Integration of OpenModelica into the Multi-paradigm Modeling Environment of Ptolemy II

by

Mana Mirzaei

LIU-IDA/LITH-EX-A--13/065--SE

2014-01-09
Examensarbete

Integration av OpenModelica med Multi-Paradigm Modelleringsramverket Ptolemy II

av

Mana Mirzaei

LIU-IDA/ LITH-EX-A--13/065--SE

2014-01-09

Handledare: Dr. Lena Buffoni
Examinator: Prof. Peter Fritzson
Contents

Chapter 1 Introduction ........................................................................................................... 7
1.1 Background .................................................................................................................... 7
1.2 Purpose ......................................................................................................................... 7
1.3 Thesis Overview ............................................................................................................ 8

Chapter 2 Background ......................................................................................................... 9
2.1 Modelica ....................................................................................................................... 9
2.2 OpenModelica .............................................................................................................. 9
2.3 Ptolemy II ................................................................................................................... 10

Chapter 3 OpenModelica in Ptolemy II ............................................................................. 11
3.1 Motivation ................................................................................................................... 11
3.2 Drawbacks and Problems Today ................................................................................. 11
3.3 Approach .................................................................................................................... 12

Chapter 4 OpenModelica Extension ............................................................................... 13
4.1 The Software Architecture of OpenModelica to Create New Extension of Ptolemy II .... 14
4.1.1 Interactive Simulation of the Modelica Models ....................................................... 15
4.1.2 Non-Interactive Simulation of the Modelica Models ............................................. 17
4.2 The Software Architecture of Ptolemy II to Create New Extension of OpenModelica ... 18
4.2.1 OpenModelica Director ....................................................................................... 18
4.2.2 OpenModelica Actor ......................................................................................... 23
4.3 OpenModelica Execution in Ptolemy II ....................................................................... 25

Chapter 5 Implementation ................................................................................................ 27
5.1 Graphical User Interface ............................................................................................ 27
5.2 Code structure ............................................................................................................. 29
5.3 Implementation Challenges ........................................................................................ 38
5.3.1 Incomplete OpenModelica API ........................................................................... 38
5.3.2 Time Challenge .................................................................................................... 38

Chapter 6 Testing ............................................................................................................... 40
6.1 System Testing in Ptolemy II ..................................................................................... 40
6.2 Test Cases .................................................................................................................. 41
6.3 Performance ............................................................................................................... 42

Chapter 7 Conclusion ....................................................................................................... 44
7.1 Future Works .............................................................................................................. 44
7.1.1 Improve Ptolemy II by Adding OpenModelica Import Menu ......................... 44
7.1.2 Design New Library Known as BCVTB for OpenModelica Library ................. 44

Appendix A User Guide ..................................................................................................... 45
7.2 Getting Started .......................................................................................................... 45
7.2.1 Installation ............................................................................................................ 45
7.3 Model Examples ......................................................................................................... 45
7.3.1 Example 1 – Simple Model for Plotting PLT File ................................................. 46
7.3.2 Example 2 - Composite Model for Plotting CSV File by XYPlotter ................. 48
7.3.3 Example 3 – Composite Model for Displaying the Result of Simulation in Interactive Processing Mode .......................................................... 51
7.3.4 Example 4 – Composite Model for Modifying Parameter of the Super Model .... 54

Bibliography ...................................................................................................................... 58
Table of Acronyms ........................................................................................................... 60
### Table of Figures

<table>
<thead>
<tr>
<th>Figure</th>
<th>Description</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>1.</td>
<td>Client-Server interconnection structure of the compiler/interpreter main</td>
<td>14</td>
</tr>
<tr>
<td></td>
<td>program and interactive tool interfaces.</td>
<td></td>
</tr>
<tr>
<td>2.</td>
<td>Client-Server interconnection structure of the compiler/interpreter main</td>
<td>15</td>
</tr>
<tr>
<td></td>
<td>program and interactive tool interfaces after adding new Ptolemy II as a</td>
<td></td>
</tr>
<tr>
<td></td>
<td>new client.</td>
<td></td>
</tr>
<tr>
<td>3.</td>
<td>OMI communication architecture.</td>
<td>16</td>
</tr>
<tr>
<td>4.</td>
<td>Key packages and composing classes in Ptolemy II.</td>
<td>17</td>
</tr>
<tr>
<td>5.</td>
<td>OpenModelica director with its parameters.</td>
<td>18</td>
</tr>
<tr>
<td>6.</td>
<td>Domain package in Ptolemy II.</td>
<td>19</td>
</tr>
<tr>
<td>7.</td>
<td>Domain package after adding OpenModelica domain.</td>
<td>20</td>
</tr>
<tr>
<td>8.</td>
<td>Flow of the control of Ptolemy II’s composite actor.</td>
<td>21</td>
</tr>
<tr>
<td>9.</td>
<td>OpenModelica actor with its sample parameters.</td>
<td>22</td>
</tr>
<tr>
<td>10.</td>
<td>Interaction diagram between Ptolemy II and OpenModelica.</td>
<td>23</td>
</tr>
<tr>
<td>11.</td>
<td>OpenModelica.XML</td>
<td>24</td>
</tr>
<tr>
<td>12.</td>
<td>OpenModelica.DIRECTOR in EXPERIMENTAL.DIRECTORS.XML.</td>
<td>25</td>
</tr>
<tr>
<td>13.</td>
<td>OpenModelica sub-packages.</td>
<td>26</td>
</tr>
<tr>
<td>14.</td>
<td>Class diagram of the lib package.</td>
<td>27</td>
</tr>
<tr>
<td>15.</td>
<td>Part of the constructor of OpenModelica class.</td>
<td>28</td>
</tr>
<tr>
<td>16.</td>
<td>Code snippet of OMIThread class.</td>
<td>29</td>
</tr>
<tr>
<td>17.</td>
<td>The code snippet shows the implementation of Fire() method one of the key</td>
<td>30</td>
</tr>
<tr>
<td></td>
<td>functions of OpenModelica actor.</td>
<td>31</td>
</tr>
<tr>
<td>18.</td>
<td>Functions included in the fire method of OpenModelica actor.</td>
<td>32</td>
</tr>
<tr>
<td>19.</td>
<td>Function which is invoked in the wrapup method.</td>
<td>33</td>
</tr>
<tr>
<td>20.</td>
<td>Number of lines in Java classes.</td>
<td>34</td>
</tr>
<tr>
<td>21.</td>
<td>Test case S01.</td>
<td>35</td>
</tr>
<tr>
<td>22.</td>
<td>Test case S02.</td>
<td>36</td>
</tr>
<tr>
<td>23.</td>
<td>Test case S03.</td>
<td>37</td>
</tr>
<tr>
<td>24.</td>
<td>Execution time of test models in Ptolemy II and OpenModelica.</td>
<td>38</td>
</tr>
<tr>
<td>25.</td>
<td>OpenModelica.PLTPlot.xml.</td>
<td>39</td>
</tr>
<tr>
<td>26.</td>
<td>Parameters of OpenModelica in OpenModelica.PLTPlot.XML.</td>
<td>40</td>
</tr>
<tr>
<td>27.</td>
<td>Parameters of DCMotor.</td>
<td>41</td>
</tr>
<tr>
<td>28.</td>
<td>Plotting OpenModelica simulation by PLTPlotter.</td>
<td>42</td>
</tr>
<tr>
<td>29.</td>
<td>OpenModelicaXYPlotter.XML and the contained model.</td>
<td>43</td>
</tr>
<tr>
<td>30.</td>
<td>Parameters of OpenModelica in OpenModelicaXYPlotter.XML.</td>
<td>44</td>
</tr>
<tr>
<td>31.</td>
<td>Parameters of Ramp actor in OpenModelicaInteractive.XML.</td>
<td>45</td>
</tr>
<tr>
<td>32.</td>
<td>Displaying part of simulation result of OpenModelicaInteractive.XML by</td>
<td>46</td>
</tr>
<tr>
<td></td>
<td>Display actor.</td>
<td></td>
</tr>
<tr>
<td>33.</td>
<td>OpenModelicaInherit.xml with the contained model.</td>
<td>47</td>
</tr>
<tr>
<td>34.</td>
<td>Set parameter of OpenModelica actor with part of the CSV generated file</td>
<td>48</td>
</tr>
<tr>
<td></td>
<td>in non-interactive processing.</td>
<td></td>
</tr>
<tr>
<td>35.</td>
<td>Set parameter of OpenModelica actor with part of the simulation result in</td>
<td>49</td>
</tr>
<tr>
<td></td>
<td>interactive processing.</td>
<td></td>
</tr>
</tbody>
</table>

