ETH
Eidgenössische Technische Hochschule Zürich
Swiss Federal Institute of Technology Zurich

TIK Institut für Technische Informatik und Kommunikationsnetze

Semester Thesis
at the Department of Information Technology
and Electrical Engineering

Software Development Environment for Many-Core Systems

AS 2011

Etienne Geiser

Advisors: Lars Schor
Devendra Rai
Professor: Prof. Dr. Lothar Thiele

Zurich
30th March 2012

Abstract

Working with the new Distributed Application Layer (DAL) framework poses various difficulties due to the lack of DAL optimized tools. In this thesis, an editor is designed and implemented, which simplifies the creation and modification of DAL projects. Due to the availability and extensibility of the Eclipse development environment, the tool is implemented as an Eclipse plug-in. The resulting editor provides the programmer a set of tools to specify DAL applications as process networks and their interactions as a Mealy machine. The modular structure of the editor allows to easily extend the tool such that an even more powerful tool can evolve in the future.

Acknowledgements

I would like to express my sincere gratitude to Prof. Dr. Lothar Thiele for granting me the opportunity to write this semester thesis in his research group.

For their continuous support and great help during this semester thesis, I would like to extend my thanks to my advisers Lars Schor and Devendra Rai. Without them I would not have been able to complete it.

Last but not least I would like to thank my family for their never-ending support and motivation.

Contents

List of Figures

1 Introduction

1.1 Motivation

Current trends in CPU development include shifting from single-core designs towards having many cores on a single chip. For example, Intel's new experimental "Single-chip Cloud Computer" has 48 cores on a single chip. However, we cannot make good use of these multiple cores with conventional single threaded programming. Multi-threaded programmes are a much better approach in this aspect, but coding such programmes is considerably more difficult. Even more experienced programmers lose the overview if an application is composed of more than a handful of threads. The Distributed Application Layer (DAL) [1] is a new methodology to make programming of concurrent applications easier. The goal of this thesis is to create a graphical editor for specifying DAL applications and their interactions.

1.2 DAL Framework

Application Specification

Applications are specified as process networks (pn). Each process has its source code where the behaviour is defined. Those files are written in c or c++. The communication between processes is handled by software channels, Fig. 1.1 shows an example of a process network.

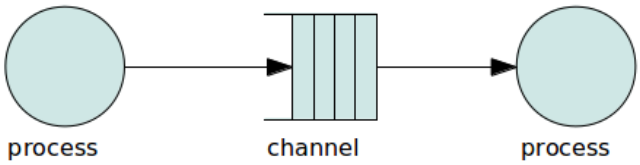


Figure 1.1: An example of a simple process network.

Interactions between Applications

The interactions between applications are handled as a finite state machine (fsm), where each state represents an execution scenario. From these scenarios, transitions lead the fsm to the next scenario. Apart from an origin and a target, a transition comprises of events and actions. The events define when the transition is fired (the fsm goes to the next scenario). Actions define what to do, e.g. start or stop an application. For an example of a finite state machine, refer to Fig. 1.2.

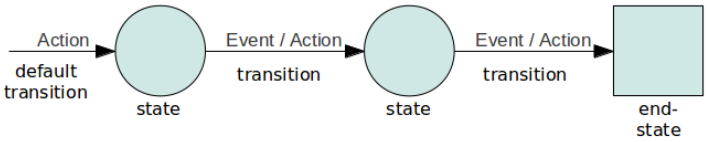


Figure 1.2: An example of a simple finite state machine.

Input Specification Format in DAL

DAL makes use of its own XML (Extensible Markup Language) semantic to save process networks and finite state machines. XML defines a set of rules for encoding documents in a format that is readable for both humans and machines. It makes use of tags to specify the start <construct> and the end </construct> of a construct. In DAL, examples of such constructs are states, processes or connections. In Fig. 1.3, an example of such an XML code is depicted.

Using this format comes with a few disadvantages. With increasing complexity of the system, it becomes hard to keep the overview. Also, when writing plain XML, all tags have to be typed by hand. If we use a visual editor, which automatically creates these tags, we cannot only reduce the amount of work, but also the number of typing errors.

```
<?xml version="1.0" encoding="UTF-8"?>
<fsm xmlns="http://www.tik.ee.ethz.ch/~euretile/schema/FSM" name="example">
  <application name="app1" src="app1/pn.xml"/>
  <transition name="DefaultTransStart-run" nextstate="run">
    <action action="DAL_START" application="app1" params=""/>
  </transition>
  <state name="run">
    <transition name="trans_0" nextstate="pause"/>
    <transition name="trans_1" nextstate="End_State">
      <action action="DAL_STOP" application="app1" params=""/>
      <event name="stop"/>
    </transition>
  </state>
  <state name="pause">
    <transition name="trans_2" nextstate="End_State"/>
  </state>
  <state name="End_State"/>
  <endstate name="End_State"/>
</fsm>
```

Figure 1.3: An example of a finite state machine written in DAL XML.

1.3 Related Work

There are already various programmes which are used to manipulate process networks or finite state machines. In the following the ones most often used by DAL programmers, namely Moses [2] and Matlab [3], are briefly mentioned.

Moses

Moses is a tool to create process networks. Its biggest disadvantages are that it does not support the creation of finite state machines and the editing of existing process networks. [2]

Matlab

Matlab does not have the same weaknesses as Moses, but it carries a hefty licence fee. Furthermore Matlab saves the finite state machines in its own format (.mdl). There is a script available to convert .mdl files into DAL XML code, but it is an additional step to take. [3]

1.4 Editor Requirements

The requirements of the desired editor are as follows:

- To improve the user satisfaction while working with DAL, the editor needs to be a graphical editor.
- The editor has to support the creation and modification of finite state machines and process networks.
- The source code of a process has to be editable.
- The finite state machines and process networks have to be saved in a DAL compliant XML file.

