Eidgenössische Technische Hochschule Zürich
Swiss Federal Institute of Technology Zurich

Institut für
Technische Informatik und
Kommunikationsnetze

Semester Thesis
at the Department of Information Technology
and Electrical Engineering

Advanced Features for a Software Development Environment for Many-Core Systems

Spring 2012

Erwin Herrsche

Advisor: Lars Schor
Professor: Prof. Dr. Lothar Thiele

Zurich
27th July 2012

Abstract

Working with the Distributed Application Layer (DAL) has been improved with the creation of the DAL Eclipse plug-in (DALipse). This thesis extends the existing plug-in with required features to provide a complete cockpit for DAL. In particular, the contributions of this thesis are an editor for DAL architecture specifications and advanced features for the existing finite state machine editor.

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.

I would also like to thank my advisor Lars Schor for his support during the whole process of this thesis.

Contents

List of Figures

1 Introduction

1.1 Motivation

Today's multicore architectures are relatively simple. It is likely that the complexity of hardware architectures will drastically increase in the next few years. For example the Intel SCC prototype architecture [1] has already 48 cores. From a system level perspective, it consists of 145 different elements. Future architectures will consist of hundreds of cores. The creation of such complex abstract models can easily lead to errors. A graphical editor simplifies the modelling process and the amount errors can be reduced.

1.2 Distributed Application Layer

The Distributed Application Layer (DAL) [2] is developed as part of the EURETILE project[1]. It is based on the DOL framework [3]. As in the DOL framework, applications are specified as Kahn process networks [4]. In addition, execution scenarios are organized as a Moore finite state machine. In each scenario, each application is either running, paused or inactive. Transitions between scenarios are triggered by either events generated by running applications or the run-time system. Hardware platforms are modelled in three different hierarchical levels, namely processors, local buses (shareds) and network-on-chips (NoCs).

[1] `http://www.tik.ee.ethz.ch/~euretile/`

1.3 Contributions

- A graphical editor for DAL architectures is designed and implemented.
- The application and transition event management for DAL finite state machines is improved.

1.4 Related Work

There are projects that do a similar job as the DALipse, e.g. HOPES[2] from the Seoul National University, which is a standalone Java application and the StreamIT development tool[3] from MIT, which is also an Eclipse plug-in.

1.5 Outline

Chapter 2 gives an overview of the used third party technologies. Chapter 3 describes the structure of the plug-in. Chapter 4 and 5 explain the implementation of the new plug-in features.

[2] http://peace.snu.ac.kr/research/hopes/

[3] http://groups.csail.mit.edu/cag/streamit/shtml/eclipse-plugin.shtml

2 Background Technologies

In this chapter, the tools and frameworks used to develop the plug-in are described.

2.1 Eclipse

As base of development the Eclipse SDK is used. Even though Eclipse is primarily a Java IDE, its functionality is extendible via plug-ins, which can be managed by the integrated software manager. The DALipse plug-in has been developed with and for Eclipse 3.7 Indigo.

2.2 Eclipse Modeling Framework Project

The Eclipse Modeling Framework Project (EMF)[1] provides code generation facilities for data models specified as XML files. It also provides import functionalities for XSD files, which has been used in this thesis. The generated code includes XML serialisation for the model data.

[1] http://www.eclipse.org/emf/

2.3 Graphical Editing Framework

The Graphical Editing Framework (GEF)[2] provides a programming framework for graphical editors. Its internal structure is based on the model-view-controller design pattern. In particular, an EMF model is used in this thesis. The view is implemented with Draw2d, the drawing toolkit included with GEF. From within the editor the underlying model is changed with commands, which form the controller. Each executed command is put on a command stack. This stack can be manipulated with the Eclipse undo and redo UI functions. It is also used to determine if a project needs to be saved.

2.3.1 Layout File Format

A custom layout file format is used to store the layout of any of the three GEF editors. The file has to have the same base name as the corresponding model XML file. If no layout file is found for a given model XML file, the elements are placed randomly and the layout file is created the next time the model is saved. The file format is very simple, each file consists of one line per element following the form

```
element_name:x,␣y,␣width,␣height:
```

with x and y being the coordinates of the top left corner of the element. The numbers have to be separated by a comma followed by a space. Examples can be found in Sections 4.3 and 5.3.

[2] http://www.eclipse.org/gef/

3
Overview DALipse

In this chapter, an overview of the plug-in is given. For a description of the source code structure, see Appendix B.

3.1 Editors

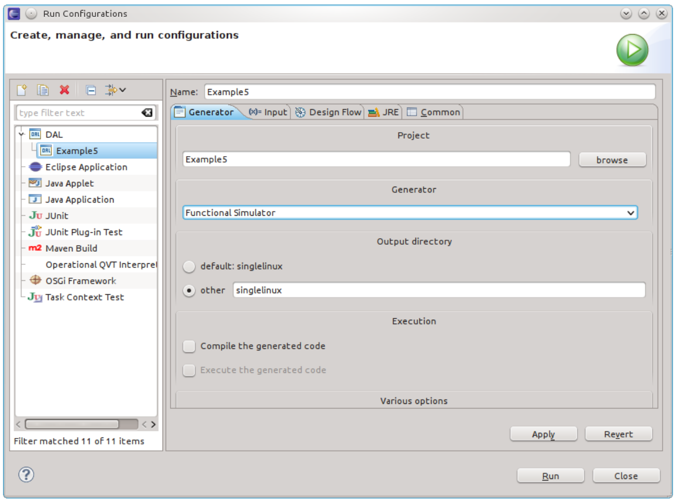
The plug-in consists of three different editors, the process network editor, the architecture editor and the finite state machine editor. The first editor has been developed as part of a prior thesis [5], and the later two editors are described in more detail in Chapters 4 and 5.

3.2 Launcher

The launcher serves as the main interaction with the DAL tool-chain. The most prominent part of the launcher is the green run button in the Eclipse tool bar. Fig. 3.1 shows the other part, the launch configuration dialog, where the run behaviour can be configured.

3.3 Project Wizard

There are two wizards for the plug-in, the “New Project” wizard and the example wizard. The “New Project” wizard (Fig. 3.2) consists of a name



Run Configurations

Create, manage, and run configurations

Name: Example5

Project: Example5

Generator: Functional Simulator

Output directory

○ default: singlelinux

● other singlelinux

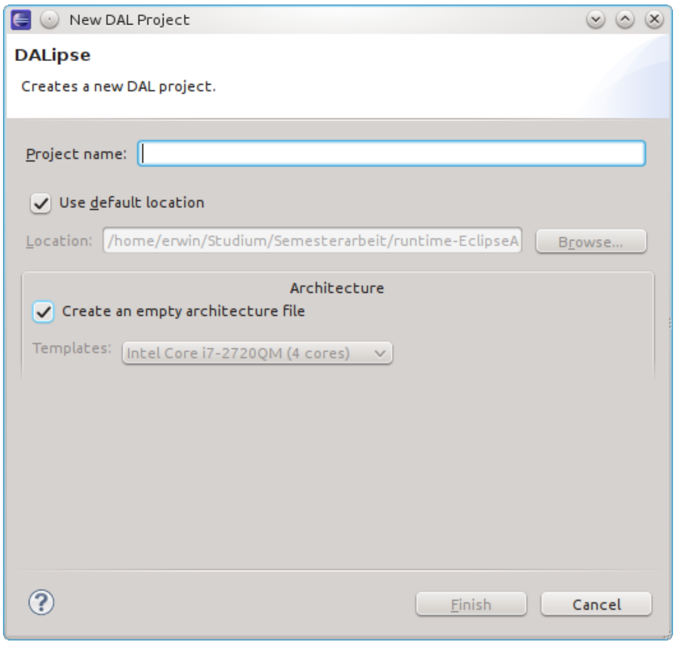
Execution

□ Compile the generated code

□ Execute the generated code

Filter matched 11 of 11 items

Figure 3.1: Launch configuration dialog showing the "Generator" tab for DAL launch configuration.



