Eclipse Scheduling and Network Delay Analysis Plugin

Gustavo Hidalgo

July 13, 2015

Contents

1 Introduction

This report describes the design and implementation of an Eclipse plug-in for the purpose of designing real-time systems based on the ARINC 653 standard for integrated modular avionics and ARINC 664 for network connections. The plug-in defines a meta-model for a system which can be used by several algorithms to analytically ensure schedulability stability and an upper bound on the network delay for frames sent over a network. The plug-in also has a comprehensive validation suite which prevents users from creating structurally invalid configurations along with a graphical editor for the network definition which greatly helps to describe and debug network configurations.

This document contains hyperlinks to online resources. View it on a computer to follow the links for more information.

2 Eclipse Modeling

The Eclipse Modeling Framework is a tool for designing and manipulating models in Eclipse with many facilities for code generation and persistent data manipulation. It is important to be familiar with this framework in order to understand this project. The most important features used are:

1. **Ecore tools** : A graphical meta-model editor

2. **OCLinEcore** : A modified version of the Object Constraint Language used to define structural constraints for an instance of a meta-model

3. **Code Generation** : A large part of the code in the plug-in was generated by Ecore.

4. **Sirius** : A framework for creating graphical DSLs based on Ecore models

These tools are Eclipse plug-ins which must be installed first before modifying the plugin. If you installed the version of Eclipsed used for modeling, you will find a button in the toolbar like in figure 1. This

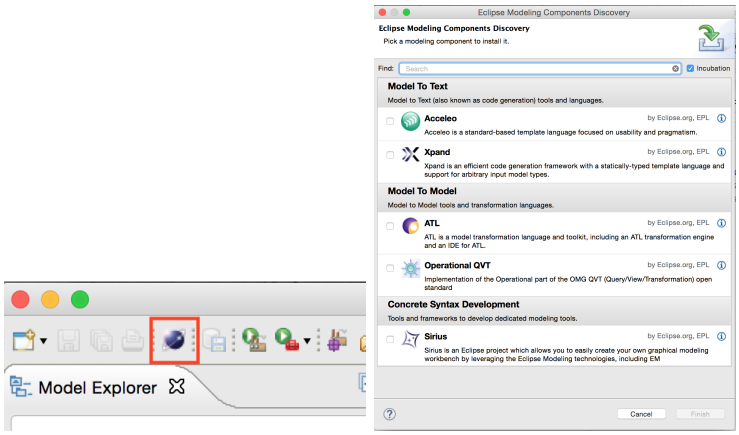

Figure 1: Install modeling plugins

opens a window with many modeling plugins to install, you should install Sirius and EcoreTools.

3 Meta Model

The classes in the Ecore meta-model are described in this section along with the OCLinEcore validation steps that are applied.

For reference, the full meta-model is displayed first and the individual components after.