1.5 Outline

In Chapter 2, a short introduction is given about the tools used to create the editor. Chapter 3 focuses on the implementation of the plug-in. Chapter 4 wraps-up the paper and gives an outlook.

2

Libraries to Develop an Eclipse Plug-in

The widely used Eclipse SDK was chosen as tool to design a graphical editor for DAL. Eclipse was selected because it is an open-source SDK and it possesses a highly extensible plug-in system. Many Eclipse libraries exist, which support the creation of new plug-ins. In this thesis the Eclipse Modelling Framework Project (EMF) [4] and Graphical Editing Framework (GEF) [5] were used to develop the plug-in. Both libraries are discussed in the following.

2.1 Eclipse Modelling Framework Project

The EMF project is a modelling framework and code generation facility for building tools and other applications based on a structured data model. One such generated tool allows the model to be converted into XML. EMF was used to model the process network and the finite state machine which are the foundations of the created editor. [4]

2.2 Graphical Editing Framework

GEF provides technologies to create rich graphical editors and views for the Eclipse Workbench UI. Using GEF allows to create an editor with a minimal effort and enables to focus on the functionality of the editor, instead of having to spend a lot of time designing the user interface. [5]

3 Plug-in

In the first section of this Chapter the structure of the Plug-in is explained. The second part is about the features of the editor implements. Finally, the most important components of the user interface are explained based on a screen shot.

3.1 Structure

As outlined in Fig. 3.1, the plug-in consists of multiple parts. These parts can be split into a model part and an editor part. The model part includes the models of the finite state machine and the process network, and their respective support classes, for editing and presenting the models. The editor part provides the graphical interface and is made up out of three sub parts. Firstly, there is a more general DAL part. This part is the entry point of the plug-in. Here it is decided if the opened file is a finite state machine or a process network, and which sub editor should be opened. It also provides a wizard, which allows the user to easily create a new DAL project.
The second and third part are the sub-editors for the finite state machine and the process network. These sub-editors provide specialized classes, which are needed to handle the different models.

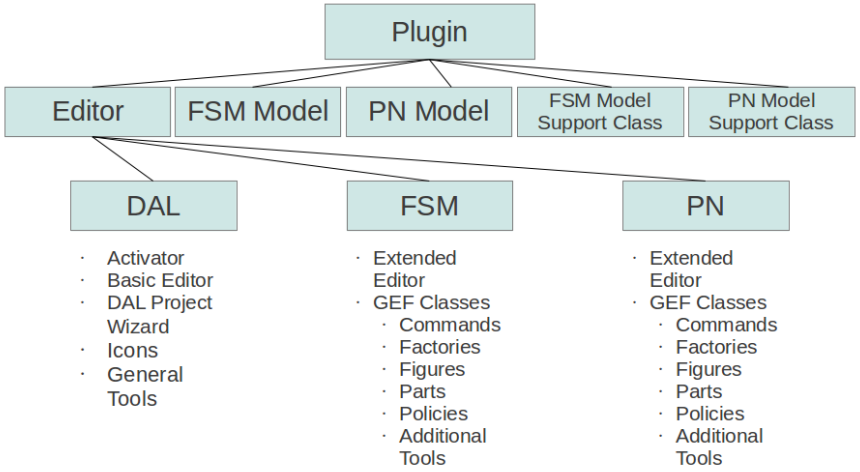


Figure 3.1: The hierarchy of the plug-in.

3.2 Features

The provided editor fulfils the requirements stated in section 1.4. In addition the editor provides a few further features. One is that the editors for the finite state machine and for the process network appear as one editor. There is no need to select the right editor when the user opens a file. Furthermore, the editor automatically creates the skeleton when a new source file is created. There are also some built-in auto-check features. For example, a channel has to have exactly one input and one output connection. Also connections between two processes or two channels are not allowed. Furthermore, a finite state machine needs to have a default transition and an end-state.

3.3 Layout

The interface of the plug-in consists of the following parts, which are also highlighted in Fig. 3.2.

Action Bar:
Provides buttons for undo, redo and delete.
Editor Field:
Here you see the opened process network or finite state machine.
Editor Tabs:
For each editor that is opened, a new tab is added to the list.

Palette:
Provides the tools to create processes, states, channels, etc.. Only the tools for either a process network or finite state machine are shown.
Projects:
List of projects in the workspace.
Selected Object Properties:
View for more specific properties of the object selected in the editor field.

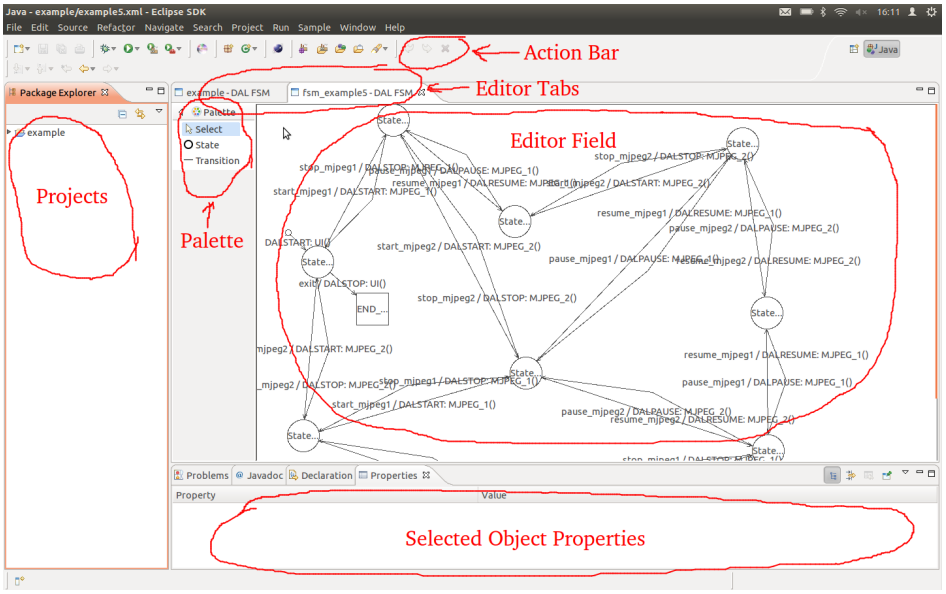


Figure 3.2: The layout of the editor.