New DAL Project

DALipse

Creates a new DAL project.

Project name:

☑ Use default location

Location: /home/erwin/Studium/Semesterarbeit/runtime-EclipseA

Architecture

☑ Create an empty architecture file

Templates: Intel Core i7-2720QM (4 cores)

Figure 3.2: 'New project' wizard for a new DAL project.

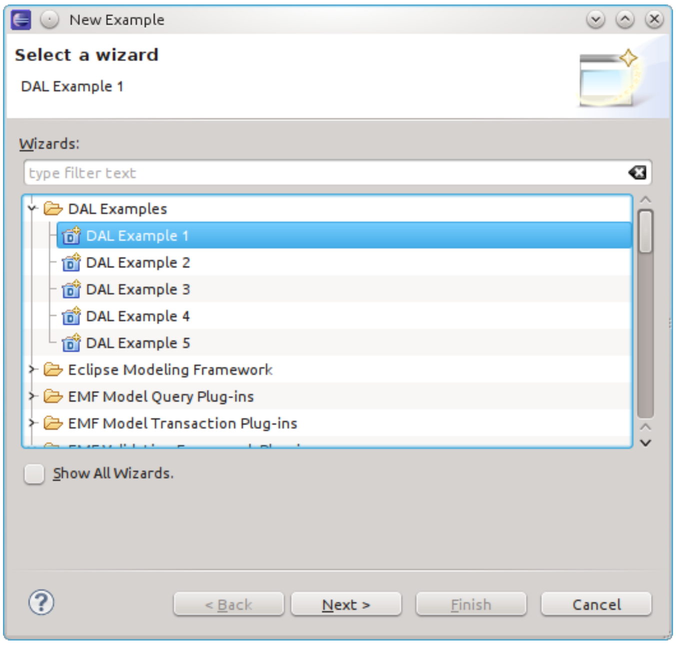


Figure 3.3: Eclipse example wizard showing the DAL examples.

field, an optional location field and an optional architecture field, which can be used to select one of the template architectures. As of the time of writing this thesis, there are two template architectures included in DALipse, namely an architecture template for Intel's SCC processor [1] and and Intel i7 processor with four cores.

3.3.1 Examples

The plug-in comes with five examples, a brief description of each can be found in the example wizard and the following list. All examples are delivered with an architecture file for an Intel i7 processor with four cores.

Example 1 Two producer-consumer applications are consecutively executed.

Example 2 An FFT application and a Matrix Multiplication application are consecutively executed.

Example 3 A MULTIFIR application and a FIR application are running concurrently as soon as a producer-consumer application has been completed.

Example 4 The producer-consumer application is executed. After completing 20 iterations, each process sends a corresponding event.

Example 5 A video player that can concurrently plays two videos. It consists of a user interface application (UI) and two MJPEG decoder applications.

4
Architecture Editor

In this chapter the first major contribution of this thesis, the architecture editor, is described. The architecture editor can be used to visually specify hardware platforms by connecting an number of different hardware elements. In the remainder of this chapter, first the model used in the architecture editor is explained. The second section is about the actual editor and its implementation with the GEF framework. In the last section, a case study is shown to illustrate the capabilities of the editor.

4.1 Model

The architecture model distinguishes three different levels of hierarchy. These are namely processors, shareds (local buses) and networks-on-chips (NoCs). All these elements have certain properties in common including the element name and the substitute property. Substitute elements are spare elements, i.e. no processes are mapped onto these processors or onto all processors connected to a shared or NoC.

4.1.1 Model Constraints

The model has some constraints, which do not directly follow from the XML specification [6]. The following statements have to be fulfilled at any time:

- Processors may only be connected to at most one shared.

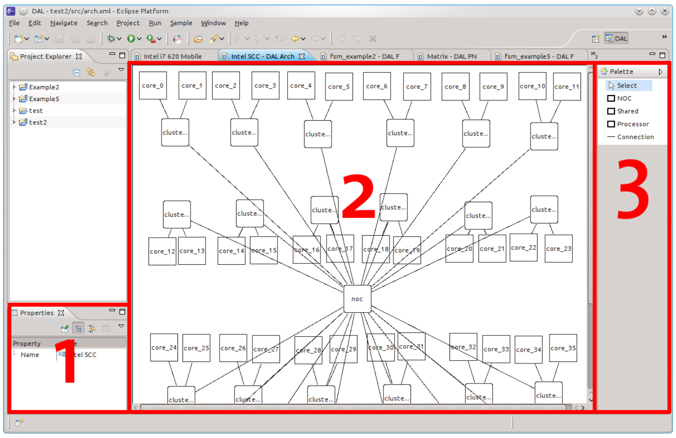


Figure 4.1: Architecture editor showing the Intel SCC processor [1].

- Shareds may only be connected to at most one NoC.
- Processors cannot be directly connected to NoCs.

All other connection possibilities are valid, in particular, a NoC may have connections to an arbitrary number of other NoCs.

4.1.2 Additions to the DAL Model

The model specified by DAL [6] is extended with plug-in specific elements. These additions have been kept minimal to be as close to the original model as possible. These additions are elements to store the layout of the architecture within the editor, namely the coordinates of the top left corner and the size of each hardware element. They are not saved in the XML file, but they are saved in an extra .layout file. For the specification of these files see Section 2.3.1.

4.2 Editor

The editor (Fig. 4.1) consists of three parts, namely the property view (1), the palette window (3) and the actual editor window (2). The property view can be used to edit the values of the attributes of the currently selected element. If no element is selected, it just shows the attributes of the whole architecture, which is the name attribute of the platform. The palette window shows all available tools. These are the select tool, to select and edit editor elements, and the create tools for all architecture elements (NoCs, shareds, processors,

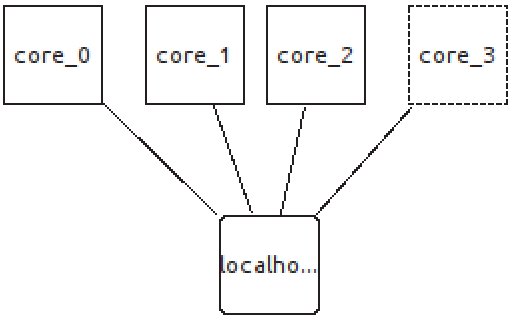


Figure 4.2: Architecture editor representation of the Intel i7 620 Mobile platform.

and connections). The connection create tool implements the constraints mentioned in Section 4.1.1.