*Note: The page numbers are placeholders and should be adjusted according to the actual document.*
Acknowledgments

I would like to express my gratitude towards the people who helped me through this thesis, without them this project would not have been made possible. First of all, I would like to thank Christopher Brooks, Software Manager of the Ptolemy Project and Adeel Asghar, OpenModelica developer for offering significant support throughout the thesis work. Furthermore, I would like to express my appreciation to my supervisor Dr. Lena Buffoni and examiner Professor Peter Fritzson for providing great inspiration during this thesis work.

Linköping, 2014
Mana Mirzaei
Chapter 1
Introduction

This chapter is the introduction of a thesis work that completes my studies in Master of Science at Linköping University. The thesis work has been performed at the Programming Environment Laboratory (PELAB) in IDA in collaboration with the Center for Hybrid and Embedded Software Systems (CHESS) in the Department of Electrical Engineering and Computer Sciences of the University of California at Berkeley.

1.1 Background

Concurrent systems are becoming increasingly common in spite of their complexity. Therefore, a multitude of computational models and tools for modeling and developing such systems have emerged. Ptolemy II is an environment for modeling, simulation and design of large concurrent real-time systems [2] which is mainly centered on addressing heterogeneous systems. Ptolemy II which is developed at CHESS in the Department of Electrical Engineering and Computer Sciences of the University of California, is a system-level design environment that provides the possibility of blending different versions of Models of Computation (MoCs) in one hierarchical heterogeneous model [3]. Modelica [6] is a non-proprietary, object-oriented, equation based language aimed at modeling complex multi-domain physical systems. This language is also supported by a number of free and commercial tools, in particular by OpenModelica [4], an open source compiler and tool suite.

1.2 Purpose

This thesis work concerns the integration of OpenModelica into the multi-paradigm environment of Ptolemy II. The integration of OpenModelica and Ptolemy II allows the simulation of non-causal models in Modelica by the OpenModelica Compiler and simulator within the Ptolemy II environment where models communicate with different versions of MoCs with the aim of addressing the heterogeneous design as well.
1.3 Thesis Overview

This section explains how the report is constructed and gives a brief description of the content of each chapter. The purpose of this section is providing a quick overview for the reader:

Chapter 2: its only purpose is giving a short background about Modelica, OpenModelica and Ptolemy II.

Chapter 3: provides the motivation for this thesis work, the current drawbacks and the overview of the proposed solution.

Chapter 4: provides the implementation process of OpenModelica extension.

Chapter 5: provides the code structure of the implementation, challenges during the implementation and brief comparison between the performance of running two test models in Ptolemy II and OpenModelica.

Chapter 6: investigates two testing approaches in Ptolemy II, examines the selected one for the integration and illustrates three test cases with their snapshots.

Chapter 7: Provides the conclusion and future suggestions on the development of collaborative project of OpenModelica and Ptolemy II.
Chapter 2
Background

2.1 Modelica

According to [Peter Fritzson. Principles of Object-Oriented Modeling and Simulation with Modelica 2.1, Wiley-IEEE Press, 2004.], “Modelica is a modeling language that allows specification of mathematical models of complex natural or man-made systems”. The arrival of the Modelica modeling language brings a revolution in modeling and simulation based on the ability to create model libraries with re-usable components and on supporting modeling and simulation of multi-domain applications [7].

The Design of the Modelica language provides the opportunity of generating efficient simulation code automatically with the aim of facilitating exchange of models, model libraries and simulation specifications by tools.

In terms of software component modeling, Modelica is based on a powerful model that provides the opportunity of creating and connecting components. The Modelica language realizes a connection mechanism among components, components are instances of the Modelica classes and connectors are interfaces of components [7]. Moreover, connections impose equation constraints on variables.

2.2 OpenModelica

There has been an enormous evolution in the specification of Modelica over the past 13 years. The Modelica user community has grown to become fairly large, a number of commercial tools are available implementing the Modelica language such as Dymola [15] and MathModelica [8]. Additionally, an open-source tool, OpenModelica, an open-source environment with the purpose of modeling and simulation of Modelica models [4] is under development since September 1997 [10].

The effort for developing OpenModelica is supported and coordinated by the Open Source Modelica Consortium (OSMC). The first major computational and simulation goal of OpenModelica effort is providing an interactive computational environment for the Modelica language in order to make the Modelica language widely available for the development and execution of algorithms for controlling system design, solving nonlinear equation systems, or developing optimization algorithms that are applied to complex applications [6].
The other goal is providing an environment for having a rather complete implementation of the Modelica language that entails the simulation of equation-based models and additional facilities in the programming environment [6].

2.3 Ptolemy II

The Ptolemy Project\(^1\) started working on developing Ptolemy II in 1996. Indeed the key goals of developing Ptolemy II were: utilizing the capabilities of Java, network integration, code migration and built-in threading as well as centering on the mechanism of combining actors with multi-paradigm modeling [1]. Ptolemy II also proposed the concept of domain polymorphism\(^2\) and modal models\(^3\) [13]. Some of the fundamental Ptolemy II features are as follows [14]:

- **Java**, Ptolemy II is implemented in Java, which allows Ptolemy II to be:
  - *Platform independent*, currently Ptolemy II runs under Solaris and Windows.
  - *Threaded*, Ptolemy II builds on the threading primitives in Java.
- **Modularity**, Ptolemy II is composed of software packages that can be utilized independently.
- **Mutable System**, Ptolemy II allows the design to be modified as it is running.
- **Software Architecture**, Ptolemy II has designed with object modeling and design patterns in mind.

---

\(^1\) [http://ptolemy.eecs.berkeley.edu/index.htm](http://ptolemy.eecs.berkeley.edu/index.htm)

\(^2\) A component can be created to run on various MoCs.

\(^3\) Hierarchical combination of finite state machines with other MoCs.
Chapter 3
OpenModelica in Ptolemy II

This chapter explains the reason that Modelica is of significant interest in Ptolemy II and investigates the existing approach for simulation of the Modelica models within the Ptolemy II environment. Afterwards, I provide a brief description of the integration mechanism to improve the previous approach.

3.1 Motivation

In Modelica, components are concurrent and communicate via ports, these ports are considered neither inputs nor outputs and the connections between ports impose equation constraints on variables that all lead to actor-like semantics of Modelica. Modelica is actually most suited to modeling physical systems, that can be intuitively described by equations.

Ptolemy II project mainly addresses the heterogeneous modeling, simulation and design of concurrent embedded systems. The major motivation of this project is generating models that can be implemented in a broad variety of computational models such as discrete-event, continuous-time, finite-state machine, synchronous data-flow and etc.

Therefore, the use of the Modelica models in the causal environment of Ptolemy II allows the simulation of the non-causal models of Modelica in combination with variant Models of computation that are realized in Ptolemy II, such as synchronous data-flow to support the design of heterogeneous systems.