4

Conclusion and Outlook

4.1 Conclusion

In this thesis, a graphical editor for DAL projects was created. The editor fulfils the requirements defined in Chapter 3.1, and provides the basic tools needed to create and modify DAL projects. Thanks to the modular build of the plug-in additional functionality can easily be added in the future. This allows to improve the editor further and achieve an even better user satisfaction.

4.2 Outlook

There are a number of possibilities to extend the editor. First, the editor could be extended with more features offered by DAL, for example the iterator [6]. Or the amount of auto-check could be increased. Another example would be to check in a finite state machine if the running applications are the same for all possible ways of reaching a certain state. More possibilities to enhance the editor are listed in Appendix C.

A

Plug-in Manual

A.1 Creating a new DAL Project

Click in the menu bar on *File>New>Other* or press *Ctrl+N*. Search for *DAL Project* and click *Next*. Here you can enter the name of your new project. See Fig. A.1 for reference.
Pressing the *Finish* button will create your new project. The second page is shown in Fig. A.2.

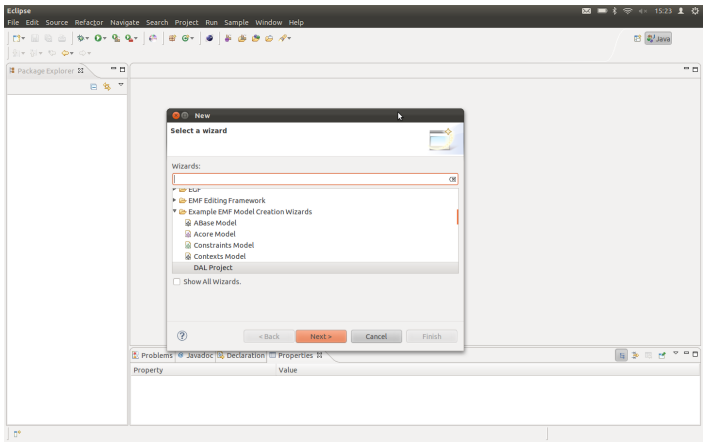


Figure A.1: First page shown after opening a DAL project wizard.

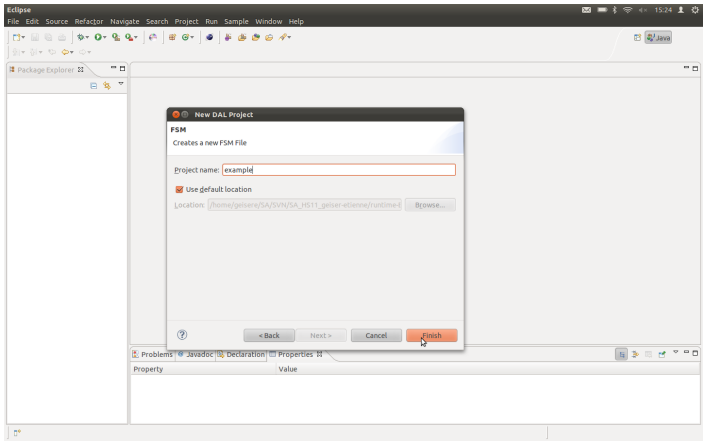


Figure A.2: Second page of the DAL project wizard.

A.2 Editing a Finite State Machine (fsm)

A.2.1 Creating a new State

A new state is created by left clicking on *State* in the palette and clicking left again on to the editor field. Now you have the option to name your state, otherwise the state will be named with a generic name. In Fig. A.3 an example is given.

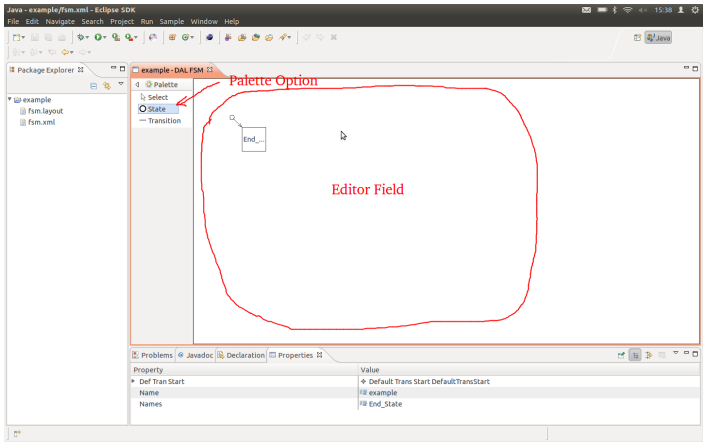


Figure A.3: Creating states.

A.2.2 Renaming a State

To rename a state you have to select a state by left clicking on it. When you left click on the state again, an editor for the name is opened. Please note that this is not a double-click, you need to wait a short time between the two clicks.

A.2.3 Connecting States

To connect two states, you first have to choose the *transition* tool by left clicking on the Transition option in the Palette. Then you left click on the state you want to be the origin, afterwards you click on the target state. Note, that the end-state cannot be the origin of a transition. You should also be aware that the fsm default-state can neither be target nor origin of a transition. The only exception is the default transition which can only be reconnected to another target, but not newly created or deleted. See Fig. A.4 for an example of how to connect states.