4.2.1 Editor Window

The editor window is the part of the UI where the architecture elements are laid out and drawn, where new elements are added, and where existing elements can be selected. NoCs and shareds are drawn as rectangles with rounded corners. Processors are drawn with sharp corners. If the substitute property of an element is set to 1, the outline of any element becomes dashed. The label of a NoC, shared or processor shows its name property. Connections do not have any label, their name is automatically generated and only used internally.

4.3 Example

This section shows the Intel i7 620 Mobile platform within the DALipse plug-in in Fig. 4.2. The corresponding XML file is included along with the layout file.

Architecture XML File Example

```
<?xml version="1.0" encoding="UTF-8"?>
<architecture xmlns:xsi="http://www.w3.org/2001/XMLSchema-
    instance" xmlns="http://www.tik.ee.ethz.ch/~euretile/
    schema/ARCHITECTURE" xsi:schemaLocation="http://www.tik.
    ee.ethz.ch/~euretile/schema/ARCHITECTURE␣http://www.tik.
    ee.ethz.ch/~euretile/schema/architecture.xsd" name="
    Intel␣i7␣620␣Mobile␣platform␣(as␣one␣cluster)">
```

```
  <shared name="localhost">
    <port name="port1"/>
    <port name="port2"/>
    <port name="port3"/>
    <port name="port4"/>
  </shared>
  <processor id="0" name="core_0" type="RISC">
    <port name="port1"/>
  </processor>
  <processor id="1" name="core_1" type="RISC">
    <port name="port1"/>
  </processor>
  <processor id="2" name="core_2" type="RISC">
    <port name="port1"/>
  </processor>
  <processor id="3" name="core_3" substitute="1" type="RISC">
    <port name="port1"/>
  </processor>
  <link name="link_1">
    <end_point_1 name="core_0">
      <port name="port1"/>
    </end_point_1>
    <end_point_2 name="localhost">
      <port name="port1"/>
    </end_point_2>
  </link>
  <link name="link_2">
    <end_point_1 name="core_1">
      <port name="port1"/>
    </end_point_1>
    <end_point_2 name="localhost">
      <port name="port2"/>
    </end_point_2>
  </link>
  <link name="link_3">
    <end_point_1 name="core_2">
      <port name="port1"/>
    </end_point_1>
    <end_point_2 name="localhost">
      <port name="port3"/>
    </end_point_2>
  </link>
  <link name="link_4">
    <end_point_1 name="core_3">
      <port name="port1"/>
    </end_point_1>
    <end_point_2 name="localhost">
      <port name="port4"/>
    </end_point_2>
  </link>
</architecture>
```

Architecture Layout File Example

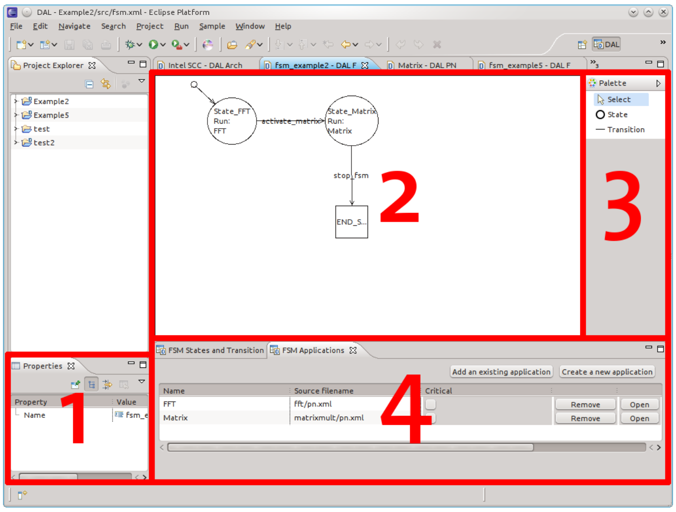
```
localhost:120, 117, 50, 50:
core_0:12, 12, 50, 50:
core_1:83, 12, 50, 50:
core_2:143, 12, 50, 50:
core_3:214, 12, 50, 50:
```

5
Finite State Machine Editor

This chapter is about the changes made to the existing DAL Finite State Machine (FSM) editor. The first section is about the model and the changes made to the DAL specification [6], which made the changes to the editor necessary, namely the type of machine used was changed from Mealy machines to Moore machines (see Section 4.3 in [2]). The second section is about the additional and changed elements within the editor itself. It starts with the graphical part of the editor and then describes the newly added views. In the last section, a case study is shown to illustrate the capabilities of the editor.

5.1 Model

The model, described in [2] to specify an FSM is a Moore machine. This is in contrast to the earlier version described in [6] which used a Mealy machine. Each FSM consists of three different types of elements, namely applications, scenarios and transitions. Applications have a unique name and a reference to their source process network file. This file path is either absolute or relative to the directory of the FSM file. There exists already an editor[5] to edit the process networks. The other two element types were changed with the transition to Moore machines. Each state represents a scenario and consists of a unique name and two lists of applications, one for running and one for paused applications. Applications in neither of these lists are inactive in that scenario. Transitions have a list of events which



Property	Value
Name	fsm_e

Name	Source filename	Critical
FFT	fft/pn.xml	□
Matrix	matrixmult/pn.xml	□

Figure 5.1: FSM editor showing Example 2 from the DALipse example set.

activate that transition.

5.1.1 Additions to the DAL Model

Similar to the architecture model, the FSM model specified by DAL [6] is extended with plug-in specific elements. These additions have been kept minimal to be as close as possible to the original model. These additions are, among others, elements to store the layout of the architecture within the editor, namely the coordinates of the top left corner and the size of each hardware element. They are not saved in the XML file, but they are saved in an extra .layout file. For the specification of these files see Section 2.3.1.

Another addition is a reference to the parent FSM element which has been added to states and transitions. States also include a list of their incoming and outgoing transitions. These have been added to simplyfy the editor implementation but they are not saved within the XML file or the .layout file. They are added to the model when the model is loaded and changed on request.

5.2 Editor

The editor (Fig. 5.1) consists of four parts, namely the property view (1), the actual editor window (2), the palette window (3) and the view area (4). The property view can be used to edit the values of the attributes of the currently selected element. If no element is selected, it shows the

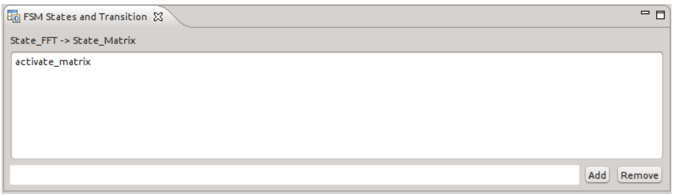


Figure 5.2: The transition view of the FSM view.

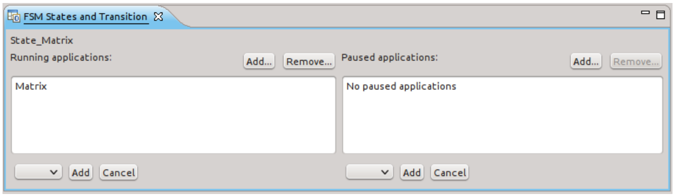


Figure 5.3: The state view of the FSM view.

