

Keep it Simple: Pruning Papyrus(-RT) for Real-Time Systems

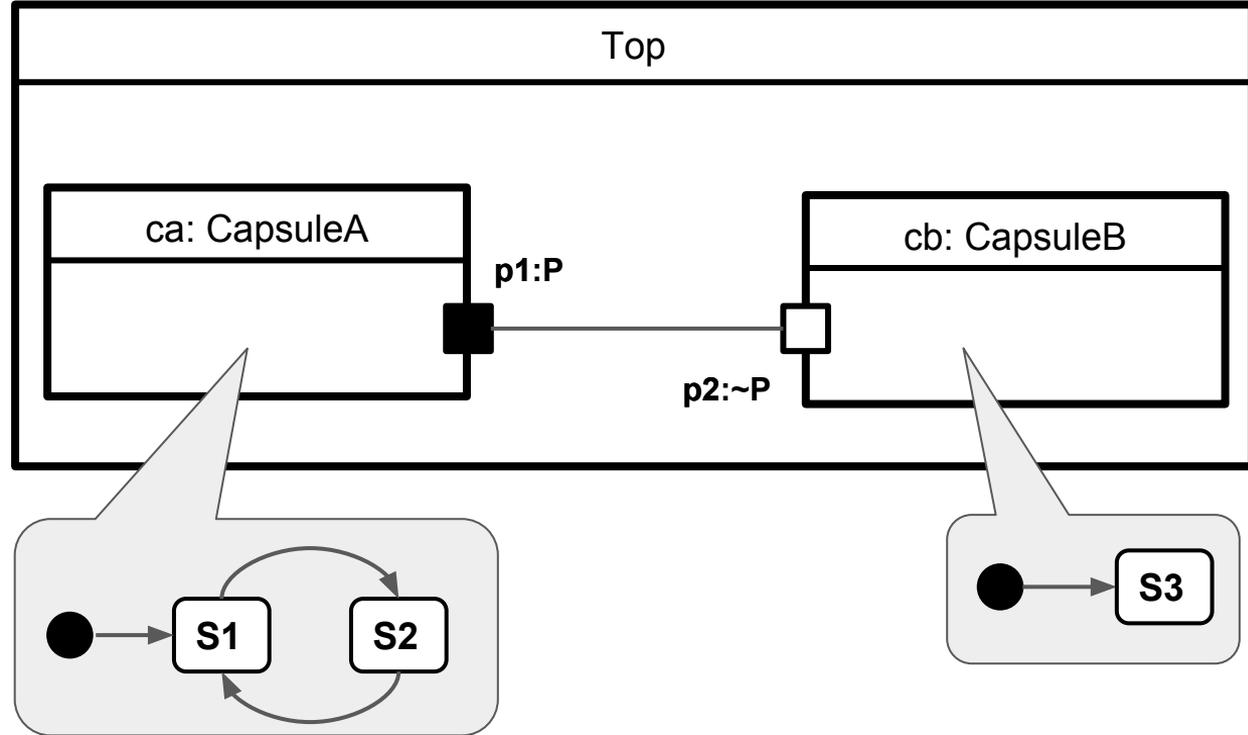
Nicolas Hili, Juergen Dingel

Sum Ergo
I am therefore I compute
Computo

Context

UML for Real-Time:

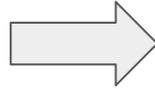
- Subset of UML2
- A small set of concepts:
 - Capsules
 - Capsule Parts
 - Ports
 - Protocols
 - Connectors
 - StateMachines
- Action semantics



Motivation

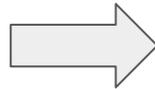


UML-RT is specific



Need for dedicated environments

Development of Real-Time
(Embedded) Systems



Support for specific real-time activities
Support for (configurable) platform modeling



Part I

Customizing Papyrus-(RT)

How and Where to Customize Papyrus ?

The screenshot displays the Eclipse SDK interface for a Papyrus project named 'runtime-rover - Java - PingPongCopy/pingPong.di'. The main workspace shows a UML class diagram for 'Top' with two classes: '# pinger: Pinger [1]' and '# ponger: Ponger [1]'. A connector links the 'pinger' and 'ponger' attributes. Below this, a state machine diagram for 'Ponger::PongerStateMachine' shows an 'init' transition leading to a state 'Playing' with the entry action '/entry OpaqueBehavior null'. The left-hand Model Explorer shows a tree structure with 'Pinger' and 'Ponger' as primary nodes. The right-hand side contains two palettes: the top one for 'Edges' (Link, Connector) and the bottom one for 'Nodes' (Region, State, Transition, Link). The bottom Properties view is open for the 'Ponger : Ponger' class, showing various attributes and their values.