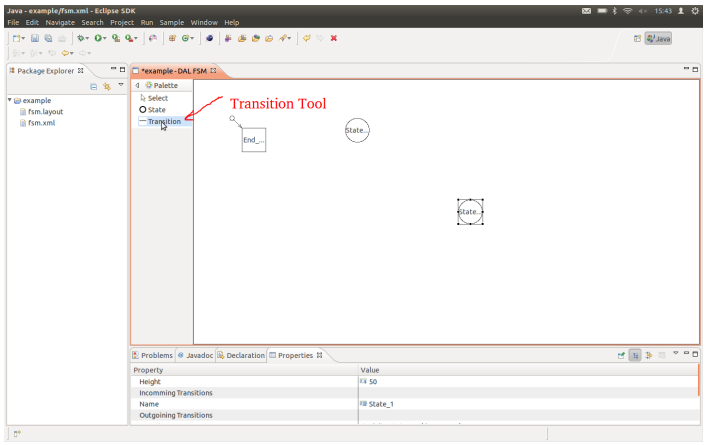


Figure A.4: Connecting states.

A.2.4 Moving a State

The states can simply be moved by drag and drop. Left click on the state, keep the mouse button pressed and then move it to the desired position.

A.2.5 Changing Transition Endpoints

To change the start- or endpoint of a transition you have to select the transition (left click). Two squares appear, one at the beginning and one at the

end of the transition. Simply drag and drop the square to the desired state. The squares are marked in Fig. A.5.

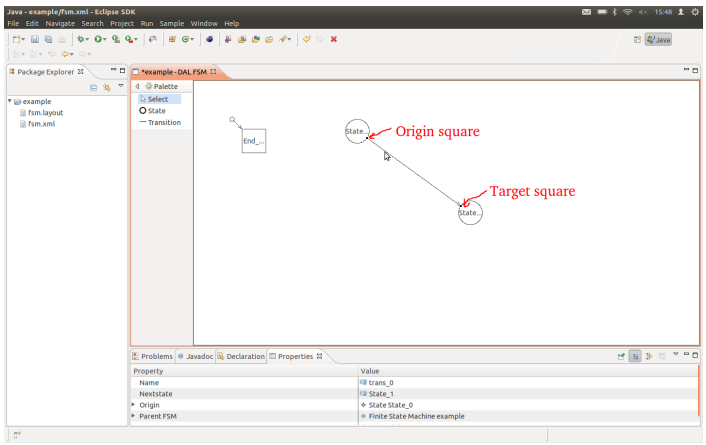


Figure A.5: Reconnecting a transition.

A.2.6 Deleting an Element

An element (state, transition) can be deleted by selecting the element and then clicking on the *delete* button in the action bar. The end-state and the default-transition (small point and the transition coming from it) can NOT be deleted. Fig. A.6 shows an example.

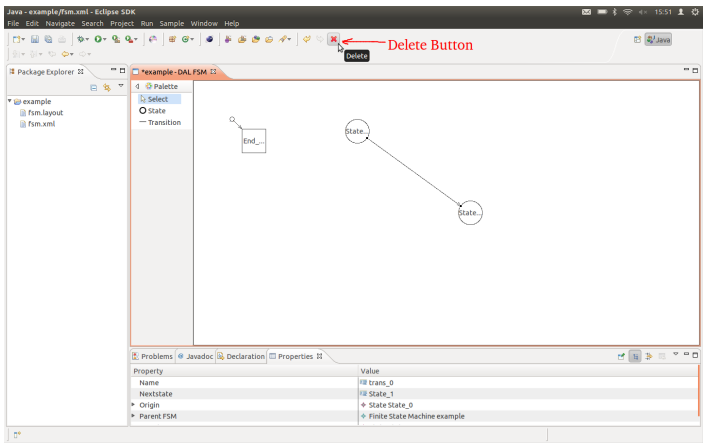


Figure A.6: Deleting a state.

A.2.7 Adding and Deleting an Application

To get to the manage window for the application, you have to right click in the editor window and choose *Manage Applications*. A new window is opened, see Fig. A.7 for a screenshot. Here you have a field for the name and a field for the path and file name of the application. The path is added to the path of the directory where the .xml file with the fsm is saved. Furthermore you have the options *Add*, *Remove*, *Create* and *Open*:

- *Add*: Here you add an existing application to the fsm. The function only adds a link, so the path needs to lead to a .xml file, in which an application is specified. For creating a new application use *Create* (see below).

- *Remove*: With this button the selected applications can be deleted. This includes removing them from the list of applications and removing all actions involving the selected applications.

- *Create*: Creates a new application. A new .xml file specified by the source path is created and then added to the list of applicable applications.

- *Open*: Opens the selected applications with the DAL editor.

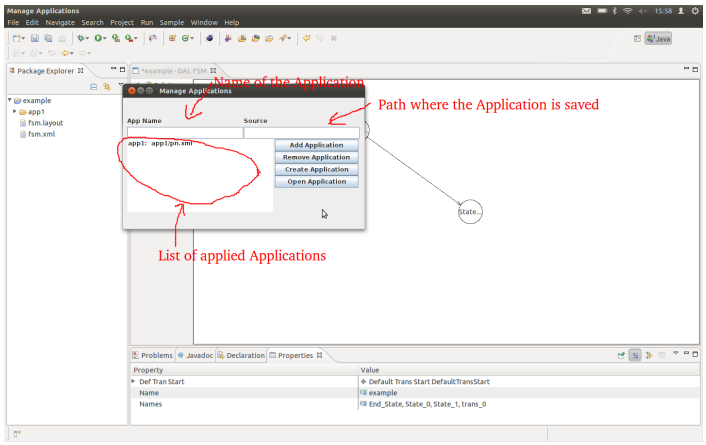


Figure A.7: Managing applications.