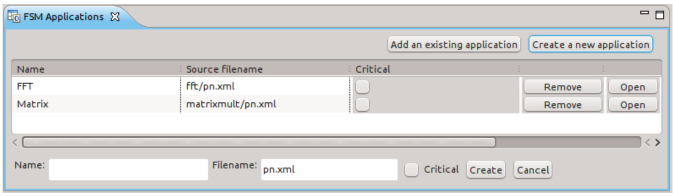
attributes of the whole FSM, which is just the name attribute of the FSM. The actual editor has already been implemented in a previous thesis [5], but slight adjustments had to be made, due to the model changes. The palette window shows all available tools. These are the select tool, to select and edit editor elements and the create tools for states and transitions. The view area holds the two newly added views, the FSM view and the application view.

5.2.1 Views

There were two views added to the FSM editor. The first one is a combined view for state application lists, which show the running and paused applications in that state (Fig. 5.3), and the event lists for transitions (Fig. 5.2). The second one is an application management view.

FSM View

Depending on the current selection in the editor window, the FSM view shows either nothing, the state application lists or the transition event list. If a state is selected, a list with all applications running in this state is displayed on the left half of the view area. On the right half, a list with all applications paused in this state is displayed. Each list has an add and remove button. If an application is selected within one of these lists, clicking on the remove button removes that application from that list. By clicking on the add button, the add controls on the bottom of the view area are displayed. These consist of an actual add button, a cancel button and a drop down list with the all applications in the application database of the



Name	Source filename	Critical		
FFT	fft/pn.xml	☐	Remove	Open
Matrix	matrixmult/pn.xml	☐	Remove	Open

Name: Filename: pn.xml ☐ Critical

Figure 5.4: The FSM application view for the FSM in Example 2.

FSM which are not yet in one of the list of the selected state. If a transition is selected, the event list for that transition is displayed. On the bottom of the view area the add and remove controls are displayed. Each add and remove action is added to the command stack of the editor window and can be undone and redone as long as the command stack supports it.

Application View

The application view (Fig. 5.4) shows the application database for the current FSM. The properties can be edited by clicking on the corresponding item in the list. Each application has two buttons. One to remove it from the list and another to open and edit the process network of the application. Note that the remove button only removes the application from the list, but does not delete the source files from the disk. On the top row, there are two buttons to add an existing application or create a new application. Clicking on either button opens the controls in the bottom part of the view area, but clicking on the add button, opens a file dialog to choose an existing PN source file first. The filename field is then filled with the chosen filename or, in case of creating a new application, the default filename. If a new application is created, first, a new directory within the directory of the FSM file is created and named with the new name of the application. Within that directory, a new file with the given filename and a minimal PN XML structure is created. Each add, create and, remove action is added to the command stack of the graph editor and can be undone and redone as long as the command stack supports it. However, note that undoing the creation of an application has no effect.

5.3 Example

This section shows Example 2 from the DALipse example set within the DALipse plug-in in Fig. 5.5. The corresponding XML file is included along with the layout file. The example consists of three states, namely two states with active applications and the end state. In the first state an FFT appli-

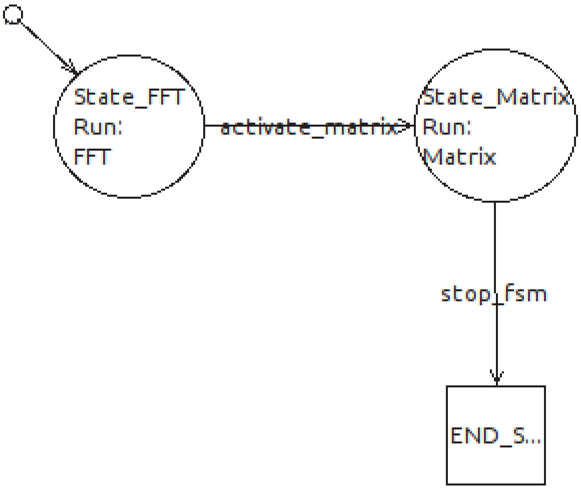
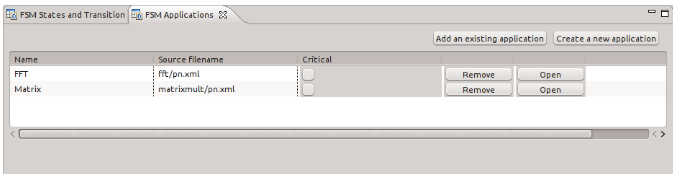


Figure 5.5: FSM editor representation of the example FSM.



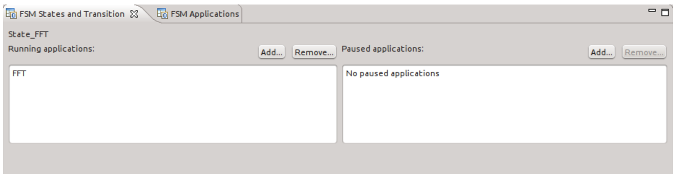
Name	Source filename	Critical
FFT	fft/pn.xml	□
Matrix	matrixmult/pn.xml	□

Figure 5.6: Application view for the example showing the two applications.

cation is executed. Upon receiving the `activate_matrix` event the runtime system switches to the second state in which a matrix multiplication application is executed. The `stop_fsm` event then switches to the end state.

FSM XML File Example

```
<?xml version="1.0" encoding="utf-8"?>
<fsm xmlns="http://www.tik.ee.ethz.ch/~euretile/schema/FSM"
        xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
```



State_FFT

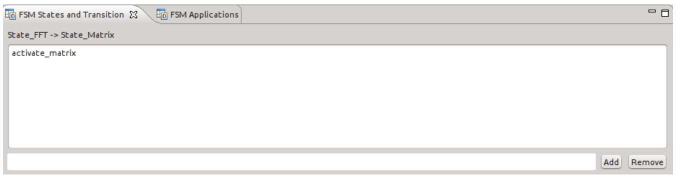
Running applications:

FFT

Paused applications:

No paused applications

Figure 5.7: FSM view for the first state showing the running FFT application.



