Supporting the Model-Driven Development of Real-time Embedded Systems with Simulation and Animation via Highly Customizable Code Generation

Nondini Das, Suchita Ganesan, Leo Jweda, Mojtaba Bagherzadeh, Reza Ahmadi, Nicolas Hili, Juergen Dingel
{ndas, ganesan, juwaidah, mojtaba, ahmadi, hili, dingel}@cs.queensu.ca
School of Computing, Queen’s University, Kingston, Ontario, Canada

Progress Report Meeting, École Polytechnique de Montréal, May 2016
Outline

Introduction and Motivation

The Supporting Infrastructure
  Overview
  Observability of Models: Towards a Taxonomy of Events
  Extending the PapyrusRT Code Generation
  Defining the Libraries: The Rover Model

Animating and Interacting with the model

Integrated Debugging
Introduction and Motivation

Monitoring Tools (e.g. LTTng)
Introduction and Motivation

From Runtime Model Monitoring...
Monitoring Tools (e.g. LTTng)

From Runtime Model Monitoring...

- Timing / Resource Constraint Violation
Introduction and Motivation

From Runtime Model Monitoring...

- Timing / Resource Constraint Violation
- Code-driven

Monitoring Tools (e.g. LTTng)

Code Execution Flow
Introduction and Motivation

From Runtime Model Monitoring...:

- Timing / Resource Constraint Violation
- Code-driven
- LTTng acts as an observer:
  - Listens for specific events
  - Does not disrupt the execution flow
Introduction and Motivation

- Code Execution Flow
- Monitoring Tools (e.g. LTTng)
- User’s Actions
- Custom Code
- Serious games (e.g. Minecraft)
- Model Animation
- Drivers

From Runtime Model Monitoring...

- Timing / Resource Constraint Violation
- Code-driven
- LTTng acts as an observer:
  - Listens for specific events
  - Does not disrupt the execution flow

... to a more general vision
Introduction and Motivation

Code Execution Flow

Monitoring Tools (e.g. LTTng)

User's Actions

Custom Code

Drivers

Serious games (e.g. Minecraft)

Model Animation

From Runtime Model Monitoring...

- Timing / Resource Constraint Violation
- Code-driven
- LTTng acts as an observer:
  - Listens for specific events
  - Does not disrupt the execution flow

...to a more general vision

- Pluggable components:
  - Animation and Interaction
  - Debugging
Introduction and Motivation

From Runtime Model Monitoring...

- Timing / Resource Constraint Violation
- Code-driven
- LTTng acts as an observer:
  - Listens for specific events
  - Does not disrupt the execution flow

...to a more general vision

- Pluggable components:
  - Animation and Interaction
  - Debugging
- Consumer / Producer of events
Three activities...
...For two purposes...

Three activities...
Infrastructure Overview

...to support model-driven design

- Allows for continuous development
- Driven by the code generation
- Highly Configurable

...For two purposes...

Three activities...
Open Source tool Support

The Supporting Infrastructure Overview

The GNU Project Debugger
TRACE COMPASS

+  +
Integrated debugging  Monitoring & Simulation  Animation & Interaction

+  +
Quality Assurance  Communication

May 2016  5 / 14
Infrastructure’s Challenges

Challenges to address:
- Each pluggable component is an observer that consumes / produces specific events;
- Each component has to interact with the generated code;
- The generated code has to interact with the hardware platform.

How we addressed them:
- Definition of a Context Configuration Model that lists all monitorable events;
- Extension of the PapyrusRT code generator;
- Definition of a Rover Library to interact with the hardware.
Infrastructure’s Challenges

Challenges to address

Each pluggable component is an observer that consumes / produces specific events; each component has to interact with the generated code; the generated code has to interact with the hardware platform.

How we addressed them:

- Definition of a Context Configuration Model that lists all monitorable events;
- Extension of the PapyrusRT code generator;
- Definition of a Rover Library to interact with the hardware.
Infrastructure’s Challenges

Challenges to address

Each pluggable component is an observer that consumes / produces specific events;
Infrastructure’s Challenges

Challenges to address

- Each pluggable component is an observer that consumes / produces specific events;
- Each component has to interact with the generated code;
Infrastructure’s Challenges

Challenges to address

- Each pluggable component is an observer that consumes / produces specific events;
- Each component has to interact with the generated code;
- The generated code has to interact with the hardware platform.
Infrastructure’s Challenges

Challenges to address

Each pluggable component is an observer that consumes / produces specific events;

Each component has to interact with the generated code;

The generated code has to interact with the hardware platform.

How we addressed them

Definition of a Context Configuration Model that lists all monitorable events;

Extension of the PapyrusRT code generator;

Definition of a Rover Library to interact with the hardware.
Infrastructure’s Challenges

Challenges to address

- Each pluggable component is an observer that consumes / produces specific events;
- Each component has to interact with the generated code;
- The generated code has to interact with the hardware platform.