UML	Name	Value	UML	Value
Comments	Is derived	<input type="radio"/> true <input checked="" type="radio"/> false	Is derived union	<input type="radio"/> true <input checked="" type="radio"/> false
Profile	Is leaf	<input type="radio"/> true <input checked="" type="radio"/> false	Is ordered	<input checked="" type="radio"/> true <input type="radio"/> false
Style	Is read only	<input type="radio"/> true <input checked="" type="radio"/> false	Is static	<input type="radio"/> true <input checked="" type="radio"/> false
Appearance	Is unique	<input checked="" type="radio"/> true <input type="radio"/> false	Visibility	protected
Rulers And Grid	Type	Ponger	Multiplicity	1

Papyrus-RT

runtime-Rover - Java - PingPong/pingPong.di - Eclipse SDK

File Edit Diagram Navigate Search Papyrus Project UML-RT Menu Run Window Help

Model Explorer pingPong.di

- RootElement
 - «Protocol» PingPong
 - out ping ()
 - in pong ()
 - «Capsule» Pinger
 - «RTPort» pinger : PingPong
 - «RTStateMachine» StateMachine
 - Pinger
 - «Capsule» Ponger
 - «RTPort» ponger : ~PingPong
 - «RTStateMachine» PongerStateMachine
 - «RTRegion» Region
 - Init
 - «RTPseudostate» <Pseudostate>
 - «RTState» Playing
 - <Opaque Behavior>
 - Ponger::PongerStateMachine
 - Ponger
- «Capsule» Top
 - «CapsulePart» pinger : Pinger
 - «CapsulePart» ponger : Ponger
 - «RTConnector» connector
- Top

«EPackage, ModelLibrary» UML Primitive Types

«ModelLibrary» Ecore Primitive Types

Top Pinger Ponger

PongerStateMachine

init → Playing /entry OpaqueBehavior null

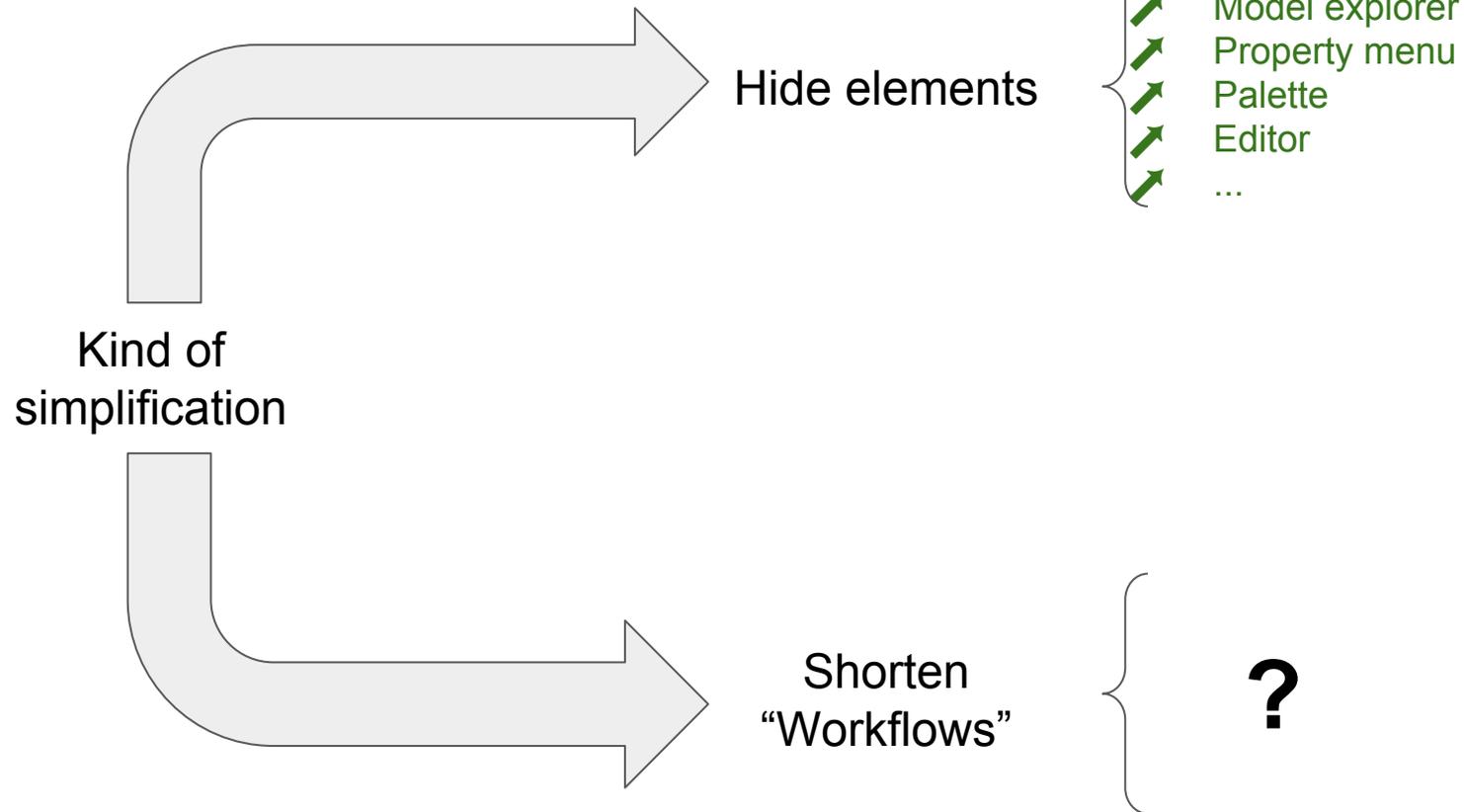
Ponger::PongerStateMachine

Problems Javadoc Declaration Properties

ponger : Ponger

UML-RT	Name	ponger
UML	Capsule	«Capsule» Ponger
Comments	Capsule Part Properties	
Profile	Kind	<input checked="" type="radio"/> Fixed <input type="radio"/> Optional <input type="radio"/> Plugin
Style	Multiplicity	1
Appearance	Aggregation	composite
Rulers And Grid		