A.2.8 Adding and Removing Events of a Transition

In order to add an event to a transition you have to select the transition and then right click on it. Now, choose *Manage Events* and a new window will open up, where you can add and remove events to/from the selected transition. Fig. A.8 shows what the menu looks like.

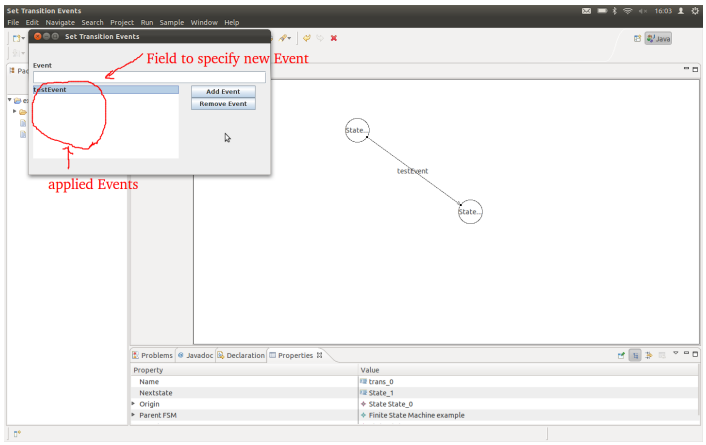


Figure A.8: Managing events.

A.2.9 Adding and Removing Actions of a Transition

Select the desired transition, right click on it and choose *Manage Actions*. To add an action to a transition you have to select the action and the application which it should affect. Furthermore, you have a field where you can add optional parameters. In Fig. A.9 you see an example.

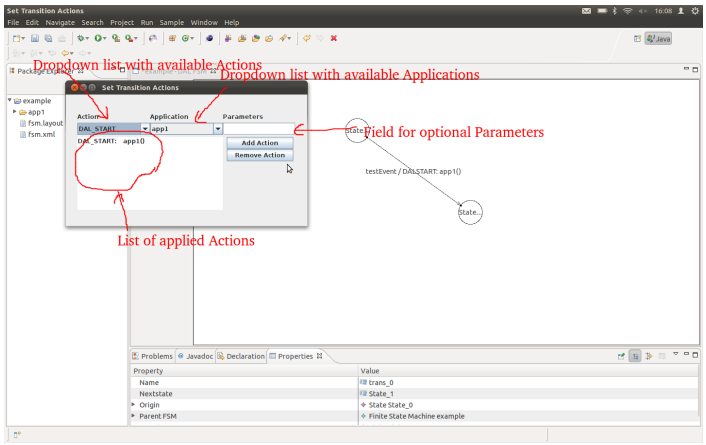


Figure A.9: Managing actions.

A.3 Editing a Process Network

A.3.1 Creating a new Process

As outlined in Fig. A.10, a new process is created by selecting *Process* in the palette and then clicking on the editor field. Now you have the option to name your process otherwise the process will be named with a generic name.

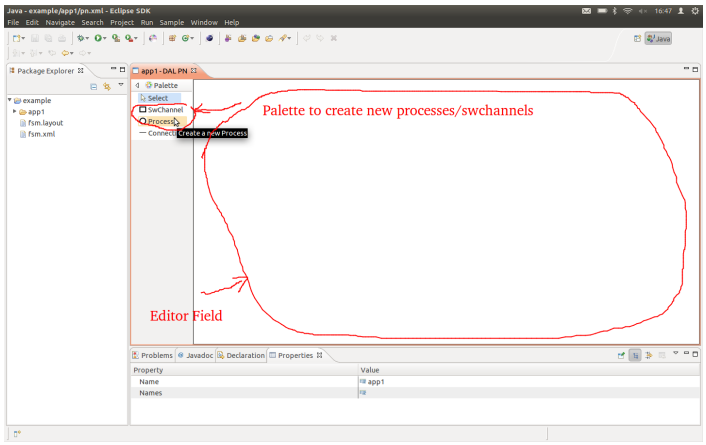


Figure A.10: Creating new processes.

A.3.2 Creating a new Channel

Click on *SwChannel* in the palette and then click again on the editor field. You have now the option to name your channel otherwise the channel will be named with a generic name. Refer to Fig. A.10 for an illustration of this step.

A.3.3 Renaming a Process/Channel

To rename a process/channel you have to select it by left clicking on it. When you left click on the state again an editor for the name is opened. Note that this is not a double-click, you need to wait a short time between the two clicks.

A.3.4 Creating Connections

Connections are created using the Connection tool in the palette. After selecting the Connection tool, click first onto the origin of the connection

and then onto the target. Connections can only be made between processes and channels. It is not possible to make a connection between two processes or two channels. Furthermore a channel can only have one input and one output connection. See Fig. A.11 for guidance.

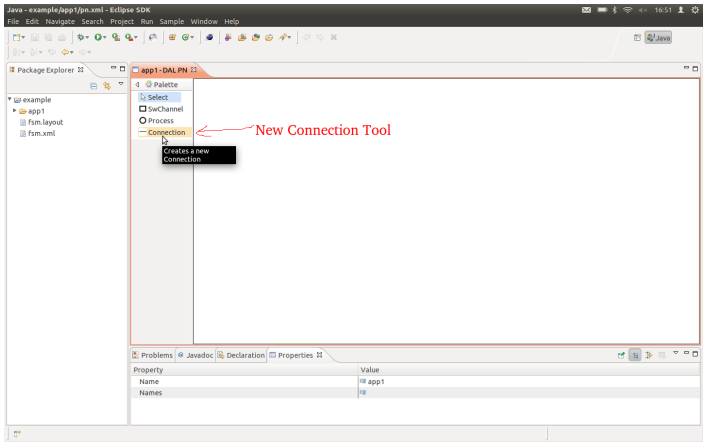


Figure A.11: Creating connections.

