
- the size of the last frame is the size of one message.
Optimized Producer-Consumer Model

- Only messages used by at least one consumer
Frame Scheduler

- Based on Reversed EDF: schedule frames as late as possible
- Adds control frames
- Conforms to the platform properties (e.g., bps, protocol overhead, inter frame gaps, min/max payload, clock resolution on the nodes)

Before scheduling a frame, we first try to multiplex or/and merge it with a previously scheduled frame if the timing constraints are met
  - Leads to reallocation of messages to frames and a reduction in the number of frames
## Multiplexing and Merging

### Multiplexing
- Allocating to the same frame messages from different modes of operations of the same module
  - These messages will be sent in *different bus cycles*
  - Maximum size

### Merging
- Allocating messages that will be sent in the same bus cycle
  - Sums sizes

### Combining Multiplexing and Merging

<table>
<thead>
<tr>
<th>Multiplexing</th>
<th>Merging</th>
</tr>
</thead>
<tbody>
<tr>
<td>M1.m1.msg1</td>
<td>M1.m1.msg2</td>
</tr>
<tr>
<td>M1.m3.msg4</td>
<td>M1.m3.msg5</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>2</th>
<th>4</th>
<th>6</th>
<th>8 bytes</th>
</tr>
</thead>
<tbody>
<tr>
<td>2</td>
<td>4</td>
<td>6</td>
<td>8 bytes</td>
</tr>
</tbody>
</table>

![Diagram](image-url)
Evaluation of the Optimizations

- Tool that generates TDL modules
  - 1 .. 25 nodes
  - 2 modules per node
  - each module has 5 modes, mode period 24ms, random producers and consumers less than 5
  - consumers use remote values uniformly from all other modules in the system
  - each task has a port of 4 bytes

- Case study on CAN + time synchronization layer
  - 1Mbit/s and 200us clock resolution on nodes

- Within a bus cycle of 24ms, we can send maximum 120 frames
Producer-Consumer Optimizations

![Graph showing total number of messages vs number of nodes for Basic Prod-Cons and Optimized Prod-Cons.]
Multiplexing and Merging

![Graph showing data throughput and total number of frames vs. total number of messages in Basic PC model.]

- Basic PC
- + mux
- + merge
- + mux + merge
- Optimized PC + mux + merge
Observations

- Different optimizations to save bandwidth and improve the scheduler’s feasibility
  - Producer-consumer optimizations and multiplexing reduce the payload,
  - merging reduces only the frame overhead

- The schedule specifies statically the size of the frame and the allocation of messages, but
- the structure of a frame changes depending on the active modes at run-time
Summary

- Automatic generation of
  - bus-schedule
  - e-code
  - glue code for the run-time system
- Respects LET semantics, thus supporting determinism, compositionality, and portability
- We can change the underlying platform and distribute components
  - without affecting the overall system behavior and
  - without changing the code of components
Thank you for your attention!