State_FFT -> State_Matrix

activate_matrix

Figure 5.8: FSM view for the transition between the first and second state.

```
         name="fsm_example2"
         xsi:schemaLocation="http://www.tik.ee.ethz.ch/~
            euretile/schema/FSM_http://www.tik.ee.ethz.ch/~
            euretile/schema/fsm.xsd">
   <application name="FFT" src="fft/pn.xml" critical="0"/>
   <application name="Matrix" src="matrixmult/pn.xml"
        critical="0" />
   <transition name="trans_1" nextstate="State_FFT" />
   <state name="State_FFT">
      <transition name="trans_2" nextstate="State_Matrix">
        <event name="activate_matrix"/>
          </transition>
          <run application="FFT" />
   </state>
   <state name="State_Matrix">
      <transition name="trans_3" nextstate="END_STATE">
        <event name="stop_fsm"/>
          </transition>
          <run application="Matrix" />
   </state>
   <state name="END_STATE"/>
   <endstate name="END_STATE"></endstate>
</fsm>
```

FSM Layout File Example

```
State_FFT:80, 34, 76, 72:
State_Matrix:260, 31, 82, 77:
END_STATE:276, 199, 50, 50:
```

6
Conclusion and Outlook

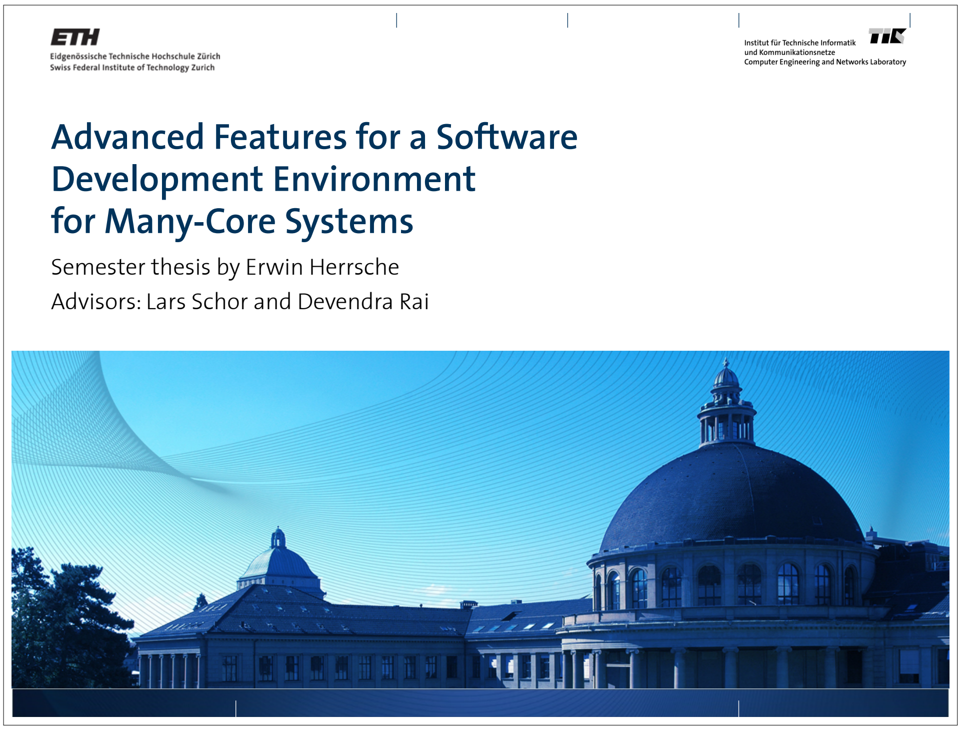
6.1 Conclusion

In this thesis, the DAL Eclipse plug-in (DALipse) has been improved with a graphical editor for DAL architecture specifications and editors for the application and transition event management. The newly added features followed the modular approach of the foundation, thus conserving the extensibility of the plug-in.

6.2 Outlook

Some features of the DAL XML specification [6] are not yet supported by the graphical editors, e.g. the iterator element. There are also no model verifications beyond the XML specification, e.g. to search for non-reachable states in a FSM. Finally, the interaction with the DAL tool-chain is still very basic, e.g. there is no mapping viewer.

A
Presentation Slides



ETH
Eidgenössische Technische Hochschule Zürich
Swiss Federal Institute of Technology Zurich

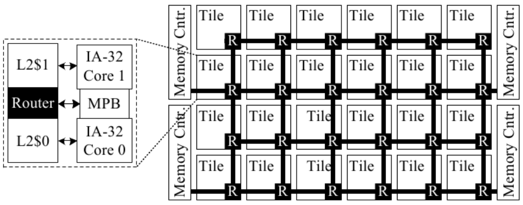
Institut für Technische Informatik
und Kommunikationsnetze
Computer Engineering and Networks Laboratory

Advanced Features for a Software Development Environment for Many-Core Systems

Semester thesis by Erwin Herrsche

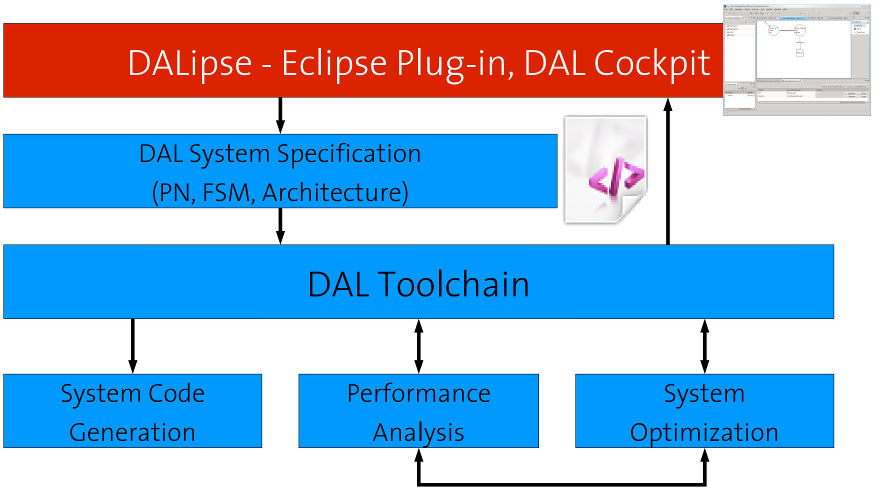
Advisors: Lars Schor and Devendra Rai

Motivation



- Intel SCC: 48 cores + 24 local buses + 1 NoC + 72 links
  = 145 elements to model
- Future: many more cores, hundreds of elements to model

Background



DALipse - Eclipse Plug-in, DAL Cockpit

DAL System Specification
(PN, FSM, Architecture)

DAL Toolchain

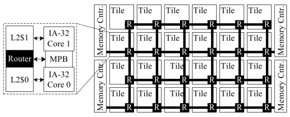
System Code Generation

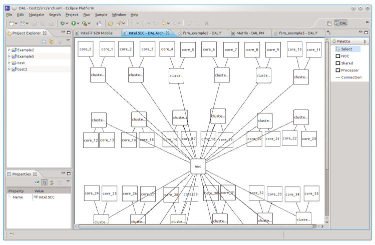
Performance Analysis

System Optimization

Goals

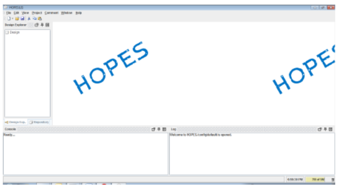
- Extend the existing Eclipse plug-in (DALipse) with
  - a graphical editor to specify DAL architectures
  - a graphical editor to specify application interactions as a Moore machine
  - a management system for applications and transition events