Simplifying Papyrus



Shortening “Workflows”: Element creation

The screenshot shows a UML modeling environment with the following components:

- Model Explorer:** A tree view on the left showing the project structure. The `«Capsule» Ponger` package is selected, and its `«CapsulePart» pinger : Pinger` part is highlighted with a blue oval.
- Diagram:** The main workspace shows a `«Capsule» Top` capsule containing a `# pinger: Pinger [1]` compartment and a `+ pinger: PingPong [1]` compartment. A blue arrow points from the `«CapsulePart» pinger : Pinger` in the Model Explorer to the `+ pinger: PingPong [1]` compartment in the diagram.
- Callout 1:** A white box with a blue border and arrow pointing to the diagram, containing the text: "DnD the `Ponger` capsule into the `Top` capsule...".
- Callout 2:** A white box with a blue border and arrow pointing to the `«CapsulePart» pinger : Pinger` in the Model Explorer, containing the text: "... Will automatically initialize the `pinger` capsule part."
- Palette:** A vertical palette on the right with a "Capsule St..." section containing `Port`, `Capsule Part`, and `Connector`.
- Properties View:** At the bottom, the `«Capsule» Ponger` properties are shown, with the `Name` field containing `Ponger`.

Shortening “Workflows”: Diagram Initialization & Navigation

The screenshot displays the Eclipse IDE interface with the following components:

- Model Explorer:** Shows a project structure with a capsule part selected: `«CapsulePart» ponger : Ponger`.
- Diagram:** Shows two capsule diagrams. The left one contains:

```
# pinger: Pinger [1]
+ pinger: PingPong [1]
```

The right one contains:

```
# ponger: Ponger [1]
pingPong: ~PingPong [1]
```
- Palettes:** The right palette shows elements like Port, Capsule Part, and Connector.
- Properties View:** Shows properties for the selected capsule part, including UML-RT, UML, Comments, Profile, and Style.

Three callout boxes provide instructions:

- Top box: "Double-clicking on the newly created capsule part..." (points to the right capsule diagram).
- Middle box: "Double-clicking on the capsule does nothing, though." (points to the left capsule diagram).
- Bottom box: "... Will automatically open the `Ponger` diagram (automatically initialized when the `Ponger` capsule was created)." (points to the Properties view).

Shortening "Workflows": StateMachine modeling

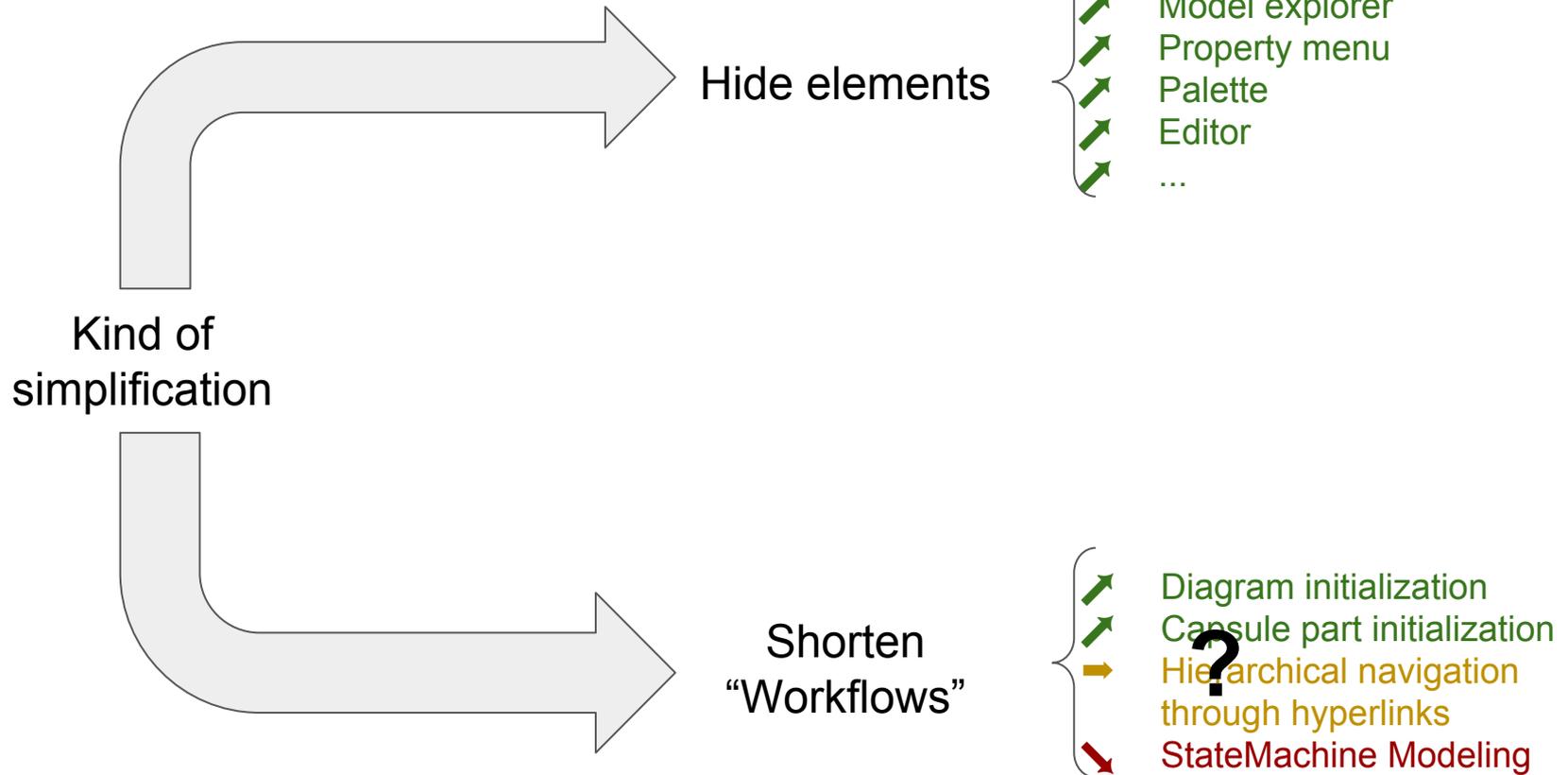
The screenshot shows the Eclipse IDE interface with the following components:

- Model Explorer:** A tree view on the left showing the project structure. It includes a `RootElement` containing several `«Protocol» PingPong` and `«Capsule» Pinger` elements, along with their associated state machines and regions.
- Diagram Editor:** The central workspace displays a UML diagram for a `Pinger` capsule. It shows a state machine with a state `# pinger: Pinger [1]` and a transition `+ pinger: PingPong [1]`. A callout points to this diagram with the text: "Before defining the implementation".
- Language Selection:** At the bottom, the `UML-RT Language` dropdown menu is open, showing `C++` selected. A callout points to this menu with the text: "Defining the language can be done globally...".
- Callouts:** A callout from the Model Explorer points to the `RootElement` with the text: "The language has to be locally specified...". Another callout from the bottom points to the `C++` selection with the text: "But every time a Behavior has to be created...".
- Palettes:** On the right, there are palettes for `Capsule St...` and `Port`, `Capsule Part`, and `Connector`.

Shortening “Workflows”: Behavior modeling

The screenshot shows the Eclipse IDE interface for modeling a Pong game. The main window displays a state machine diagram for `PongerStateMachine` with an `init` transition leading to a `Playing` state. The Model Explorer on the left shows the project structure, including the `init` state and the `Ponger::PongerStateMachine` class. Two dialog boxes are open: `Create a new Constraint` and `Create a new OpaqueExpression`. A callout box points to the `Constrained element` field in the `Create a new Constraint` dialog with the text "The same goes for defining constraints".

Simplifying Papyrus



Shortening “Workflows”: Making some Assumptions

Assumption 1:

A **capsule** may or may not have a **behavior**. If a behavior is defined, then it is modeled using a **StateMachine**.

Assumption 2:

While UML supports different behaviors (e.g., **Activity**, **Function Behavior**, **Opaque Behavior**), for defining **entry / exit / do** activities and **transition's effect**, only **Opaque Behaviors** are used by Papyrus-RT.

Assumption 3:

If the **language** is globally defined, there is no need to locally redefine it when creating the **opaque behaviors**.

Assumption 4:

If an **opaque behavior** can be defined in the target language (e.g., C++), the action semantics is defined by the Papyrus **RunTime Service**.

Shortening “Workflows”: Opaque Behavior modeling

The screenshot shows the Eclipse IDE with the 'runtime-Rover - Java - PingPong/pingPong.di - Eclipse SDK' window. A dialog box titled 'Create a new Opaque Behavior' is open, showing the 'Name' field, 'Language' dropdown (set to C++), and 'Is abstract' and 'Is active' checkboxes. The 'Specification' field contains the code: `std::cout << "hello world" << std::endl;`. Callouts point to the 'port' field, the code box, and the 'Outgoing message' field. A larger callout explains that the action semantics is limited to sending messages to other capsules. The background shows a UML diagram with a state machine and a palette of UML elements.

port

Outgoing message

While anything can be written in this box...

...most of the time, the action semantics is limited to sending messages to other capsules.

UML Comments

Cancel OK

Shortening “Workflows”: Contributions

Assumption 1:

A **capsule** may or may not have a **behavior**. If a behavior is defined, then it is modeled using a **StateMachine**.

Assumption 2:

While UML supports different behaviors (e.g., **Activity**, **Function Behavior**, **Opaque Behavior**), for defining **entry / exit / do** activities and **transition's effect**, only **Opaque Behaviors** are used by Papyrus-RT.

Assumption 3:

If the **language** is globally defined, there is no need to locally redefine it when creating the **opaque behaviors**.

Assumption 4:

If an **opaque behavior** can be defined in the target language (e.g., C++), the action semantics is defined by the Papyrus **RunTime Service**.

Customization 1:

- Initializing StateMachine models
- Extending the Hyperlink feature to open StateMachine diagrams

Customization 2:

- Simple hybrid textual / graphical notation for modeling state and transition behaviors.

Shortening "Workflows": StateMachine initialization

The screenshot shows the Eclipse IDE interface with a project named 'pingPong'. The Model Explorer on the left shows a hierarchy of elements including 'PingPong', 'Pinger', and 'Top'. A dialog box titled 'No StateMachine' is open in the center, displaying the message: 'This capsule does not own a statemachine. Do you want to create one?'. The dialog has 'No' and 'Yes' buttons. A mouse cursor is hovering over the 'Yes' button. Several callout boxes provide context: one points to the 'Top' capsule in the Model Explorer with the text '... will open a dialog to create one.'; another points to the dialog title with 'Double-clicking on a capsule containing no StateMachine...'; a third points to the 'Yes' button with '...causes the creation of the StateMachine and the opening of the diagram.'; and a fourth points to the 'Yes' button with '... clicking on Yes...'. The status bar at the bottom left indicates '1 item selected'.