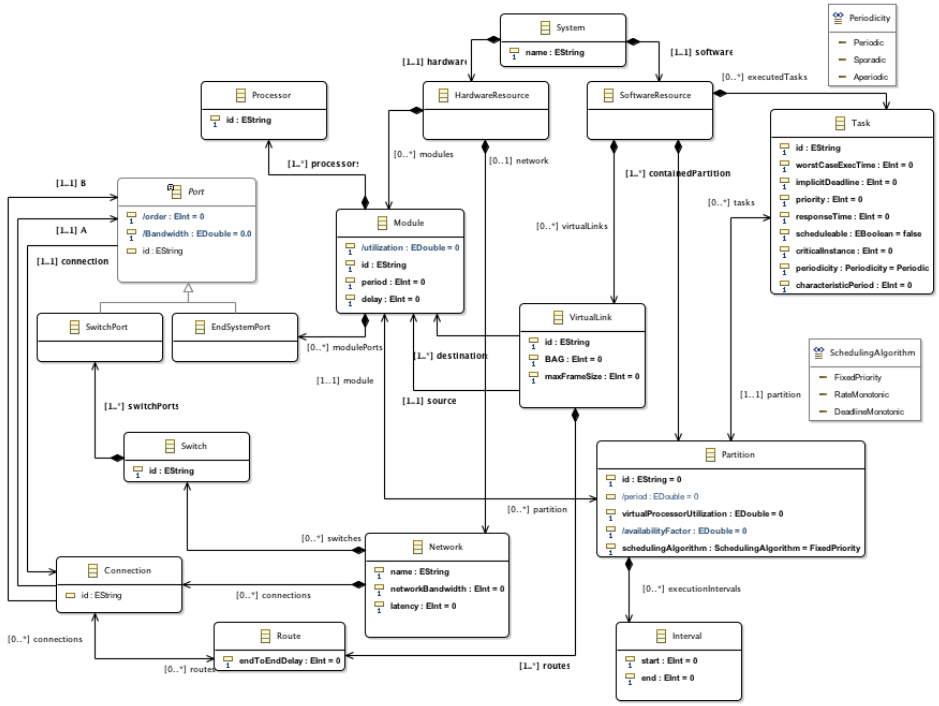

Figure 2: Full UML meta-model

System The root object for an instance of the meta model. This class serves to hierarchically organize the rest of the classes.

Listing 1: System constraints

```
class System {
    property hardware : HardwareResource { composes };
    attribute name : String;
    property software : SoftwareResource { composes };
}
```

Hardware and Software Resources These two classes hierarchically organize other model elements.

Listing 2: Hardware and Software constraints

```
class SoftwareResource {
    property executedTasks : Task[*] { ordered composes  };
    property containedPartitions : Partition[+] { ordered composes  };
    property virtualLinks : VirtualLink[*] { ordered composes  };
}
class HardwareResource {
    property modules : Module[*] { ordered composes  };
    property network : Network[?] { composes  };
}
```

Task A task is a software service that has well defined temporal activation parameters.

Listing 3: Task constraints

```
class Task {
    attribute id : String { id  };
    attribute worstCaseExecTime : ecore::EInt = '0';
    attribute implicitDeadline : ecore::EInt;
    attribute priority : ecore::EInt;
    attribute responseTime : ecore::EInt;
    attribute scheduleable : Boolean;
    attribute criticalInstance : ecore::EInt;
    attribute periodicity : Periodicity = 'Periodic';
    attribute characteristicPeriod : ecore::EInt;
    property partition#tasks : Partition;
    invariant PositiveWCET: worstCaseExecTime > 0;
    invariant ExecutionAndDeadlineAllowsCompletion:
        worstCaseExecTime <= implicitDeadline;
    invariant ExecutionAndPeriodAllowsCompletion:
        if (periodicity <> Periodicity::Aperiodic)
            then worstCaseExecTime <= characteristicPeriod
        else true
            endif;
    invariant DeadlineLessThanPeriod: implicitDeadline <= characteristicPeriod;
    invariant PositivePeriod: characteristicPeriod > 0;
}
```

Constraint Details

`PositiveWCET` The worst case execution time of a task must be a positive number.

`ExecutionAndDeadlineAllowsCompletion` The WCET must be less than or equal to the implicit deadline otherwise the task will never complete in time.

`ExecutionAndPeriodAllowsCompletion` The WCET must be less than or equal to the period of the task. This is only a constraint on *aperiodic* tasks: tasks without a minimum time between arrivals.

`DeadlineLessThanPeriod` The deadline of the task must be less than or equal to the period otherwise multiple instances of the task will exist simultaneously.

`PositivePeriod` The period of a task must be a positive number.

3.1 Complex Validation

4 Implementation

This section describes Java code and Eclipse specific implementation details of the plug in. The plug in is composed of 5 Eclipse projects, 4 generated from the Ecore model and 1 from the Sirius specification. See figure 3 for the list of projects.

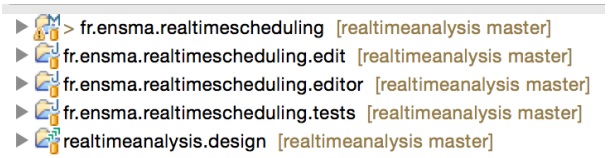

Figure 3: Plug in projects

Most of the non-generated code resides in `fr.ensma.realtimescheduling`, specifically the analysis algorithms, visualization classes, and the meta-model interfaces and implementations. This project will be refered to as the **main** project.

The projects `edit`, `editor`, and `test` contain *generated* code for editing the model (operations such as instance creation, or attribute modification), the editor GUI (see figure 5), and test code respectively. The test project is currently not used.