Related Work

- HOPES [1]
  - Seoul National University
  - Standalone Java application

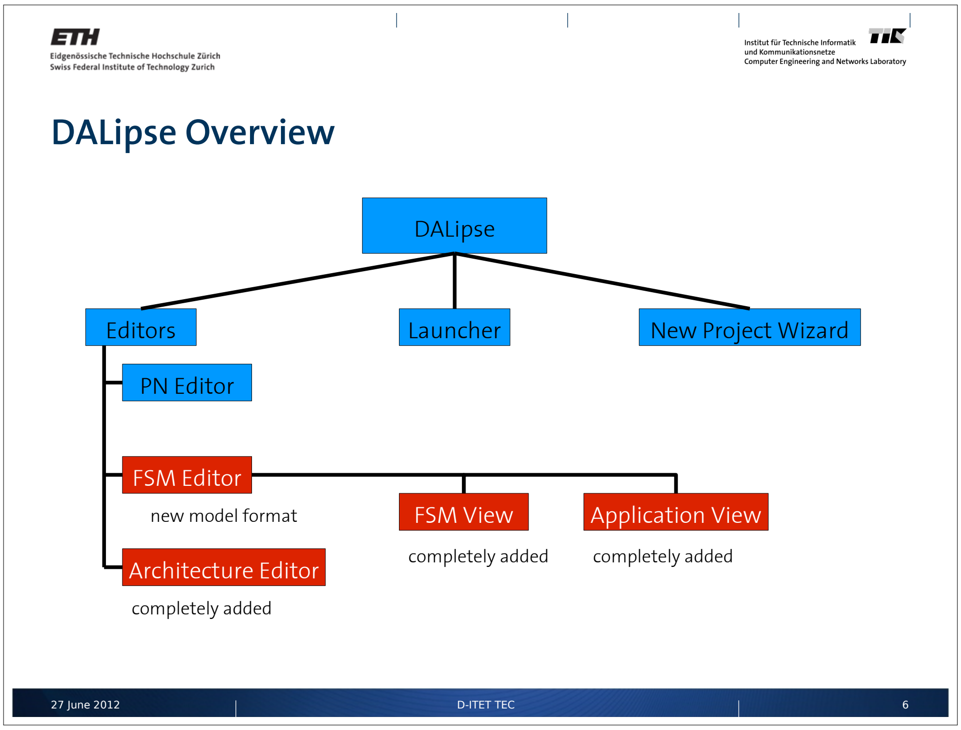


- StreamIT development tool [2]
  - MIT
  - Eclipse plug-in

StreamIt

[1]http://peace.snu.ac.kr/research/hopes/

[2]http://groups.csail.mit.edu/cag/streamit/shtml/eclipse-plugin.shtml



ETH
Eidgenössische Technische Hochschule Zürich
Swiss Federal Institute of Technology Zurich

Institut für Technische Informatik
und Kommunikationsnetze
Computer Engineering and Networks Laboratory

DALipse Overview

DALipse

Editors

Launcher

New Project Wizard

PN Editor

FSM Editor

new model format

FSM View

completely added

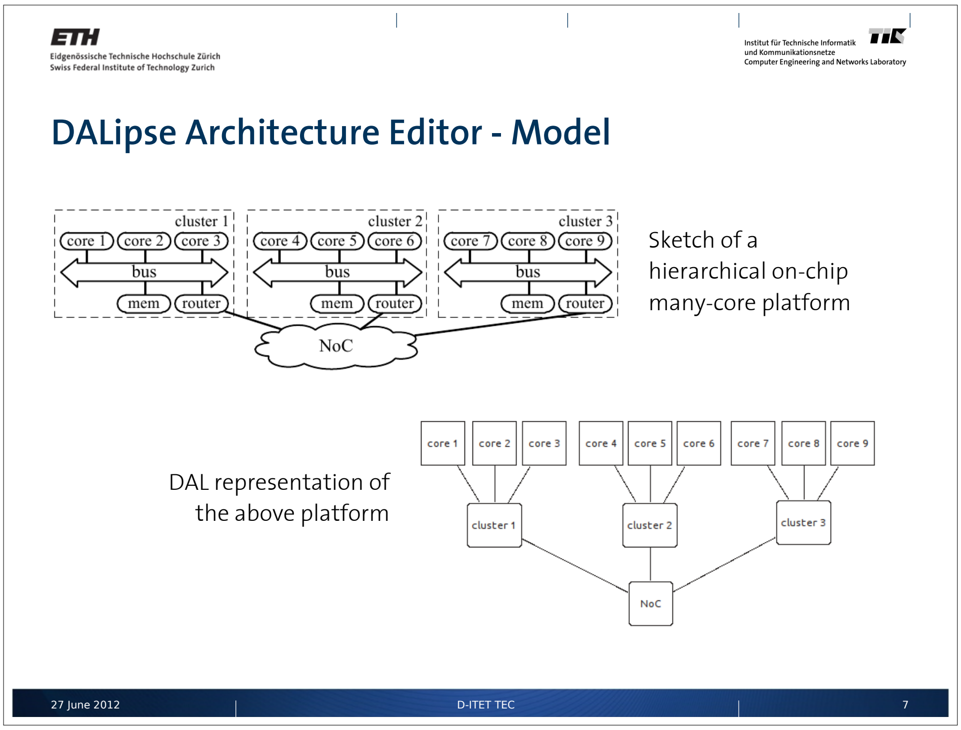
Application View

completely added

Architecture Editor

completely added

27 June 2012 D-ITET TEC 6



ETH
Eidgenössische Technische Hochschule Zürich
Swiss Federal Institute of Technology Zurich

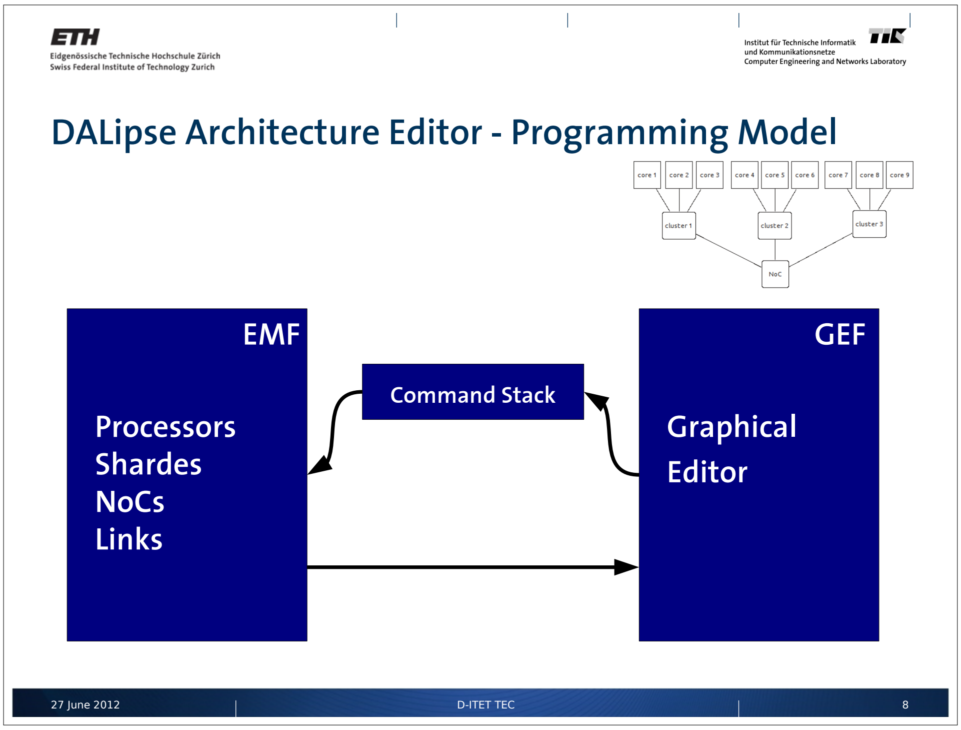
Institut für Technische Informatik
und Kommunikationsnetze
Computer Engineering and Networks Laboratory