A.3.5 Moving a Process/Channel

Processes and channels can simply be moved by drag and drop. Left click on the state, keep the mouse button pressed and then move it to the desired position.

A.3.6 Changing Connection Endpoints

To change the start- or endpoint of a connection you have to select the connection (left click). Two squares will appear, one at the beginning and one at the end of the transition, as shown in Fig. A.12. Simply drag and drop the square to the desired state.

A.3.7 Deleting an Element

An element (process, channel, connection) can be deleted by selecting the element and then clicking on the *delete* button in the action bar. Fig. A.13 demonstrates the deleting of an element.

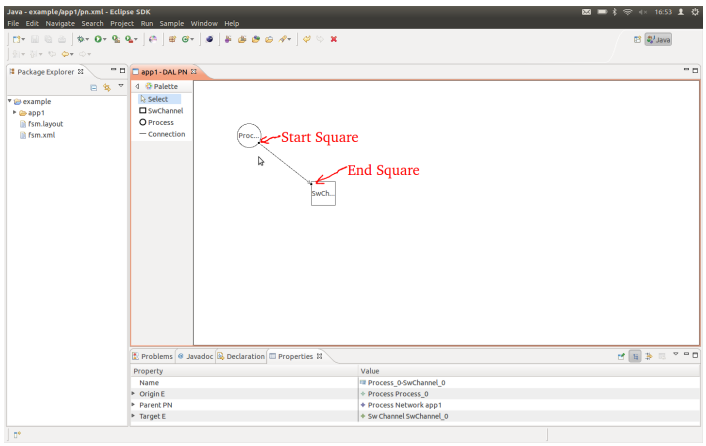


Figure A.12: Redirecting connections.

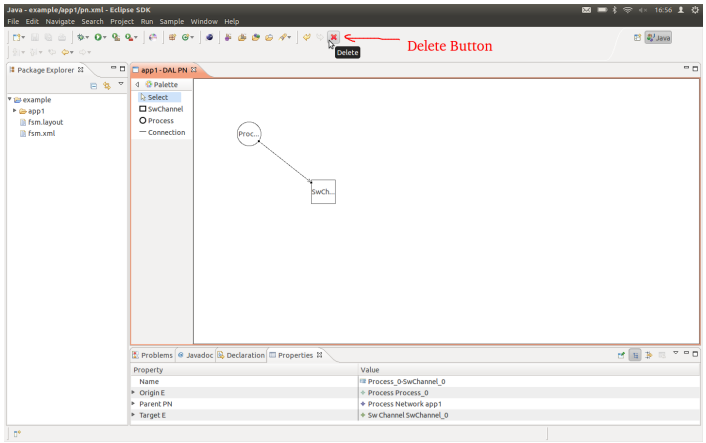


Figure A.13: Deleting a process.

B

Technical Data

Used programmes: Eclipse SDK 3.7.1

Programmed in: JavaSE-1.6

Dependencies:

Eclipse Modelling Framework (EMF) 2.7.1
Graphical Editing Framework (GEF) 3.7.1

Operation System: Ubuntu 11.10 64-bit

C

Further Development Possibilities

While working on the project, various ideas for further projects and research came up. Some of these suggestions are listed here.

- Implement an overlay protection, so that multiple items cannot be at the same place.
- Improve the routing (drawing) of connections/transitions from simply drawing a straight line to evading states, processes, etc..
- Add zooming and scrolling options for the editor.
- Allow the grouping of a part of the fsm/pn, and only show a place holder. For example, if a sub-function involves multiple processes, group them and only show one object with the function name.
- Integrate copy and paste of parts of the fsm/pn, possibly also from sources outside the current editor.
- Extend the editor with the DAL specified iterator.
- Where no layout file can be found when opening a fsm/pn file, use a smarter way than random to place the elements (process, state...).
- Remove duplicated code in pn connection create command.
- In pn connection create command, try to improve the port adding code.
- Offer snapping elements/connections to grid possibility.

- When saving a fsm/pn, automatically add comments into the .xml file. These comments should structure the .xml file and improve its readability.
- Try to clean up the pop-up menu when right clicking. In some cases there are options from other eclipse plug-ins.
- Improve the handling of the source code of a process:
  - Implement a new way to create a new source file, maybe using a wizard. For example, show a file system, where an existing one can be chosen or a new one created.
  - After setting a new source it shouldn't be necessary to defocus and refocus the process any more.
  - Allow the renaming of a source file.
  - If a process has been renamed offer the option to rename the source file too.
- Enhance comments in plug-in code (function description, etc.).
- Add short-keys (fsm and pn), e.g. a short-key for the state creation tool.
- Check the opened file for validity, and then either correct them or stop the editor.
- Add icons (palette, editor page, etc.).
- Save constants, for example the default dimension, in a more centralized manner.
- Change the Palette name (e.g. fsm/pn Palette).
- fsm: Check if the final running/paused/etc. applications are the same for different ways of getting from state A to state B.
- fsm: Manage application: To add an application, a window showing the file system should be opened. Allow to choose from the existing applications.
- fsm: Overhaul the FSMConnectionRouter class, which uses deprecated objects.