Several Eclipse extensions are also declared in the `plugin.xml` within the main project (see figure 6). These mainly consist of UI extensions to allow for easier manipulation and analysis of a model.

4.1 Main Project

The main project contains the analysis algorithms and non-trivial UI logic code for the project. The package structure of this project is given in figure 4. Several of these packages deserve special attention.

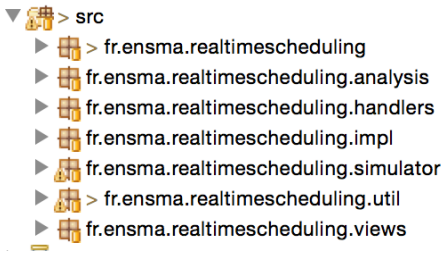

Figure 4: Main project packages

analysis This package contains the algorithms and "glue" code between the model and the algorithms. The `Analyzer.java` class holds the implementations of all the algorithms described in the CORAC-PANDA project: the response time analysis from chapter 2 and the network delay analysis from chapter 3. The code was written attempting to recreate as much as possible the structure and names of the pseudo-code from the report.
The `ModelInterface.java` class contains methods to interface between the UI and the analysis algorithms.
The rest of the classes in the package provide side-effect free functions for manipulating the model or classes to store auxilary data necessary by the algorithms. For example, the Ecore list implementation `EList` is not modifiable thus it cannot be passed to Java Collections methods such as `Collections.sort()`. Having a list of time-sorted `Intervals` is necessary for the algorithms thus the `PartitionUtils.java` file provides such a method by creating a copy of the original list.

handlers This package contains handler definitions for the commands generated by Eclipse. UI actions in Eclipse are handled by a command-handler relationship. More information on this mechanic can be found online.

views This package contains the classes used to layout and display data to charts from the model. This project uses the **JFreeChart** library to generate charts like line charts and bar charts. The classes `AbstractLineChart.java` and `AbstractBarChart.java` set up the layout of the charts which is a horizontal layout with a chart on the left and a control component on the right.

util This package is generated by Ecore, however it houses the `RealtimeschedulingValidator.java` class which contains the complex validation code that could not be expressed in OCLinEcore. The specifics of the validation are in section 3.

4.2 Edit

This project contains generated code for modifying elements of a model instance. The only modifications to this code are found in the `getText()` method of some of the instances which makes the display in the editor more user friendly. See figure 5.

4.3 Editor

This project defines the generated classes for a simple model editor which only displays a tree representation of the model.

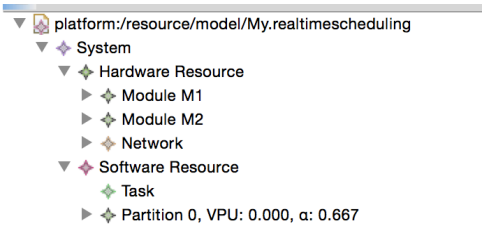

Figure 5: Model editor in use

4.4 Sirius Editor

4.5 Eclipse Extensions

The plug in provides several custom Eclipse extensions apart from the generated ones.

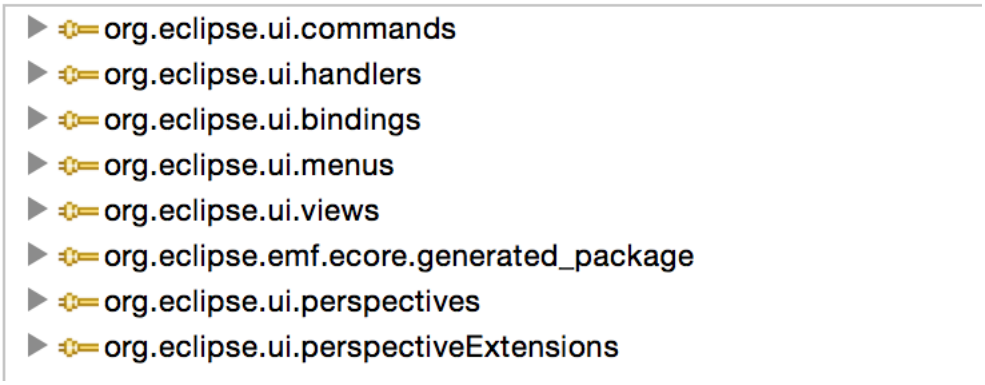

Figure 6: Plug in extensions

Menu Menu extensions add items to existing menus. In this case, I added a menu item to the main Eclipse menu bar (see figure 7). This is indicated by the root **Menu Contribution** element of the extension which indicates that this menu should be added to the menu specified by `menu:org.eclipse.ui.main.menu`.