DALipse Architecture Editor - Model

Sketch of a hierarchical on-chip many-core platform

DAL representation of the above platform

27 June 2012 D-ITET TEC 7



DALipse Architecture Editor - Programming Model

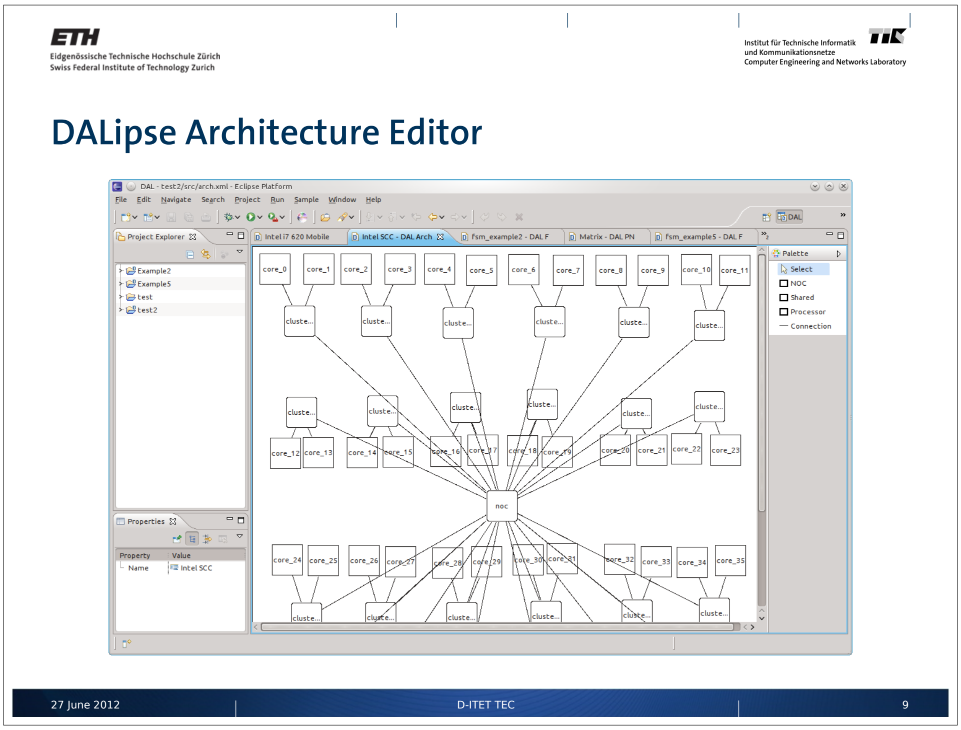
EMF

Processors
Shardes
NoCs
Links

Command Stack

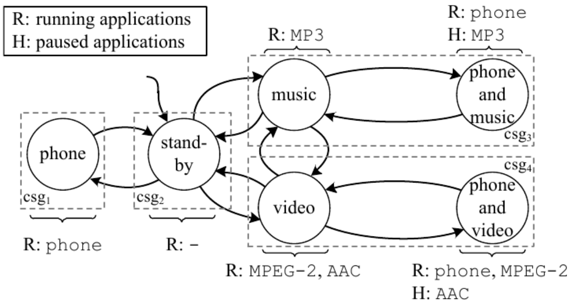
GEF

Graphical
Editor



DALipse Architecture Editor

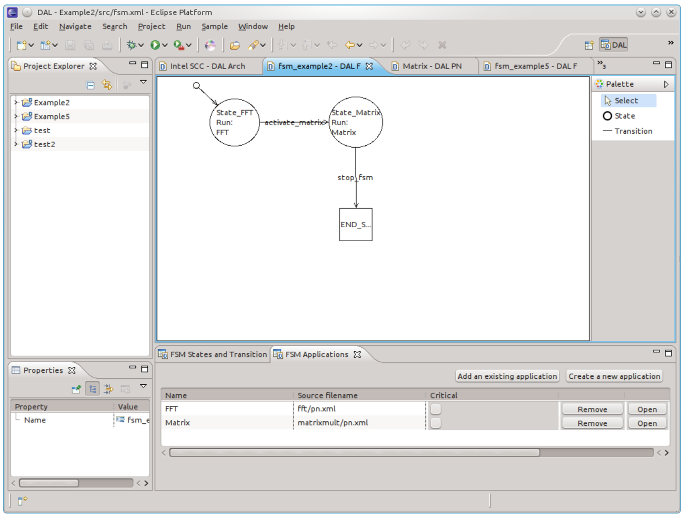
DALipse FSM Editor - Model



simplified cell phone operating system

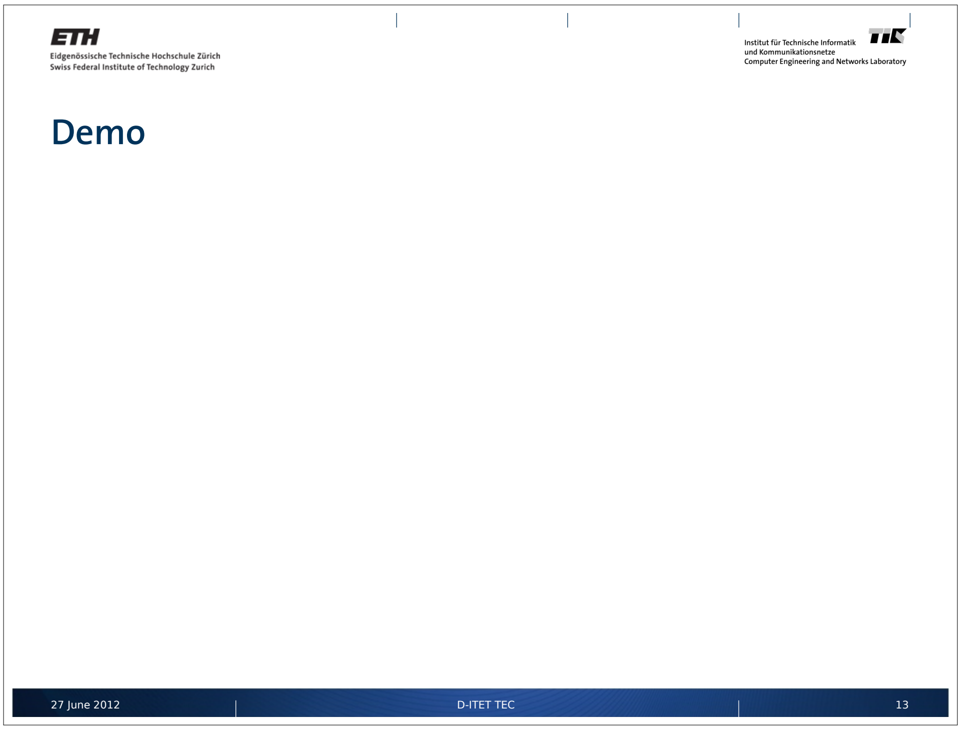
- FSM captures the dynamic behaviour of the system
- States (or scenarios) represent a set of running and paused applications