3.2 Drawbacks and Problems Today

In the approach of simulating the Modelica model in Ptolemy II through Dymola \[15\], the Ptolemy-based co-simulation tool known as Building Controls Virtual Test Bed (BCVTB) is employed for conducting the co-simulation. This approach causes the constraint of adding the BCVTB block to the Modelica model in order to enable simulation in Ptolemy II.

\[\text{http://simulationresearch.lbl.gov/bcvtb}\]
The core purposes of BCVTB are co-simulation, real-time simulation and run-time coupling of different simulation programs for data exchange. Through co-simulation at least two simulators, each solving an initial-value differential or difference equation, are coupled to exchange dependent data on the state variables. Moreover, BCVTB couples any number of simulation programs, in place of coupling two simulators directly. The software architecture of BCVTB is a modular design which provides the graphical model building environment, synchronizes the exchanged data and visualizes the system evolution during run-time [16].

In that approach, Ptolemy II acts as the middleware for implementing BCVTB and actors in Ptolemy II are responsible for starting a server that uses the BSD\(^5\) socket utilized for exchanging data between the simulator and the actor as well as implementing the inter-process communication [16].

### 3.3 Approach

Prior to this thesis work, the simulation of the Modelica model in Ptolemy II were not available. An initial batch-oriented version of simulating Modelica models in Ptolemy II was added early. During the course of this work, the possibility to simulate Modelica models from Ptolemy II using the proprietary Dymola tool was added, requiring an additional library called BCVTB.

In our approach, OpenModelica, an open source tool for modeling and simulation of the Modelica models, has been added as a new Model of Computation to Ptolemy II’s domains. OpenModelica is quickly growing to be a successful alternative to commercial simulation and mathematical packages such as Mathworks Simulink. OpenModelica makes the model-building easier through employing model/sub-model relationships to improve model reusability. The creators of Ptolemy II follow an analogous set of rules to OpenModelica through using an OOP\(^6\)-like architecture in order to build and improve models [23].

Adding this new MoC domain enables the simulation of the Modelica models via OMC within the Ptolemy II environment to communicate with other Models of Computation to address the heterogeneity of the design. Moreover, our approach removes the need for including the additional library of BCVTB block in the Modelica models that are supposed to be simulated.

---

\(^5\) Berkeley Software Distribution

\(^6\) Object Oriented Programming
Chapter 4
OpenModelica Extension

In the first part of the integration process, it is essential to inspect the software architecture of OpenModelica and Ptolemy II with the goal of figuring out the starting point of creating new extension in Ptolemy II.

In this work, the course of the integration was divided into two primary steps:

1. Investigation of the software architecture of OpenModelica to figure out how the integration to Ptolemy II should be carried out.

2. Investigation of the software architecture of Ptolemy II to figure out how an OpenModelica director should be added as a new software extension.

Then, the execution sequence of OpenModelica in Ptolemy II is explained and the overall interaction between OpenModelica and Ptolemy II is depicted with an interaction diagram.
4.1 The Software Architecture of OpenModelica to Create New Extension of Ptolemy II

First the software architecture of OpenModelica was inspected to figure out how the integration to Ptolemy II should be carried out. The OpenModelica architecture is client-server structure, as illustrated in Figure 1.

![Diagram](image)

**Figure 1.** Client-Server interconnection structure of the compiler/interpreter main program and interactive tool interfaces.

Components of OpenModelica including the *compiler/interpreter main program* act as a server which serves interactive tool interfaces such as *Graphic Model Editor, Interactive Session Handler* and *Eclipse Plugin* via CORBA\(^7\) standard. The server component that is mainly involved in this integration is the OpenModelica Compiler (OMC). OMC is invoked via CORBA from the client side and Ptolemy II acts as a new client beside other interactive tool interfaces as illustrated in the new Client-Server architecture of OpenModelica in Figure 2.

---

\(^7\) Common Object Request Broker Architecture is a standard defined by Object Management Group (OMG), [http://www.omg.org/spec/CORBA/](http://www.omg.org/spec/CORBA/)
Following the invocation of the server from Ptolemy II, the generated output from OMC is retrieved as an executable simulation application. This application provides both an interactive and non-interactive simulation runtime.

4.1.1 Interactive Simulation of the Modelica Models

OpenModelica also offers an on-line user-interactive and time synchronous simulation mode. This simulation with user interactivity is known as OpenModelica Interactive (OMI), is part of the simulation runtime core. OMI provides an executable simulation application, running the executable file in an interactive processing mode enabling users to govern the simulation runtime behavior [11].
OMI is composed of two modules, *Control* and *Transfer* which are designated for communication over *TCP/IP* as demonstrated in Figure 2.

![Figure 3. OMI communication architecture.](image)

First, an overview of the interaction mechanism of the Control and Transfer modules is given, then the network configuration on both the client and server sides is described to elaborate the communication between Ptolemy II and OMI.

- **Control module** is the interface between OMI and Ptolemy II which is implemented as a single thread to support parallel tasks and independent reactivity. The Control module is considered as the major controlling and communication instance at the simulation initialization phase as well as managing simulation properties throughout the simulation runtime. The control module also reacts to the feedback from other internal OMI components and sends some messages back to Ptolemy II including error and status messages.

- **Transfer module** gets simulation results from a result manager and sends them to Ptolemy II upon launching a simulation. Additionally, the module employs filter mask property containing names of all properties whose result values are significant to Ptolemy II.
According to the description of the modules composing OMI, some components for running these modules should be designed in both Ptolemy II as a client and OMI as a server. On the server side, OMI, two components for running the Control modules and one for running the Transfer module are designed by OpenModelica developers as configured in Table 1.

Some components should be designed on the client side, Ptolemy II, to be served by the server side components, respectively. Thus, I designed the client side components including two components for running the Control modules and one for running the Transfer module as configured in Table 2. These components are implemented as Java classes in Ptolemy II.

Table 1. OMI server and client components.

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
<th>URL</th>
</tr>
</thead>
<tbody>
<tr>
<td>Control Server</td>
<td>Waits for requests from the UI</td>
<td>By Default, waits for connection on: 127.0.0.1:10501</td>
</tr>
<tr>
<td>Control Client</td>
<td>Replies to the UI and sends out other synchronization messages</td>
<td>By Default, waits for connection on: 127.0.0.1:10501</td>
</tr>
<tr>
<td>Transfer Client</td>
<td>Sends simulation results to a UI</td>
<td>By Default, tries to connect on: 127.0.0.1:10502</td>
</tr>
</tbody>
</table>

Table 2. Ptolemy II server and client components.

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
<th>URL</th>
</tr>
</thead>
<tbody>
<tr>
<td>Control Client</td>
<td>Requests to the OMI Control Server</td>
<td>By Default, tries to connect on: 127.0.0.1:10501</td>
</tr>
<tr>
<td>Control Server</td>
<td>Waits for information from the OMI Control Client</td>
<td>By Default, waits for connection on: 127.0.0.1:10500</td>
</tr>
<tr>
<td>Transfer Server</td>
<td>Waits for simulation results from the OMI Transfer Client</td>
<td>By Default, waits for connection on: 127.0.0.1:10502</td>
</tr>
</tbody>
</table>

As a result of running the key modules and the interaction between them on both the client and the server sides, the simulation result is retrieved in string format from the server and used by the specific components of the Ptolemy II client.

### 4.1.2 Non-Interactive Simulation of the Modelica Models

The executable simulation application which is generated by OMC can also be run using the non-interactive simulation runtime, the batch mode. In contrast to the Interactive Simulation mode of OMC, the batch mode does not offer any user-interactive simulation and it is not possible to have the result back to the client step by step. Running the executable file in batch processing mode results in generating the whole simulation result in MAT, CSV⁹ or PLT¹⁰ file formats. The latter two formats can be utilized through the specific components of the Ptolemy II client.

---

⁹ Comma-Separated Values  
¹⁰ Plain Text Language
4.2 The Software Architecture of Ptolemy II to Create New Extension of OpenModelica

At the beginning of the work, I inspected the carefully designed package structure of Ptolemy II to create custom software extensions to OpenModelica, including some of the key packages and consisting of the Java classes shown in Figure 4 [1].