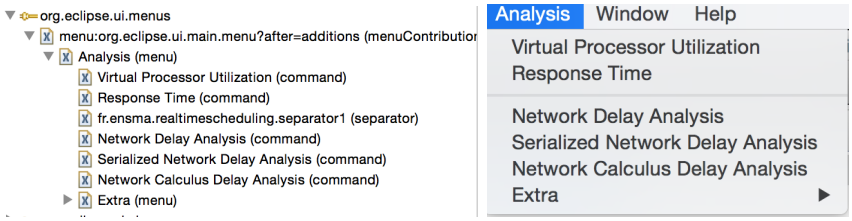

Figure 7: Menu extension definition and result

View Several new views are defined in the project. These are mainly for graphing the information derived from the analysis of the model.

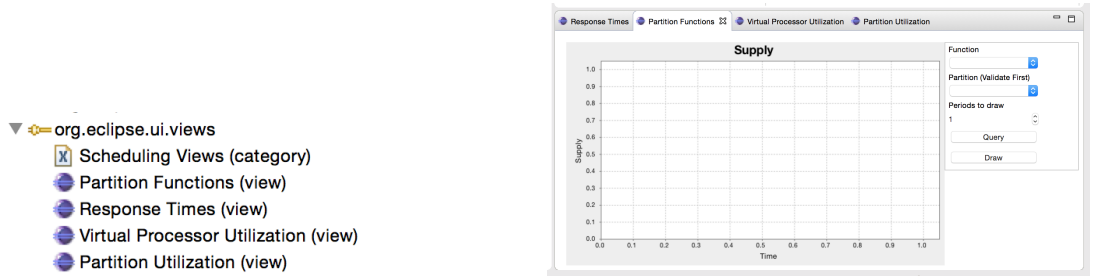

Figure 8: View extension defition and result

Perspective One new perspective is declared with the purpose of arranging the model editor and the graphing views into a coherent layout. New perspectives can be defined by first creating an extension of `org.eclipse.ui.perspective` and then of `org.eclipse.ui.perspectiveExtensions`. The first is a manifest to Eclipse that this plug-in defines a new extension; the second allows you to define the layout of the extenstion from within the `plugin.xml` editor.

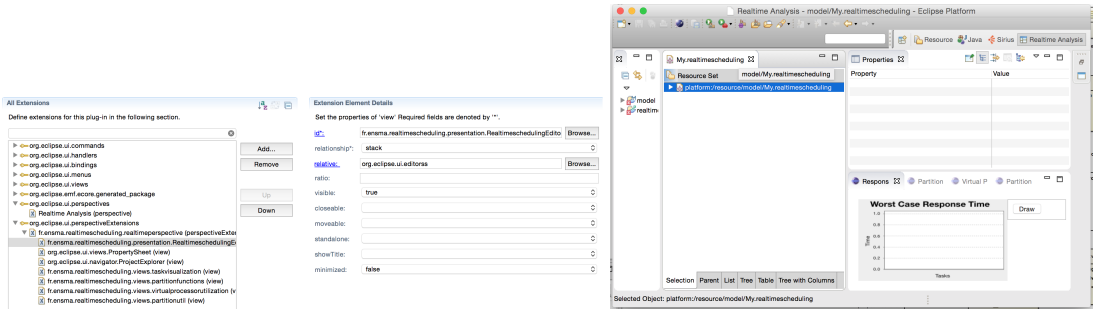

Figure 9: New perspective

5 Example

This section shows an example model created using the plug-in. I describe how to create it and how to analyze it.

6 Installation

6.1 From Source

Move the 5 projects into your Eclipse workspace. You can then highlight the 5 projects, right-click on one of them, select **Run As** then **Eclipse Application**.

6.2 From JARs

7 Future Work

This section describes what future work needs to be done to improve the project and the steps required to start that work.