Shortening “Workflows”: StateMachine initialization

The screenshot shows the Eclipse IDE interface with the following components:

- Model Explorer (Left):** A tree view showing the project structure. The path is: RootElement > «Protocol» PingPong > «Capsule» Pinger > «Capsule» Ponger. The 'Ponger' capsule is selected.
- Diagram Editor (Center):** Displays the UML diagram for the selected 'Ponger' capsule. It contains a state machine diagram with a state named 'pingPong' and a multiplicity of 1. A mouse cursor is hovering over the capsule's title bar.
- Palette (Right):** Shows the 'Capsule St...' palette with elements: Port, Capsule Part, and Connector.
- Properties View (Bottom):** Shows the properties for the selected '«Capsule» Ponger' element. It has a table with columns 'UML-RT' and 'Name'.

UML-RT	Name
UML	

Two callout boxes provide context:

- Top callout: "Double-clicking on a capsule that already contains a StateMachine..." with an arrow pointing to the capsule title bar.
- Bottom callout: "...causes the opening of the existing diagram." with an arrow pointing to the diagram editor.

Shortening “Workflows”: Hybrid Notation

The screenshot shows the Eclipse IDE interface with the following components and callouts:

- Model Explorer (Left):** Shows a project structure with artifacts like «Protocol» PingPong, «Capsule» Pinger, «Capsule» Ponger, «RTPort» ponger, «RTStateMachine», «RTRegion» Region, «RTState» Playing, and «RTPseudostate» <Pseudostate>.
- Diagram (Center):** Displays a UML diagram with a state named "Playing".
- Properties View (Right):** Lists UML elements such as Transition, Choice, Deep History, Entry Point, Exit Point, Initial State, and Junction.
- Callouts:**
 - Top center: "...and displayed."
 - Top right: "When selecting a state or a transition, the Action view's embedded editor becomes active."
 - Middle left: "...the UML artifacts are created..."
 - Middle center: "Autocompletion of ports, incoming and outgoing messages."
 - Bottom left: "It contains an Xtext embedded editor for editing state and transition behaviors."
 - Bottom center: "When the editor contents is of an Action saved..."
 - Bottom right: "When a state is selected, the editor supports the creation of entry / exit / do actions, as well as internal transitions."

Shortening “Workflows”: Hybrid Notation

The screenshot displays the Eclipse IDE interface for a Java project named 'PingPong/pingPong.di'. The main editor shows a UML State Machine Diagram for the class 'PongerStateMachine'. The diagram starts with an initial state (black dot) that transitions to a state named 'Playing'. This state has an entry point (indicated by a circle) and an opaque behavior (indicated by a box labeled 'OpaqueBehavior null'). A self-loop transition is shown on the 'Playing' state. The left-hand side of the IDE shows the Model Explorer with a tree view of the project structure, including 'RootElement', '«Protocol» PingPong', '«Capsule» Pinger', and '«Capsule» Ponger'. The right-hand side shows the Palette with various state machine elements like State, Transition, Choice, etc. Below the diagram, the Action View shows a code snippet:

```
on ponger.ping() /  
  ponger.pong().send();
```

 A callout box points to this code with the text: "The same goes for external transitions."

Shortening “Workflows”: Hybrid Notation

Hybrid textual / graphical notation:

- Implementation of a specific view ;
- Integration of an embedded Xtext editor ;
- Supports state's entry / do / exit actions ;
- Supports state's internal transition ;
- Supports external transition's effect.

Current limitations:

- Basic action semantics (e.g., does not support constraints, or multiple triggers) ;
- Does not support complex behaviors (e.g., written in C++).

Work to do:

- Improve the grammar ;
- Support of the Papyrus RTS library (log, frame, timer) ;
- Mixing with other Xtext grammars (e.g., C++) when the target language is defined.