DALipse FSM Editor



Property	Value
Name	fsm_e

Name	Source filename	Critical		
FFT	fft/pn.xml	□	Remove	Open
Matrix	matrixmult/pn.xml	□	Remove	Open

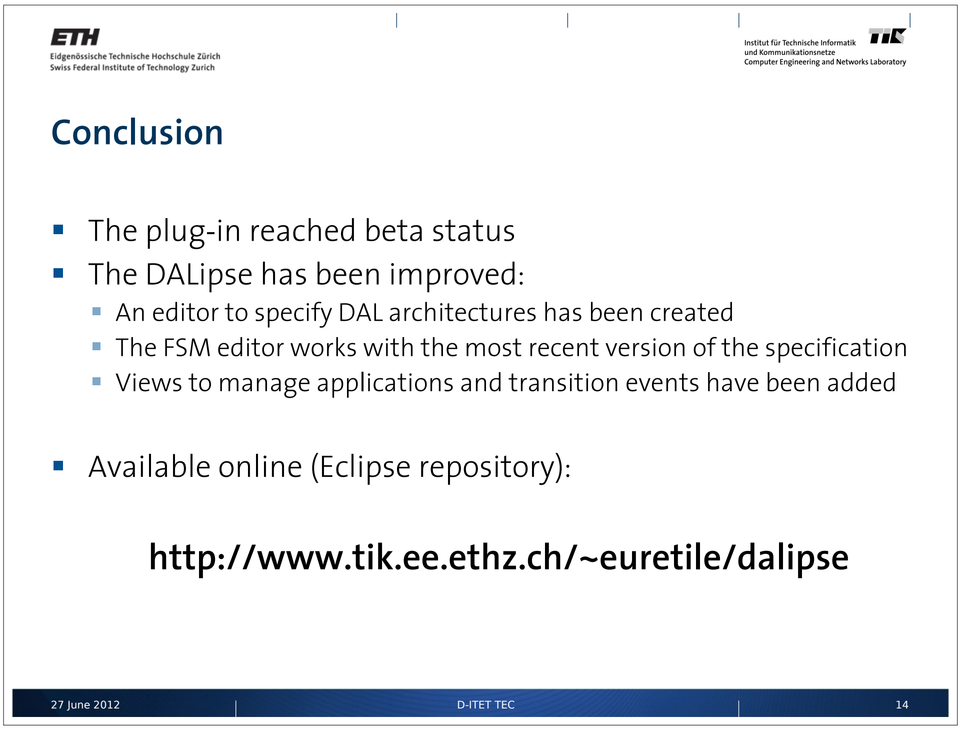


ETH
Eidgenössische Technische Hochschule Zürich
Swiss Federal Institute of Technology Zurich

Institut für Technische Informatik
und Kommunikationsnetze
Computer Engineering and Networks Laboratory

Demo

27 June 2012 | D-ITET TEC | 13



ETH
Eidgenössische Technische Hochschule Zürich
Swiss Federal Institute of Technology Zurich

Institut für Technische Informatik
und Kommunikationsnetze
Computer Engineering and Networks Laboratory

Conclusion

- The plug-in reached beta status
- The DALipse has been improved:
  - An editor to specify DAL architectures has been created
  - The FSM editor works with the most recent version of the specification
  - Views to manage applications and transition events have been added
- Available online (Eclipse repository):

http://www.tik.ee.ethz.ch/~euretile/dalipse

27 June 2012 | D-ITET TEC | 14

B
DALipse Source Code Overview

In the following <prefix> means the overall package and project prefix ch.ethz.tec.dal.

Eclispe Projects Overview

<prefix>.architecture.model	The model project for the architecture model.
<prefix>.fsm.model	The model project for the finite state machine model.
<prefix>.pn.model	The model project for the process network model.
<prefix>.editor	The main GUI plugin-in project.

Package overview for the main GUI plug-in project (ch.ethz.tec.dal.editor)

ch.ethz.tec.dal		This is the plug-in starting point package. It contains a single class which serves as entry point for the plug-in code.
`<prefix>`.editor		This is the editor starting point package. It contains the base class for the graphical editors and a class which decides which editor to open. Some things useful throughout the plug-in are also stored in this package.
`<prefix>`.editor.tool		Classes which are useful throughout the plug-in are stored in this package.
`<prefix>`.editor.util		Classes which are useful throughout the plug-in are stored in this package.
`<prefix>`.editor.architecture `<prefix>`.editor.fsm `<prefix>`.editor.pn		These packages contain the base editor class and the base palette class.
	.command	This package contains all command classes for a specific editor.
	.factory	This package contains the EMF element factories.
	.figure	This package contains all Draw2d figures. The drawing code for visible elements is located in this package.
	.part	This package contains all GEF edit parts as well as the edit part factory.
	.policy	This package contains all GEF policies used in the GEF edit parts.
	.tool	This package contains classes useful within a specific editor.
`<prefix>`.editor.fsm.view		This is the main package for the FSM views.
	.adapter	All view event adapters are located here.
	.command	All view event commands called from the event adapters are located here.
`<prefix>`.editor.wizard		
	.examples	This package contains the example wizard.
	.newproject	This package contains the new project wizard.

Bibliography

[1] J. Howard, S. Dighe, Y. Hoskote, S. Vangal, D. Finan, G. Ruhl, D. Jenkins, H. Wilson, N. Borkar, G. Schrom, F. Pailet, S. Jain, T. Jacob, S. Yada, S. Marella, P. Salihundam, V. Erraguntla, M. Konow, M. Riepen, G. Droege, J. Lindemann, M. Gries, T. Apel, K. Henriss, T. Lund-Larsen, S. Steibl, S. Borkar, V. De, R. Van Der Wijngaart, and T. Mattson, "A 48-Core IA-32 message-passing processor with DVFS in 45nm CMOS," in *Porc. Int'l Solid-State Circtuits Conference*, pp. 108–109, Feb 2010.

[2] L. Schor, I. Bacivarov, D. Rai, H. Yang, S. haeng Kang, and L. Thiele, "Scenario-based design flow for mapping streaming applications onto on-chip many-core systems," in *Proc. International Conference on Compilers Architecture and Synthesis for Embedded Systems (CASES)*, (Tampere, Finland), ACM, 2012.

[3] L. Thiele, I. Bacivarov, W. Haid, and K. Huang, "Mapping Applications to Tiled Multiprocessor Embedded Systems," in *Proc. 7th Int'l Conference on Application of Concurrency to System Design (ACSD'07)*, (Bratislava, Slovak Republic), pp. 29–40, July 2007.

[4] G. Kahn, "The Semantics of a Simple Language for Parallel Programming," in *Proceedings of the IFIP Congress*, pp. 471–475, 1974.

[5] E. Geiser, "Software Development Environment for Many-Core Systems," semester thesis, ETH Zurich, Department of Information Technology and Electrical Engineering, 2011.

[6] I. Bacivarov, D. Rai, L. Schor, and H. Yang, "Semantics of the DAL XML Schemata." http://www.tik.ee.ethz.ch/ euretile/docs/XMLSpecification.pdf, 2011. Revision 597.