The actor package shown in Figure 4 includes the principal classes defining the execution process of a model which is the key package that should be inspected to have the OpenModelica extension. For implementing this extension, I created one custom actor and one director.

In the first place, it is essential to investigate the Director class in the actor package to figure out how new director can be added to Ptolemy II.

4.2.1 OpenModelica Director

Directors in Ptolemy II implement Models of Computation (MoC) to define the semantics of a computational model. An MoC is considered to be a collection of rules which govern concurrent execution of the components and the communication between them. As was mentioned earlier in the background chapter, the major emphasis in Ptolemy II is on the heterogeneous combination of MoCs, there are different types of MoCs in Ptolemy II including Continuous time, Discrete time, Process Network, Synchronous Data-Flow and etc. [7].

---

10 Structured on the basis of a set of commands embedded in the ROM of pen plotters that lower the workload for applications programmers for creating plotter output [22].
A suitable MoC should be selected prior to adding a new director. In this integration, the one which is most relevant to the Modelica language should be considered. Some of the Models of Computation in Ptolemy II strongly depend on time such as Discrete-Event and Continuous-Time models, which actors communicate and perform computation based on a logical time line. Continuous-time models are realized using the Ptolemy II continuous-time (CT) domain which models physical processes. The continuous domain supports mixtures of discrete and continuous behaviors. Such combinations are suitable for many cyber-physical systems. Cyber-physical systems are characterized by a mixture of computational and physical processes. Additionally, the continuous dynamics of physical processes are typically represented using Ordinary Differential Equations (ODEs).

One of the fundamental concepts of the Modelica language is the evolution of a continuous-time model continuously due to being composed of variables described using Differential-Algebraic Equations (DAEs).

As the Modelica language designed for continuous and discrete-event modeling of physical systems and variables described using DAEs, continuous-time domain in Ptolemy II which models physical processes and supports mixtures of discrete and continuous behaviors is considered more suitable for the Modelica language. Thus, OpenModelica director is implemented based on the continuous-time director. The OpenModelica director with all its parameters in the Graphical User Interface of Ptolemy II known as Vergil, is displayed in Figure 5.
Each MoC has its own package which requires the design of a new package for the OpenModelica domain. I added the new package of OpenModelica domain to one of the key packages of Ptolemy II called domain, shown in Figure 6. Domain package in Ptolemy II provides domains which implement a MoC, and some of them include their own domain-specific actor libraries.
The package displayed in Figure 6 does not include all the domains and the experimental ones, it is only restricted to the leading ones. As it is shown in Figure 7, I added the OpenModelica domain as a new package to the current domain package of Ptolemy II.
Figure 7. Domain package after adding OpenModelica domain.
The following step is adding the other component which is essential for completing this thesis work, an actor which is known as OpenModelica. The Java class of the OpenModelica actor is implemented in the lib package.

### 4.2.2 OpenModelica Actor

Many models in Ptolemy II are actor-oriented models which are based on connected groups of actors. Viewing a system as a structure of actors emphasizes its causal structure and its concurrent activities, along with their communication and data dependencies [1]. The connections between actors represent variant types of interaction between components in a design, such as the asynchronous message, rendezvous communication, discrete, continuous ones, etc.

The model which supports the actor-oriented design is composed of actors. Actors, the basic building block of a system, are the executable entities which considered the core notion in this structure.

There are two types of actors in terms of the use of hierarchy: composite and atomic actor. There are two types of composite actors, opaque and transparent ones. An opaque actor consists of various computation models within an individual model with a director which is helpful for having a hierarchical heterogeneity. However, the transparent one cannot be executed alone.

Atomic actors are not considered as a Ptolemy II model itself and they should be included in other models to get executed.

In terms of the flow of control in Ptolemy II, when a composite actor is fired, the director inside the composite actor fires the actors of the contained model. In the model, the top-level composite actor contains a director D1, and A2 is a hierarchical composite actor with director D2. Hence, director D1 controls the execution order of actors A1 and A2, and director D2 controls the execution of A3 and A4 whenever A2 is executed as depicted in Figure 22.

![Figure 8. Flow of the control of Ptolemy II’s composite actor.](image)

In addition to the mechanism for categorizing actors in terms of the hierarchy, actors can also fall in two other categories, domain specific or domain polymorphic actor. Domain polymorphic actors operate under the control of multiple domains which leads to improving the reusability of actors and reducing the code duplication. However, domains specific ones works solely under the control of one specific director.
I designed the OpenModelica actor as a non-composite one which only works under the control of the OpenModelica director. The OpenModelica actor with all its parameters in Vergil is displayed in Figure 5.

Figure 9. OpenModelica actor with its sample parameters.
4.3 OpenModelica Execution in Ptolemy II

An execution in Ptolemy II falls into three phases: setup, iterate, and wrapup. The setup phase is divided into two phases, preinitialize and initialize. The preinitialize sub-phase usually handles structural information, such as constructing dynamically created actors, determining the width of ports, and creating receivers. The initialize phase initializes parameters, resets local state, and produces initial tokens. The preinitialize phase of an actor is performed exactly once during the actor’s life of execution, and before any other execution of the actor. The initialization of an actor is performed once after the preinitialize and type resolution. To organize the iterations among actors, an iteration is divided into prefire, fire, and postfire. Prefire checks if the preconditions are fulfilled for the actor to execute, such as the presence of sufficient inputs to complete the iteration. Most of the actions take place during the fire phase, which involves reading inputs, processing data, and producing outputs. MoCs like synchronous reactive models and Continuous Differential Equations compute the fixed point of actor outputs while keeping the state of each actor constant. Thus, when fixed point is reached, the state of an actor is updated which requires the firing of each actor several times before the actor is postfired. Finally, at the end of the execution, wrapup phase taken place exactly once and actors release resources that were allocated during execution [24].

Some MoCs like synchronous reactive models and CT differential equations support fixed-point iteration which enables the computation of the fixed point of actor outputs while keeping the state of each actor constant. Invocation of fire() several times prior to the invocation of postfire() results to updating the state of an actor at the time of reaching the fixed point. However, in this integration, when postfire() is invoked in the OpenModelicaDirector, it returns false in order to stop the invocation of fire() after firing once.

To sum up the execution of OpenModelica in Ptolemy II, in the first place, the OpenModelica actor starts the OpenModelica Compiler(OMC) server in the initialize() method, by invoking initializeServer(). In the fire(), value(s) of the Modelica parameter(s) are modified by Ptolemy IIs' actor(s) prior to the simulation of the Modelica model through OMC. After simulation of the Modelica model, the retrieved result back from OMC server is plotted/displayed through Ptolemy IIs’ actors. Afterwards, the OpenModelica actor sends the result to the output port which enables using the generated result by other Ptolemy IIs’ actors. All these actions are done in fire(). In the final step, OMC is stopped by sending the quit command to the server. All these steps are shown in Figure 10. The functionality of all these methods are elaborated in the following Chapter.
Figure 10. Interaction diagram between Ptolemy II and OpenModelica.
Chapter 5
Implementation

This chapter provides brief information about how OpenModelica components are added to Ptolemy II, the code structure of the implementation and challenges during the implementation.

5.1 Graphical User Interface

For designing the OpenModelica actor, I followed the approach that is already implemented for other actors. For each actor, one file in the Modeling Markup Language (MoML) format should be created in one of the sub-packages of OpenModelica, known as a lib which embraces the domain-specific actors. This package will be described together with all its constituent classes in Section 5.2.