D

Presentation Slides

ETH
Eidgenössische Technische Hochschule Zürich
Swiss Federal Institute of Technology Zurich

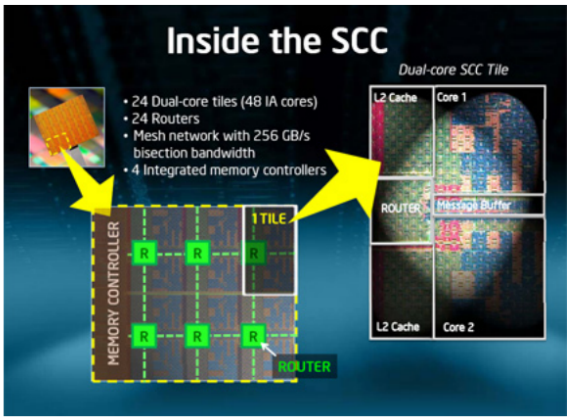
Software Development Environment for Many-Core Systems

Semester Thesis by Etienne Geiser
Advisors: Lars Schor and Devendra Rai

ETH
Eidgenössische Technische Hochschule Zürich
Swiss Federal Institute of Technology Zurich

Motivation

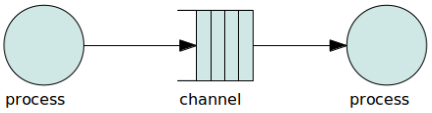
- Software development framework
- Designed for parallel programs on many-core architectures



ETH
Eidgenössische Technische Hochschule Zürich
Swiss Federal Institute of Technology Zurich

Distributed Application Layer (DAL): PN

- Applications are specified as Kahn process networks

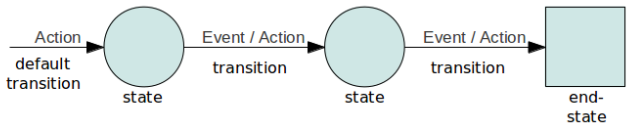


- Processes have source code, where their behaviour is defined
- Processes communicate through FIFO channels

ETH
Eidgenössische Technische Hochschule Zürich
Swiss Federal Institute of Technology Zurich

Distributed Application Layer (DAL): FSM

- The interactions between the applications are specified as a finite state machine



- Each state represents a scenario with a set of applications running or pausing
- Actions are manipulations of applications
  e.g. starting, stoping, pausing, resuming
- Events are the triggers for state transitions

ETH
Eidgenössische Technische Hochschule Zürich
Swiss Federal Institute of Technology Zurich

Problem Description

- Create a graphical software environment to develop DAL systems
- One tool which supports both PN and FSM

- Implemented as an Eclipse plug-in

ETH
Eidgenössische Technische Hochschule Zürich
Swiss Federal Institute of Technology Zurich

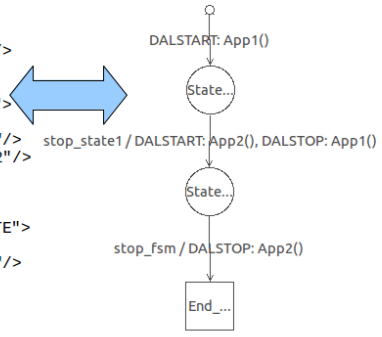
Current DAL Specification Format

- DAL Process Networks and Finite State Machines are saved in XML files
- Hard to keep overview over bigger systems
- Much „overhead“ typing to specify each state/transition....
- Less code to write => less work + less typos

ETH
Eidgenössische Technische Hochschule Zürich
Swiss Federal Institute of Technology Zurich

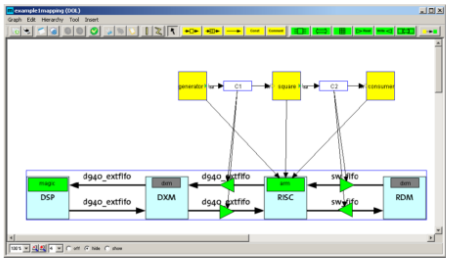
XML Example

```
<fsm>
  <application name="APP1" src="app1/pn.xml" />
  <application name="APP2" src="app2/pn.xml" />
  <transition name="trans_1" nextstate="State_1">
    <action action="DAL_START" application="APP1"/>
  </transition>
  <state name="State_1">
    <transition name="trans_2" nextstate="State_2">
      <event name="stop_state1"/>
      <action action="DAL_STOP" application="APP1"/>
      <action action="DAL_START" application="APP2"/>
    </transition>
  </state>
  <state name="State_2">
    <transition name="trans_3" nextstate="END_STATE">
      <event name="stop_fsm"/>
      <action action="DAL_STOP" application="APP2"/>
    </transition>
  </state>
  <state name="END_STATE"/>
  <endstate name="END_STATE“/>
</fsm>
```



ETH
Eidgenössische Technische Hochschule Zürich
Swiss Federal Institute of Technology Zurich

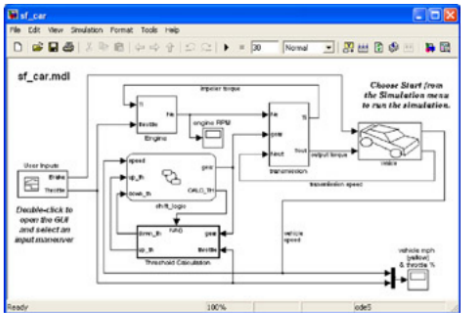
Existing Tools: Moses



- Does not support modification of existing PN
- No FSM

ETH
Eidgenössische Technische Hochschule Zürich
Swiss Federal Institute of Technology Zurich

Existing Tools: Matlab



- Costly License
- FSM are saved as .mdl (script available to convert to XML)

ETH
Eidgenössische Technische Hochschule Zürich
Swiss Federal Institute of Technology Zurich

„Optimal“ Tool

- Requirements:
  - Graphical Tool
  - Allows creating and editing of FSM and PN
  - Easy way to edit the code of a process
  - Output in DAL compliant XML schema

ETH
Eidgenössische Technische Hochschule Zürich
Swiss Federal Institute of Technology Zurich

Why Eclipse?