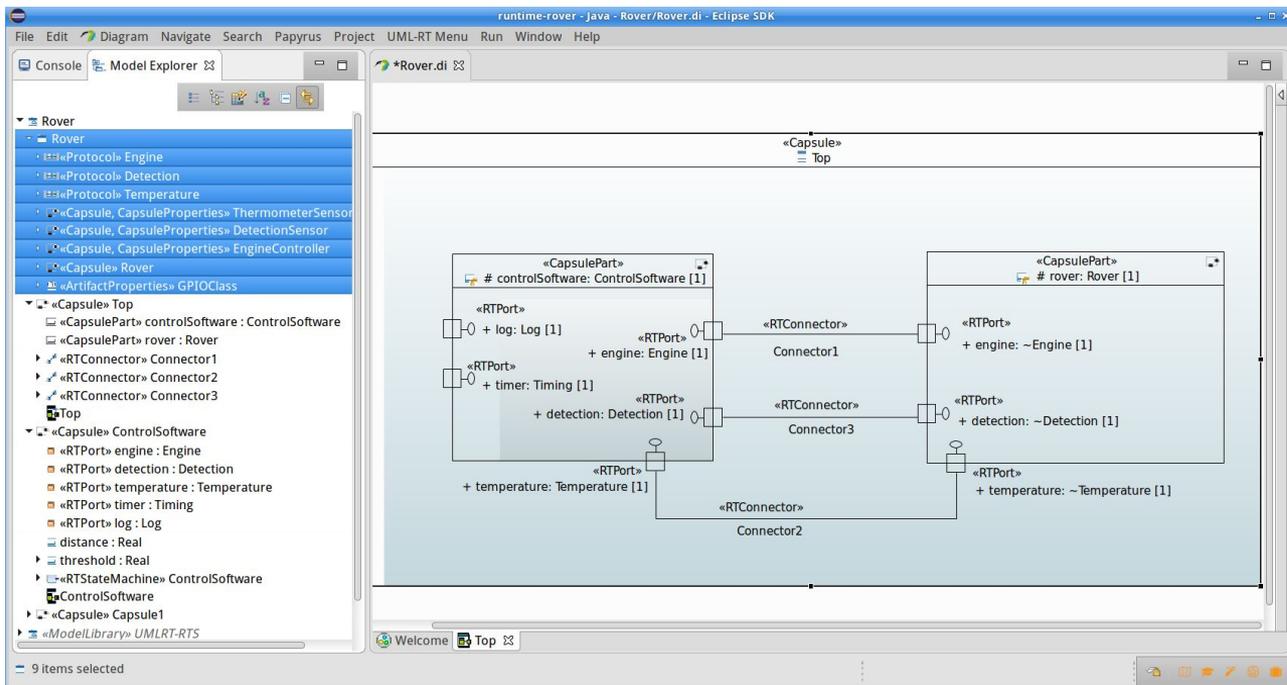


Part II

Supporting Specific Activities for Real-time (Embedded) System Development

The Rover Case Study

Graphical Model using Papyrus-RT



Textual Model in TUML-RT

```
model Model
{
    import uri "platform:/resource/org.eclipse.papyrusrt.umlrt.common"
    import uri "platform:/resource/org.eclipse.papyrusrt.umlrt.common"

    artifact GPIO {
    }
    capsule ControlSoftware {
        conjugate port engine: Rover.Engine;
        conjugate port detection: Rover.Detection;
        conjugate port temperature: Rover.Temperature;
        port timer: RTSLibrary.Timing;
        port log: RTSLibrary.Log;

        statemachine
        {
            initial controlSoftware_init;

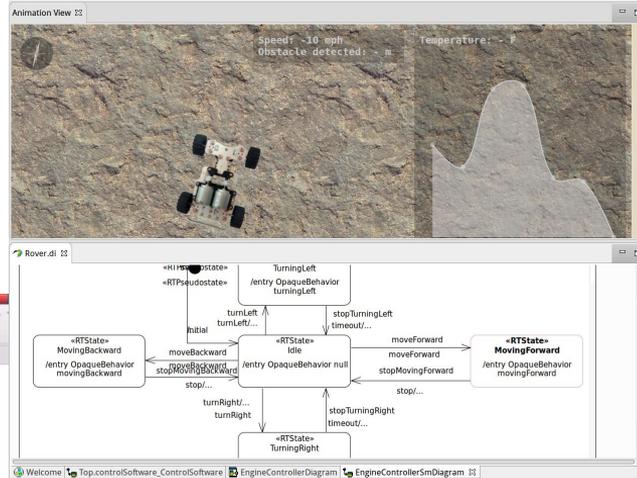
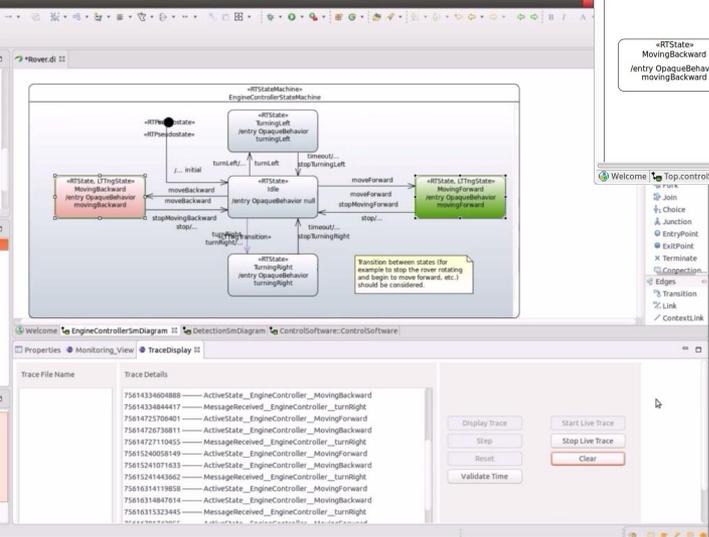
            state IDLE
            {
                entry action '
                    timer.informIn(UMLRTTimespec( 3, 0 ));
                    log.log("Entering into IDLE State");
                ';
            }

            state MOVING_FORWARDS
            {
                entry action '
                    engine.moveForward().send();
                    timer.informIn(UMLRTTimespec( 3, 0 ));
                    log.log("Entering into Moving Forward State");
                ';
            }

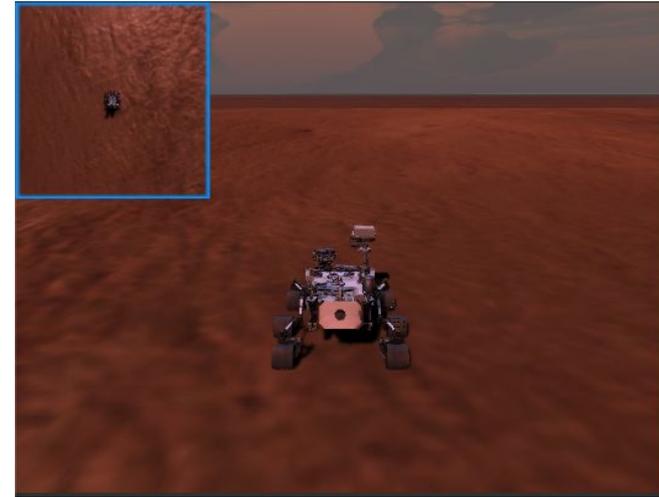
            state MOVING_BACKWARDS
            {
                entry action '
                    engine.stop().send();
                    engine.moveBackwards().send();
                    timer.informIn(UMLRTTimespec( 3, 0 ));
                    log.log("Entering into Moving Backwards State");
                ';
            }
        }
    }
}
```