The MoML format is one of the primary mechanisms for designing models which are executed over the network and the persistent file format for Ptolemy II models [14]. The MoML file for OpenModelica actor is illustrated in Figure below.

```
<?xml version="1.0" standalone="no"?>
<!DOCTYPE entity PUBLIC "-//UC Berkeley//DTD MoML 1//EN" "http://ptolemy.eecs.berkeley.edu/xml/dtd/MoML_1.dtd">
<entity name="OpenModelica" class="ptolemy.moml.EntityLibrary">
<configure>
</configure>
</entity>
```

Figure 11. OpenModelica.xml

In Ptolemy II, new directors like OpenModelica are added as experimental directors. There is a MoML file for experimental directors as shown in Figure 12 and
OpenModelicaDirector which is added as a new property with the proper location is highlighted.
5.2 Code structure

As shown in Figure 6 in 4.2.1, domain packages in Ptolemy II are composed of three sub-packages, thus I followed the same rule for creating the OpenModelica package as depicted in Figure 13. The kernel one contains the classes that extend those in the actor or kernel packages of Ptolemy II, the lib package encompasses domain-specific actors and the key classes that are necessary for running the integration. And finally, the demo package which contains the sample Modelica models that are supposed to be built by OMC such as `dcmotor.mo` and `BouncingBall.mo`. In addition to the Modelica models, the key members of demo are individual demonstration files in MoML format.

![OpenModelica sub-packages](image)

Figure 13. OpenModelica sub-packages.

The class diagram of lib package which includes the key classes of this integration is depicted below:
All actor classes in Ptolemy II are written in Java and located in the lib package of the domain. The class that I implemented in Java for the OpenModelica actor is `OpenModelica.java`. The constructor of the OpenModelica actor contains the definition of input/output ports and other parameters of the actor, such as `simulationStartTime`, `processingMode` and `output` port, as highlighted in the below code snippet.

```java
public OpenModelica(CompositeEntity container, String name)
    throws NameDuplicationException, IllegalActionException {
    super(container, name);

    input = new TypedIOPort(this, "input", true, false);
    input.setMultiport(true);

    output = new TypedIOPort(this, "output", false, true);
    output.setMultiport(true);

    fileName = new FileParameter(this, "fileName");
    fileName.setDisplayName("File name");

    processingMode = new StringParameter(this, "processingMode");
    processingMode.setDisplayName("Select the processing mode ");
    processingMode.setExpression("non-interactive");
    processingMode.addChoice("non-interactive");
    processingMode.addChoice("interactive");

    subModel = new StringParameter(this, "subModel");
```
As it is shown in Figure 14, the OpenModelica class is composed of three methods, fire(), initialize() and wrapup(). Most of the main functions for conducting this integration are invoked in fire().

In initialize(), the OpenModelica Compiler(OMC) is started by invoking initializeServer() which is implemented in OMCCOMMAND Java class as it is shown in red marked section of the below Figure.

The main method of OpenModelica class, fire() plays the key role in this integration and most of functions which are involved in this integration are invoked in this
method. The invoked functions in fire() are listed below and highlighted in Figure 18 too.

- `loadModelicaFile`, Loads the Modelica file and library.
- `modifyComponents`, Modifies parameter(s) and variable(s) of the Modelica model before building the model.
- `runModel`, Builds the Modelica model. Then, runs the executable simulation application generated by OMC in both interactive and non-interactive processing modes to generate the result. There are two alternatives for building the model, sending buildModel or simulateModel Modelica expressions to OMC. The one that is selected for this integration is buildModel expression. Due to the fact, through sending simulateModel expression process of generating the executable simulation application and running the executable file happen consecutively, thus the user interface is blocked until the simulation is completely done. However, by sending the buildModel command, the first step occurred, then the user is able to run the generated file manually which gives the user opportunity of having control on executing of the generated result.

Part of the interaction between OpenModelica and Ptolemy II which is implemented by methods in fire() are illustrated in Figure 19.

The next two functions are the constructor of OMIThread Java class and run method of this thread which results to providing the overview of OMIThread Java class in advance of going through the details of these functions.

OMIThread runs key modules of OMI, Transfer and Client modules. Transfer module tries to get simulation results from a result manager and sends them to the Ptolemy II immediately after starting a simulation. The Control module is the interface between OMI and Ptolemy II. It is implemented as a single thread to support parallel tasks and independent reactivity. As the main controlling and communication instance at simulation initialization phase and while simulation is running it manages simulation properties and also behavior. Ptolemy II can permanently send operations as messages to the Control unit, it can react at any time to feedback from the other internal OMI components and it also sends messages to the Ptolemy II, for example error or status messages. The network communication technology TCP/IPv4 is used for sending and receiving messages. The above modules are designated for a communication over TCP/IP. These functionalities are implemented in the thread constructor and run method of the thread as explained below:

- `OMIThread(variableFilter.getExpression(),simulationStopTime.getExpression(), output)`, Constructor of OMIThread, Constructs client and servers, sets the IP and ports of the servers and sets up streams for transferring information between client and servers.
• \texttt{omiThread.run()}, Displays the simulation result step by step according to the start time, stop time and number of intervals of the OpenModelica actor. The formula for calculating the step time is as follow:

\[
\text{step time} = (\text{stop time} - \text{start time}) / \text{number of intervals}
\]

There is one issue, the simulation does not stop automatically at the stop time that is selected as simulationStopTime parameter due to a property of the OpenModelica tool. Thus, during reading the result back from the Transfer server, one condition is set up to check if the stop time is reached, as highlighted in yellow in Figure 17.