- Widely used
- Open-source SDK
- Highly extensible plug-in system
- Provides frameworks to create custom editors

ETH
Eidgenössische Technische Hochschule Zürich
Swiss Federal Institute of Technology Zurich

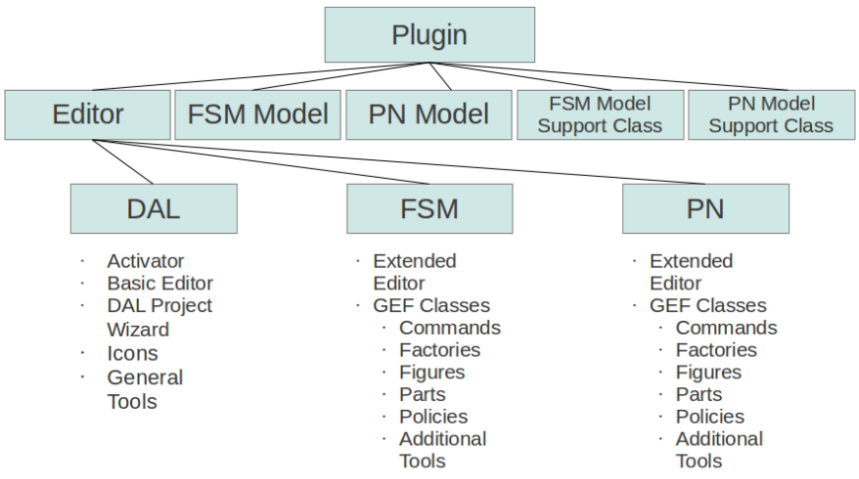
Eclipse: EMF & GEF

- EMF(Eclipse Modeling Framework):
  - Modeling framework and code generation facility

- GEF(Graphical Editing Framework):
  - For graphical editors and views for the Eclipse Workbench UI

ETH
Eidgenössische Technische Hochschule Zürich
Swiss Federal Institute of Technology Zurich

Plugin Structure



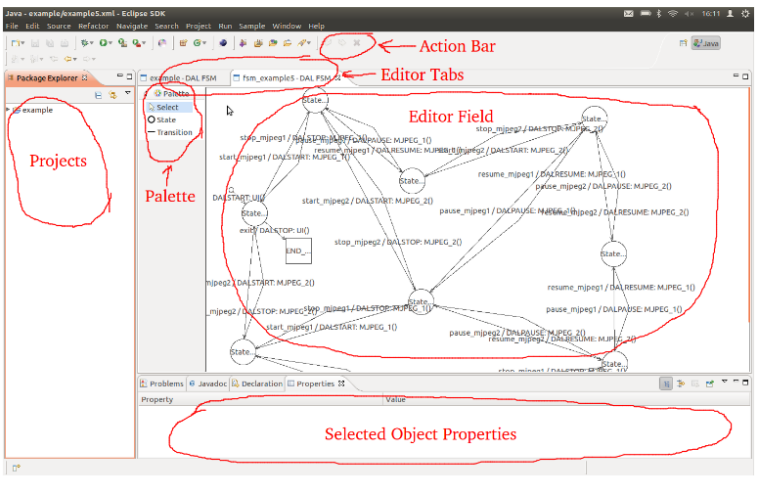
ETH
Eidgenössische Technische Hochschule Zürich
Swiss Federal Institute of Technology Zurich

Plug-in Features

- Additional features:
  - All in one editor
  - Automatic skeleton generation for process code files
  - „Auto Check“:
    - Channel can only have one input and one output
    - Connections only allowed between process and channel
    - Default transition and endstate have to exist in a FSM otherwise, they are created

ETH
Eidgenössische Technische Hochschule Zürich
Swiss Federal Institute of Technology Zurich

Plug-in Layout

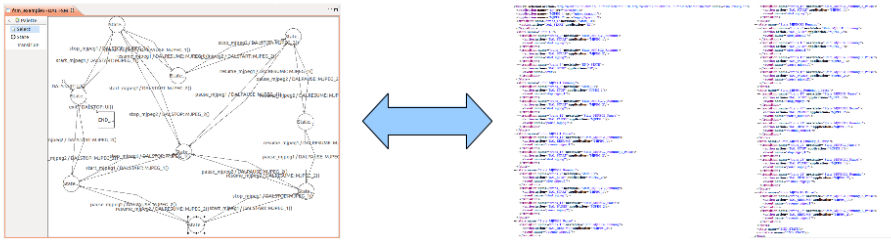


ETH
Eidgenössische Technische Hochschule Zürich
Swiss Federal Institute of Technology Zurich

Demo

ETH
Eidgenössische Technische Hochschule Zürich
Swiss Federal Institute of Technology Zurich

Conclusion



- Designed a visual software development enviroment for the DAL tool chain.
- Features:
  - „Drag and Drop“ of process networks and finite state machines
  - Automatic creation of source file skeletons
- Implemented as Eclipse plug-in

Bibliography

[1] EURETILE, "Distributed Application Layer," Mar. 2012. [Online]. Available: http://www.tik.ee.ethz.ch/~euretile/dal.php

[2] ETH, "Modeling, Simulation, and Evaluation of Systems," Mar. 2012. [Online]. Available: http://www.tik.ee.ethz.ch/~moses/

[3] MathWorks, "Matlab," Mar. 2012. [Online]. Available: http://www.mathworks.com/products/matlab/index.html

[4] E. Foundation, "GEF (Graphical Editing Framework)," Mar. 2012. [Online]. Available: http://www.eclipse.org/gef/

[5] ——, "Eclipse Modeling Framework Project (EMF)," Mar. 2012. [Online]. Available: http://www.eclipse.org/modeling/emf/

[6] L. Thiele, I. Bacivarov, W. Haid, and K. Huang, "Mapping Applications to Tiled Multiprocessor Embedded Systems," in *Proc. Int'l Conf. on Application of Concurrency to System Design (ACSD)*, Jul. 2007, pp. 29–40.