Supporting Specific Activities for RTE Systems

Model Monitoring of timing constraints (LTTng)

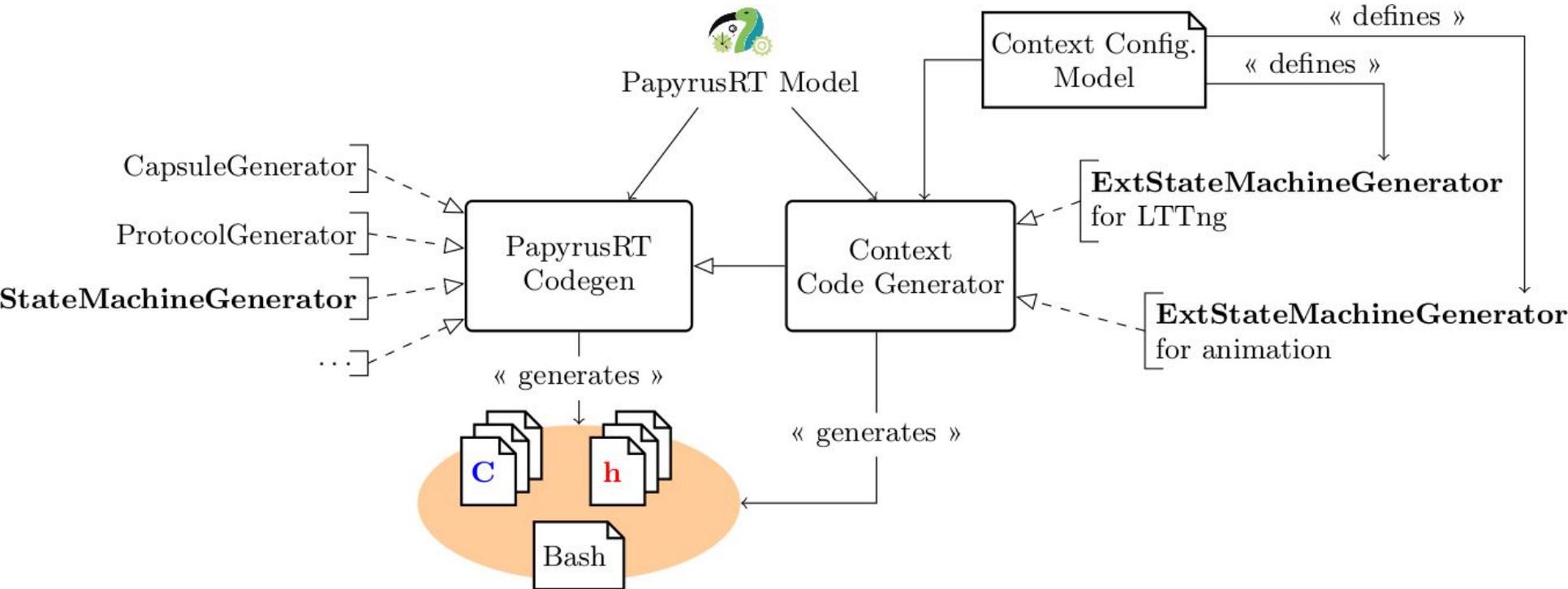


Model Animation in a 2D environment (Web-based)



Model Animation in a 3D environment (Unity)