The simulation result back from OMC is wrapped as \texttt{StringToken} and sent to the output port of the OpenModelica actor to be displayed by Ptolemy II’s actor or employed by other actors, as highlighted in yellow in Figure 17.

```java
// Checking the status of Transfer socket, connection and stream.
try {
    if (_transferServer.isBound() && _transferConnection.isConnected() && _inFromTransferServer.ready()) {
        try {
            // It returns -1 if the end of the stream is reached.
            // If not, it returns the number of chars that have been read.
            TransferServerBuffer: while ((_inFromTransferServer .read(transferServerBuffer)) != -1) {
                wholeSimulationResult = new String(transferServerBuffer).trim();
                lineSplitResult = wholeSimulationResult .split(lineDelimiter);
                for (int i = 0; i < lineSplitResult.length - 1; i++) {
                    lineSimulationResult = lineSplitResult[i];
                    timeSplitResult = lineSimulationResult .split(timeDelimiter);
                    for (int j = 1; j < timeSplitResult.length; j++) {
                        if (j == 1) {
                            // Stop the simulation when the stopTime is reached.
                            // There is a need to have this condition, because the simulation does not
                            // stop automatically.
                            if ((timeSplitResult[j].toString()).startsWith(_stopTime)) {
                                break TransferServerBuffer;
                            } else {
                                if (outputResult != null) {
                                    outputResult += "At time : ";
                                    + timeSplitResult[j]
                                } else {
```

```java
```
```
outputResult = "At time : " + timeSplitResult[j] + ";
} else {
    outputResult += timeSplitResult[j] + ";
}
else
    outputResult += "\n";
}
try {
    outputPort.send(0, new StringToken(outputResult));
} catch (NoRoomException e) {
    new IOException(e.getMessage()).printStackTrace();
} catch (IllegalActionException e) {
    new IOException(e.getMessage()).printStackTrace();
} catch (IOException e) {
    new IOException(e.getMessage()).printStackTrace();
}

Figure 17. Code snippet of OMThread class.
parameter.getExpression();
} else {
  _omcCommand.modifyComponents(inputPort.toString(), subModel.getExpression(), parameter.getExpression());
}

else {
  _omcLogger.getInfo("There is no component to modify prior to running the model!");
}
}

} catch (ConnectException e) {
  throw new IllegalActionException("Unable to modify components’ values!" + e.getMessage());
}

// There is no value to be passed to the OpenModelica actor’s port and the new value is set by // actors’ parameters.
} else if (!(input.getWidth() > 0)) {
  if (!(parameter.getExpression().isEmpty() && initialValue.getExpression().isEmpty())) {
    try {
      if (baseModel.getExpression().isEmpty()) {
        _omcCommand.modifyComponents(initialValue.getExpression(), subModel.getExpression(), parameter.getExpression());
      } else {
        _omcCommand.modifyComponents(initialValue.getExpression(), baseModel.getExpression(), parameter.getExpression());
      }
    } catch (ConnectException e) {
      throw new IllegalActionException("Unable to modify components’ values of " + baseModel.getExpression() + "!") + e.getMessage());
    }
  } else {
    _omcLogger.getInfo("There is no components to modify prior to running the model!");
  }
}

// Build the Modelica model and run the executable result file.
// Plot the result file of the simulation that is generated in plt format.
try {
  if (!(dependencies.getExpression().isEmpty() && baseModel.getExpression().isEmpty())) {
    _omcCommand.runModel(dependencies.getExpression(), baseModel.getExpression(), simulationStartTime.getExpression(), simulationStopTime.getExpression(), Integer.parseInt(numberOfIntervals.getExpression()), outputFormat.getExpression(), processingMode.getExpression());
    if (outputFormat.getExpression().equalsIgnoreCase("plt") && processingMode.getExpression().equalsIgnoreCase("non-interactive")) {
      // Plot the result file of the simulation that is generated in plt format.
      // Plot the result file of the simulation that is generated in plt format.
    }
  }
}
```java
_omcCommand.plotPltFile(baseModel.getExpression());
}
} else {
    _omcCommand.runModel(fileName.getExpression(),
        subModel.getExpression(),
        simulationStartTime.getExpression(),
        simulationStopTime.getExpression(),
        Integer.parseInt(numberOfIntervals.getExpression()),
        outputFormat.getExpression(),
        processingMode.getExpression());
    if (outputFormat.getExpression().equalsIgnoreCase("plt")
        && processingMode.getExpression().equalsIgnoreCase("non-interactive")
    ) {
        _omcCommand.plotPltFile(subModel.getExpression());
    }
}
// In case of building the model in an interactive mode, client and servers
// are created,
// IP and ports of the servers are set and streams for transferring
// information
// between
// client and servers are set up all in the constructor of the thread.
// Through starting the thread, the simulation result is sent from the
// server to
// Ptolemy II in the string format.
if (processingMode.getExpression().equalsIgnoreCase("interactive")) {
    _omiThread = new OMIThread(variableFilter.getExpression(),
        simulationStopTime.getExpression(), output);
    _omiThread.run();
} catch (UnknownHostException e) {
    e.printStackTrace();
    throw new IllegalActionException("Host Exception: 
        + e.getMessage());
} catch (IOException e) {
    e.printStackTrace();
    throw new IllegalActionException("Socket Connection Error: 
        + e.getMessage());
} catch (ConnectException e) {
    e.printStackTrace();
    throw new IllegalActionException("ServerError: 
        + e.getMessage());
}
```

Figure 18. The code snippet shows the implementation of fire() method one of the key functions of OpenModelica actor.
The last function is \texttt{wrapup()} which invokes \texttt{stopServer()} to leave OpenModelica environment, destroy the OMC process and de-allocate resources as marked in the below Figure.

\begin{figure}[h]
\centering
\includegraphics[width=\textwidth]{fig19.png}
\caption{Functions included in the \texttt{fire} method of OpenModelica actor.}
\end{figure}

\begin{figure}[h]
\centering
\includegraphics[width=\textwidth]{fig20.png}
\caption{Function which is invoked in wrapup method.}
\end{figure}
The implementation of functions which are invoked in initialize(), fire() and wrapup() are implemented by OMCCOMMAND Java class which results to having the most number of lines in the lib package as shown in Table 3.

Table 3. Number of lines in Java classes.

<table>
<thead>
<tr>
<th>Class Name</th>
<th>Number of Lines</th>
</tr>
</thead>
<tbody>
<tr>
<td>OpenModelica</td>
<td>437</td>
</tr>
<tr>
<td>OpenModelicaDirector</td>
<td>84</td>
</tr>
<tr>
<td>OMCCCommand</td>
<td>1273</td>
</tr>
<tr>
<td>IOMCCCommand</td>
<td>138</td>
</tr>
<tr>
<td>OMIThread</td>
<td>400</td>
</tr>
<tr>
<td>CompilerResult</td>
<td>100</td>
</tr>
<tr>
<td>ICompilerResult</td>
<td>69</td>
</tr>
<tr>
<td>ConnectException</td>
<td>68</td>
</tr>
<tr>
<td>CompilerException</td>
<td>6</td>
</tr>
<tr>
<td>OMCLogger</td>
<td>175</td>
</tr>
<tr>
<td>Sum</td>
<td>2750</td>
</tr>
</tbody>
</table>

5.3 Implementation Challenges

5.3.1 Incomplete OpenModelica API

Typically, tool integration is troublesome due to challenges such as API incompatibilities, unstable or undocumented APIs, unclear semantics, syntactic incompatibilities, and unmaintainable code bases.

One of the confronted challenges during the implementation was the partly undocumented API\(^1\) that was clarified with the valuable assistance of one of the OpenModelica developers, Adeel Asghar\(^2\).

5.3.2 Time Challenge

5.3.2.1 Faulty Ptolemy II’s Component

One of the Ptolemy IIs’ components, XYPlotter which is fully explained in Example 2 - Composite model for plotting CSV file by XYPlotter, was faulty throughout the implementation. I reported the bug of this actor to the respective developer to get that resolved, the amount of time spent for resolving the bug was relatively short, but the main time-consuming part was the struggle of figuring out the origin of this fault.

5.3.2.2 Conflicted Design Insight

There were some times during the course of the implementation that OpenModelica and Ptolemy II developers were not on the same page about the design. For instance

---

\(^1\) [https://build.openmodelica.org/Documentation/OpenModelica.Scripting.html](https://build.openmodelica.org/Documentation/OpenModelica.Scripting.html)

\(^2\) [http://www.ida.liu.se/~adeas31/](http://www.ida.liu.se/~adeas31/)
for implementing OMI, from OpenModelica developer’s point of view, it would be better to ask the user for inserting input which should be sent to the server. However, from the Ptolemy developer’s point of view, having user intervention does not have great significance. The mistake of not discussing the primary use case with the Ptolemy II developers prior to beginning the implementation led to the time-consuming process of the design modification.
Chapter 6
Testing

This chapter outlines two testing approaches in Ptolemy II, examines the selected one for this integration and presents three test cases and the comparison between the performance of running two test models in Ptolemy II and OpenModelica.

6.1 System Testing in Ptolemy II

Two approaches for designing tests in Ptolemy II [18]:

2. Unit Testing using Tool Command Language(Tcl), the test suite infrastructure is based on the Tcl test suite code.

Tcl tests are unit tests, whereas tests that use MoML are system tests and may mask unit test bugs. The former approach is employed for designing below test cases due to being relatively straightforward, however, there are some challenges to handle corner cases and test erroneous conditions through writing MoML [18].

The testing infrastructure automatically runs any MoML models located in test/auto directories. Steps for creating the infrastructure for a new test directory:

1. Choose an existing test/ directory that contains an auto/ directory. A good example is $PTII/ptolemy/actor/lib/test.
2. Create your test/ and test/auto/ directories.
   mkdir test/test/auto
3. cd to your test/ directory.
   cd test
4. Copy over the testDefs.tcl and the makefile file from the example directory chosen in step 1 above.
   cp $PTII/ptolemy/actor/lib/test/testDefs.tcl
5. Modify these two files to fit your situation, which may differ from the example situation. In particular:

Then run all the tests by the below commands:

```
make
```

If you run `make` in a test directory that contains tests written in `Tcl` for testing Java classes, then the right thing should just happen [18].

## 6.2 Test Cases

### 6.2.1 Plot the CSV Result File of Running the Executable Simulation Application in Non-Interactive Processing by Xyplotter

The purpose of designing this test case is the verification of the result of the simulation of the Modelica model in Ptolemy II. To ensure that executable simulation application generated by OMC is run properly in a non-interactive mode according to the set parameters of the OpenModelica actor through plotting the result by `XyPlotter`.

<table>
<thead>
<tr>
<th>ID:</th>
<th>S01</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Creation Date:</strong></td>
<td>2013-11-14</td>
</tr>
<tr>
<td><strong>Author:</strong></td>
<td>Mana Mirzaei</td>
</tr>
<tr>
<td><strong>Description:</strong></td>
<td>Plot the CSV result file by <code>XyPlotter</code>.</td>
</tr>
<tr>
<td><strong>Test method:</strong></td>
<td>Automated.</td>
</tr>
<tr>
<td><strong>Input:</strong></td>
<td>Double click on the <code>OpenModelica</code> actor. Fill out the parameters of OpenModelica actor according to Figure 16. Run the model.</td>
</tr>
<tr>
<td><strong>Expected output:</strong></td>
<td>As Figure 17.</td>
</tr>
<tr>
<td><strong>Pass/Fail criteria</strong></td>
<td>Pass, if the output is according to Figure 17. Otherwise, Fail.</td>
</tr>
</tbody>
</table>

### 6.2.2 Plot the PLT Result File of Running the Executable Simulation Application in Non-Interactive Processing

The purpose of designing this test case is the verification of the result of the simulation of the Modelica model in Ptolemy II. To ensure that executable simulation application generated by OMC is run properly in a non-interactive mode according to the set parameters of the OpenModelica actor through plotting the result by `pltplotter`. 

```
6.2.3 Display the Result File of Running the Executable Simulation Application in Interactive Processing

The purpose of designing this test case is the verification of the result of the simulation of the Modelica model in Ptolemy II. To ensure that executable simulation application generated by OMC is run properly in an interactive mode according to the set parameters of the OpenModelica actor through displaying the result by Display actor.

Table 7, Test case S03.

6.3 Performance

In this section, I compare the simulation time of two test models in OpenModelica and Ptolemy II. For OpenModelica, I calculated the simulation time by simulating the
test models via OMC and for Ptolemy II calculated automatically when the simulation is done.

Table 4. Execution time of test models in Ptolemy II and OpenModelica.

<table>
<thead>
<tr>
<th>Name of the Model</th>
<th>Execution Time in Ptolemy II (ms)</th>
<th>Execution Time in OpenModelica (s)</th>
</tr>
</thead>
<tbody>
<tr>
<td>OpenModelicaXYPlotter using dcmotor.mo</td>
<td>25021</td>
<td>30</td>
</tr>
<tr>
<td>OpenModelicaPltPlotter using dcmotor.mo</td>
<td>28673</td>
<td>12</td>
</tr>
</tbody>
</table>

As simulating the models in Ptolemy II involves establishing a connection with OMC, sending the Modelica expressions to OMC and transferring the generated output by OMC back to Ptolemy II, some overhead is to be expected. However it is the tradeoff for simulation in a heterogeneous environment.
Chapter 7
Conclusion

In the course of this thesis work, the first working prototype of the integration of OpenModelica into Ptolemy II environment was implemented. This implementation enables the simulation of the Modelica models via OMC in Ptolemy II without the need for additional libraries which is a constraint in existing implementations. Furthermore, providing the well-documented code and developing in cooperation with the developers of Ptolemy II ensures that the resulting implementation can serve as a base for additional functionalities of this integration as well.

7.1 Future Works

7.1.1 Improve Ptolemy II by Adding OpenModelica Import Menu

Ptolemy II modelers made the suggestion of constructing an OpenModelica actor dynamically with the appropriate ports and parameters by importing Modelica files. Additionally, it was recommended to follow the accomplished approach of importing FMU files in Ptolemy II for developing this suggestion.

7.1.2 Design New Library Known as BCVTB for OpenModelica Library

BCVTB was designed to allow the communication of the simulators Ptolemy, EnergyPlus, Dymola, Matlab, Simulink, Radiance and BACnet. To enable the communication of BCVTB with simulators such as Dymola, BCVTB block should be added to the Modelica models that are built in Dymola. This block is provided by implementing the Modelica Buildings Library [25]. Thus, adding the BCVTB block to the Modelica models that are built by OpenModelica can lead to the co-simulation with other simulators like Simulink.
Appendix A
User Guide

This chapter describes the instruction to install *OpenModelica 1.9.0, Ptolemy II 10.0* and *Eclipse for RCP and RAP Developers*, some examples of the OpenModelica extension are illustrated as well.

7.2 Getting Started

7.2.1 Installation

The tools needed to install are Eclipse for RCP and RAP Developers and OpenModelica.

Install OpenModelica\(^3\) Download and install OpenModelica from the OpenModelica home page. Some completed notes about OpenModelica installation is written by Christopher Brooks\(^4\) as well.

Install Eclipse\(^5\) Follow the instruction in Chess website to install Eclipse and set up Ptolemy II and Eclipse.

7.3 Model Examples

In Ptolemy II, there exist two approaches to specify models. Components can be instantiated or parameterized by Java code or Vergil. Models might be simulations (the execution of a model that presents the behavior of models in a particular environment, the major purpose of the simulation is providing the opportunity to enable testing of the design and have insight about the properties of the design [1]) or implementations (the execution of the system itself). Models can fall into applications or network-integrated programs (applets, servlets, or CORBA services, for example) as well. In this integration, the model is considered simulations and the network-integrated programs via CORBA services, the interconnection between the components of the model is designed by Vergil graphically and the parameterization

\(^3\) http://chess.eecs.berkeley.edu/ptolemyII/wiki/Main/OpenModelica
\(^4\) http://ptolemy.eecs.berkeley.edu/~cxh/
\(^5\) http://chess.eecs.berkeley.edu/ptolemyII/wiki/Main/eclipse/
is carried out through Java code. This section illustrates the design of the integration by providing three examples.
This section illustrates the design of the integration by providing three examples.

7.3.1 Example 1 – Simple Model for Plotting PLT File

To start OpenModelica from Ptolemy II, a model will need to be created. Now the model OpenModelicaPltPlot.xml shown in Figure 21, you can customize OpenModelica actor by double clicking on the actor as parameters depicted in Figure 22 and changes can be committed by clicking Commit button.

![Diagram](Figure_21_OpenModelicaPltPlot.xml)

This model requires that OpenModelica be installed (see http://www.openmodelica.org). In addition, the OPENMODELICA_HOME environment variable must be set and the PATH must be set to include the OPENMODELICA_HOME/bin directory.

Under Mac OS X:

```bash
export OPENMODELICA_HOME=/opt/openmodelica
export PATH=$OPENMODELICA_HOME/bin:$PATH
```

Author: Mana Mirzaei

Figure 21. OpenModelicaPltPlot.xml.
This model demonstrates the OpenModelica actor that communicates with the OMC server by building the model, prior to building the model \texttt{resistor1.R} is changed to 8 in non-interactive processing. There is a possibility of changing multiple variables, “Initialized model parameter(s)” separated by hash and initial variables, “Initial value(s)” separated by comma. Finally, the result is generated in PLT format and plotted by pltplotter16 as depicted in Figure 24.
Using Ptolemy II to conduct the work should be easier than having a monolithic OpenModelica actor that carries out the whole tasks. To avoid having a monolithic actor, RunCompositeActor\(^{16}\), a composite one which executes the contained model completely should be employed. This composite actor is composed of CSVReader\(^{17}\), XYPlotter and RecordDisassembler\(^{18}\) actors along with the SDF director. The specific features of the model is the static order of actor execution and starting the execution upon the availability of required data inputs. The CSVReader actor reads comma separated values and returns a record, the record is sent to the RecordDisassembler in order to select the values for plotting. XYPlotter actor takes two values of fields in the record as an input and plots the values.