How we addressed them

- Definition of a Context Configuration Model that lists all monitorable events;
Configuring the infrastructure

Event
- Communication
  - Message
  - Queue
- Attribute
  - Created
  - Changed
- Capsule
  - Destroyed
- State Machine
  - Bound
- State
  - Active
- Transition
  - Triggered
Infrastructure’s Challenges

Challenges to address

- Each pluggable component is an observer that consumes / produces specific events;
- Each component has to interact with the generated code;
- The generated code has to interact with the hardware platform.

How we addressed them

- Definition of a Context Configuration Model that lists all monitorable events;

The Supporting Infrastructure

Observability of Models: Towards a Taxonomy of Events

May 2016
Infrastructure’s Challenges

Challenges to address

- Each pluggable component is an observer that consumes / produces specific events;
- Each component has to interact with the generated code;
- The generated code has to interact with the hardware platform.

How we addressed them

- Definition of a Context Configuration Model that lists all monitorable events;
- Extension of the PapyrusRT code generator;

The Supporting Infrastructure

Observability of Models: Towards a Taxonomy of Events

May 2016 8 / 14
Extending the PapyrusRT Code Generator

The Supporting Infrastructure

Extending the PapyrusRT Code Generation

May 2016

9 / 14
Extending the PapyrusRT Code Generator

The Supporting Infrastructure

Extending the PapyrusRT Code Generation

May 2016
Extending the PapyrusRT Code Generator

The Supporting Infrastructure Extending the PapyrusRT Code Generation

May 2016
Extending the PapyrusRT Code Generator

The Supporting Infrastructure Extending the PapyrusRT Code Generation
Extending the PapyrusRT Code Generator

The Supporting Infrastructure Extending the PapyrusRT Code Generation

May 2016 9 / 14
Extending the PapyrusRT Code Generator

The Supporting Infrastructure

Extending the PapyrusRT Code Generation
Infrastructure’s Challenges

Challenges to address
- Each pluggable component is an observer that consumes / produces specific events.
- Each component has to interact with the generated code.
- The generated code has to interact with the hardware platform.

How we addressed them
- Definition of a Context Configuration Model that lists all monitorable events.
- Extension of the PapyrusRT code generator.

The Supporting Infrastructure Extending the PapyrusRT Code Generation

May 2016 10 / 14
Infrastructure’s Challenges

Challenges to address

- Each pluggable component is an observer that consumes / produces specific events;
- Each component has to interact with the generated code;
- The generated code has to interact with the hardware platform.

How we addressed them

- Definition of a Context Configuration Model that lists all monitorable events;
- Extension of the PapyrusRT code generator;
- Definition of a Rover Library to interact with the hardware.
Definition of the Rover Library

Control Software
- Contains the Business Logic
- Does not know about the hardware configuration
- Interacts with the Rover Library
- Makes the glue with the Hardware
- Defines the protocols the Business Logic will have to interact with
- Specific to a design configuration

The Supporting Infrastructure Defining the Libraries: The Rover Model

May 2016 11 / 14
Definition of the Rover Library

- **Control Software**: Contains the Business Logic
- **Does not know about the hardware configuration**
- **Interacts with the Rover Library**

The Supporting Infrastructure Defining the Libraries: The Rover Model
Definition of the Rover Library

- Control Software
  - Contains the Business Logic
  - Does not know about the hardware configuration
  - Interacts with the Rover Library

- Rover Library
  - Makes the glue with the Hardware
  - Defines the protocols the Business Logic will have to interact with
  - Specific to a design configuration

- GPIO Class
- File System
- Hardware
Definition of the Rover Library
Animating the Model

Web based animation & State Machine
Live Monitoring

Animating and Interacting with the model
Animating the Model

Web based animation & State Machine
Live Monitoring

☑ Animation of the Rover Model
Animating the Model

Web based animation & State Machine
Live Monitoring

- Animation of the Rover Model
- Code-driven (different from Moka)
Animating the Model

Web based animation & State Machine Live Monitoring

- Animation of the Rover Model
- Code-driven (different from Moka)
- Works as an observer:
  - Bi-directional socket communication with the C++ code
  - Listen all events (state changes, transitions fired)
  - Would at last interact with the code execution flow (not supported yet)
What’s Next?

**Vision**

- Improve the different parts of the infrastructure, especially the code generator to allow for several configurations to be used at the same time;

- Define different libraries for different models;

- Some bugs have to be corrected in Papyrus / PapyrusRT (e.g. Internal transition with effects, graphical glitches since the new Eclipse version, etc.).

**Animation & Interaction**

- Allow the user to interact with the model using the animation view;

- Implement other animation engines (2D/3D, Unity, etc.)

- Propose a creation tool to automatically create animation views to animate and interact with the model;

- Look at Moka to see if it can be used to simulate the state machine execution;
Thank You!

Questions?