Extending the PapyrusRT code generator



Platform Layering

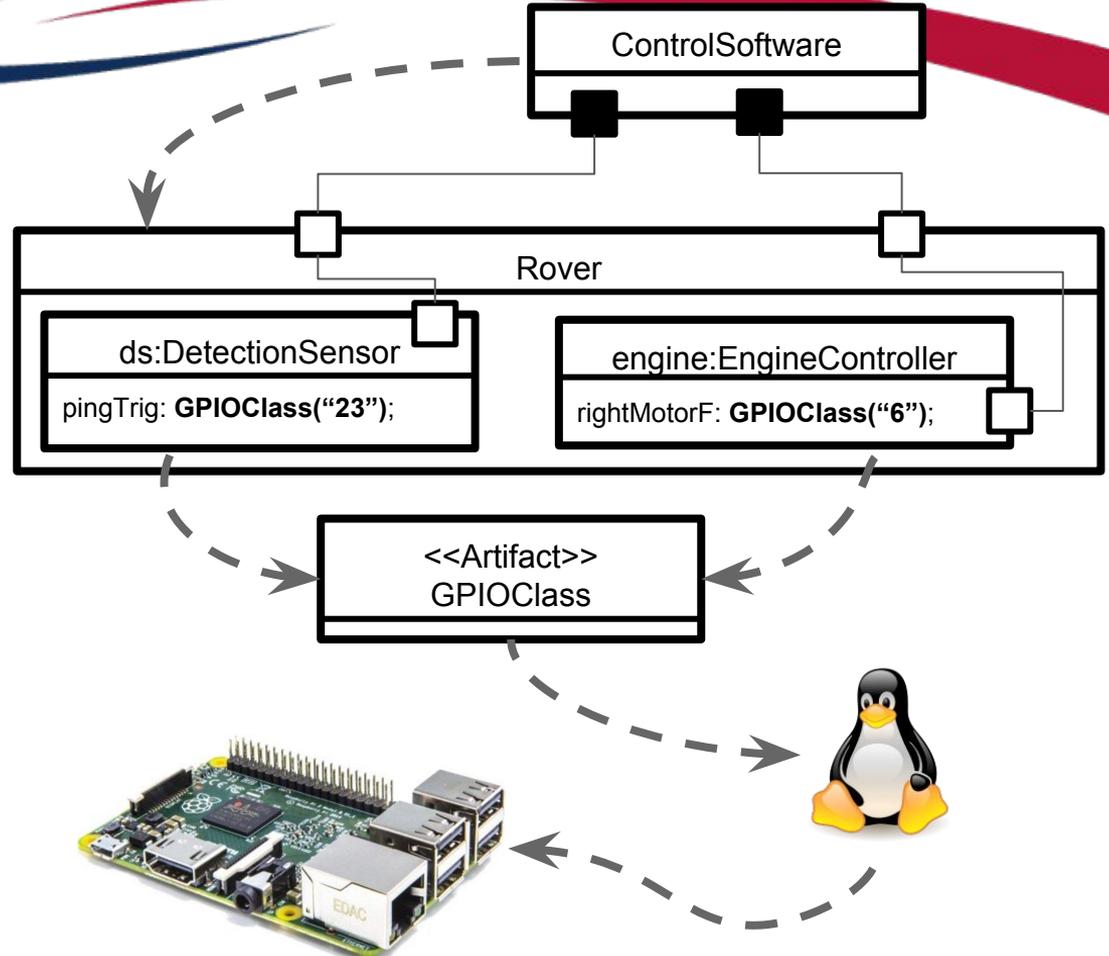
Application

Rover Library

GPIO Class

File System

Hardware



Platform Layering

Application

- Contains the Business Logic ;
- Does not know about the HW configuration ;
- Interacts with the Rover Library.

Rover Library

- Define the protocols the Business Logic will have to interact with ;
- Interacts with the Hardware ;
- **Specific to a design configuration.**

GPIO Class

File System

Hardware

? Problem:
How to change the design configuration without having to modify the Rover Library ?

✓ Proposition:
Embedding a specific configuration page with the loaded library

Configuration Page to configure the Rover package

Allow for editing the design configuration (i.e., the GPIO mapping)

Give an overview of the platform.

The UMLRT-Rover package is loaded in the model.

Property	Capsule	GPIO
Trig	DetectionSensor	GPIO 23
pinEcho	DetectionSensor	GPIO 24
rightMotorBackwards	EngineController	GPIO 5
leftMotorForward	EngineController	GPIO 22
rightMotorForward	EngineController	GPIO 6
leftMotorBackwards	EngineController	GPIO 27

Configuration page

CPU Core : Quadcore ARM Cortex-A53, 64Bit
Clock Speed : 1.2GHz
GPIOs : 2 x 20 Pin Header
GPU : 400 MHz VideoCore IV®
Name : Raspberry PI 3 Model B
RAM : 1 GB
Power Supply : 2.5 A

Property	Value
Info	
derived	false
editable	true

Conclusion



Several customizations have been done for RTE system development within Papyrus-RT.

Some are general improvement :

- Shortening “Workflows” ;
- Simplifying the modeling of behaviors.

Others are specific to RTE systems :

- Animation view ;
- Monitoring view ;
- Configuration Page.

Work to do:

- Improve the action semantics ;
- Provide a generic implementation for loading platform libraries and configuration pages.

Tutorial @ MODELS 2016

MODELS 2016 Program - Chromium

MODELS 2016 Program x

program.models2016.irisa.fr

MODELS 2016 - Program

Sunday, Oct 2, 2016

	Sessions	Rotonde Surcouf	Rotonde J. Cartier	Bouvet 1	Bouvet 2	Charcot	Vauban 1	Vauban 2
09:00				Doctoral Symposium ☆	Advanced Model Management with Epsilon ☆	ARCADIA in a Nutshell ☆	Hybrid Graphical/Textual Modelling and Code Generation with PapyrusRT ☆	ME ☆
10:30		Coffee Break						
11:00				Doctoral Symposium ☆	Advanced Model Management with Epsilon ☆	ARCADIA in a Nutshell ☆	Hybrid Graphical/Textual Modelling and Code Generation with PapyrusRT ☆	ME ☆
12:30			Lunch					

SCHOOL OF
Queen's Computing
Queen's University
Kingston, Ontario, Canada



Sum Ergo
I am therefore I compute
Computo

Thank You