The model OpenModelicaXYPlotter.xml shown in Figure 25 demonstrates the OpenModelica actor that can be customized by double clicking on the actor as parameters depicted in Figure 26 and changes can be committed by clicking Commit button. This model communicates with the OMC server by building the model in non-interactive processing, modification of variable(s) is also possible in the same way as explained in Example 1 – Simple model for plotting PLT file. Finally,

---
\(^{16}\)One of the lifecycle management actors, actors that associate ports with parameters of the model, that executes the contained model till completion.
\(^{17}\)One of the key IO actors that “reads files or URLs that are in CSV format, or comma-separated values”.
\(^{18}\)“Extracts fields from a record, the name of the output port must match the name of the field”.

---

\[Figure 24.\] Plotting OpenModelica simulation by pltplotter.

### 7.3.2 Example 2 - Composite Model for Plotting CSV File by XYPlotter

Using Ptolemy II to conduct the work should be easier than having a monolithic OpenModelica actor that carries out the whole tasks. To avoid having a monolithic actor, RunCompositeActor\(^{16}\), a composite one which executes the contained model completely should be employed. This composite actor is composed of CSVReader\(^{17}\), XYPlotter and RecordDisassembler\(^{18}\) actors along with the SDF director. The specific features of the model is the static order of actor execution and starting the execution upon the availability of required data inputs. The CSVReader actor reads comma separated values and returns a record, the record is sent to the RecordDisassembler in order to select the values for plotting. XYPlotter actor takes two values of fields in the record as an input and plots the values.

The model OpenModelicaXYPlotter.xml shown in Figure 25 demonstrates the OpenModelica actor that can be customized by double clicking on the actor as parameters depicted in Figure 26 and changes can be committed by clicking Commit button. This model communicates with the OMC server by building the model in non-interactive processing, modification of variable(s) is also possible in the same way as explained in Example 1 – Simple model for plotting PLT file. Finally,
the result by Display\textsuperscript{19} actor is displayed and plotted by XYPlotter\textsuperscript{20} as depicted in Figure 27.

Figure 25. OpenModelicaXYPlotter.xml and the contained model.

Figure 26. Parameters of OpenModelica in OpenModelicaXYPlotter.xml.

\textsuperscript{19} “Displays values of inputs in a text window that opens when the model is executed.”

\textsuperscript{20} “Plots input data from one input port vs. input data from its other input port.”
In the Composite actor that is utilized for plotting CSV format, OpenModelicaDirector controls the execution order of OpenModelica actor and RunCompositeActor, and SDF director which is known as the inside director of RunCompositeActor controls the execution of CSVReader, RecordDisassembler and XYPlotter whenever RunCompositeActor is executed as illustrated in Figure 19. The flow of control of RunCompositeActor contained in OpenModelicaXYPlotter model is depicted in Figure 28 according to Figure 8 which illustrates the flow of control of composite actors in Ptolemy II.
7.3.3 Example 3 – Composite Model for Displaying the Result of Simulation in Interactive Processing Mode

The model OpenModelicaInteractive.xml shown in Figure 29 demonstrates OpenModelica actor, that can be customized by double clicking on the actor as parameters depicted in Figure 30 and changes are committed by clicking Commit button. New value(s) for variable(s) of the Modelica model can be set by “initial value(s)” parameter or Ramp actor\(^\text{21}\) “init” parameter depicted in Figure 31 which can be customized in the same way as OpenModelica actor can be. The value of “init” parameter of Ramp actor overrides the value of “initial value(s)” parameter of OpenModelica actor. The result of this modification is highlighted in green in Figure 32.

In fact, a compatibility between the number of variables and values leads to place a constraint when Ramp actor is utilized. Ramp actor only generates one integer, thus

---

21. “Sources library yields a continuously increasing/decreasing output sequence.”
as a result of the stated constraint one variable can be solely modified by Ramp “init” parameter, otherwise mismatch error will be generated.

Following the modification, the model is built in interactive processing and the result of the simulation is demonstrated by Display actor in accordance with “Filter for displaying the simulation result” parameter of the OpenModelica actor. There exists another constraint for the number of filter variables which cannot be more than two as illustrated in Figure 30 and 32 by yellow highlight.

The step time of the simulation in interactive processing is calculated by following formula:

\[
\text{Simulation step time} = \frac{(\text{Simulation stop time} - \text{Simulation start time})}{(\text{Number of intervals})}
\]

According to the above formula, the step time is \(0.0002 \ [\ (1.0 - 0.0)/500]\) in this example. The calculated step time results in the start of simulation at 0.0002 and completion at 0.0998, as demonstrated in Figure 30 and 32 by red highlight.

Figure 29. OpenModelicaInteractive.xml with the contained model.
Figure 30. Parameters of OpenModelica actor in OpenModelicaInteractive.xml.

Figure 31. Parameters of Ramp actor in OpenModelicaInteractive.xml.
Figure 32. Displaying part of simulation result of OpenModelicaInteractive.xml by Display actor.

7.3.4 Example 4 – Composite Model for Modifying Parameter of the Super Model

The model OpenModelicaInherit.xml shown in Figure 33 is designed similarly to OpenModelicaInteractive.xml excluding Ramp actor in the contained model. The leading purpose of this example is modifying variable(s) in the base model, set parameters depicted in Figure 34 and 35 for non-interactive and interactive processing respectively include generated results as well.
This model modifies value(s) of variable(s) of the base model solely, prior to building the model in any processing runtime and finally display the result of simulation.

CompositeOpenModelica is composed of OpenModelica actor and one output port.

OpenModelica actor sets value(s) for variable(s) of the base model prior to running the model in any processing mode and the simulation result will be displayed by Display actor. The result of modification is shown in generated result file in CSV format.

**Figure 33.** OpenModelicaInherit.xml with the contained model.
Figure 34. Set parameter of OpenModelica actor with part of the CSV generated file in non-interactive processing.
Figure 35. Set parameter of OpenModelica actor with part of the simulation result in interactive processing.
Bibliography


14 Ptolemy II Features, Department of Electrical Engineering and Computer Science, University of California, Berkeley, USA. [http://ptolemy.eecs.berkeley.edu/ptolemyII/summary.htm](http://ptolemy.eecs.berkeley.edu/ptolemyII/summary.htm), Last accessed Nov. 2013


18 Ptolemy II Testing,  
http://ptolemy.eecs.berkeley.edu/ptolemyII/ptII8.0/ptII/doc/coding/testing.htm,  

19 Open Source Modelica Consortium,  
https://openmodelica.org/index.php/home/consortium,  

20 UsingVergil,  
http://ptolemy.eecs.berkeley.edu/ptolemyII/ptIIlatest/ptII/doc/design/usingVergil/usingVergil.pdf,  
Last Accessed June 2013.

21 Eclipse for RCP and RAP developers,  
http://www.eclipse.org/downloads/packages/eclipse-rcp-andrapdevelopers/keplersr1,  
Last Accessed Nov 28.

22 What is PLT,  
http://www.coolutils.com/Formats/PLT,  
Last Accessed Dec. 1st.


# Table of Acronyms

<table>
<thead>
<tr>
<th>Acronym</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>BCVTB</td>
<td>Building Controls Virtual Test Bed</td>
</tr>
<tr>
<td>BSD</td>
<td>Berkeley Software Distribution</td>
</tr>
<tr>
<td>CHESS</td>
<td>Center for Hybrid and Embedded Software Systems</td>
</tr>
<tr>
<td>CORBA</td>
<td>Common Object Request Broker Architecture</td>
</tr>
<tr>
<td>DAE</td>
<td>Differential Algebraic Equation</td>
</tr>
<tr>
<td>FMI</td>
<td>Functional Mockup Interface</td>
</tr>
<tr>
<td>FMU</td>
<td>Functional Mockup Unit</td>
</tr>
<tr>
<td>FSM</td>
<td>Finite State Machine</td>
</tr>
<tr>
<td>GUI</td>
<td>Graphical User Interface</td>
</tr>
<tr>
<td>IDA</td>
<td>Department of Computer and Information Science</td>
</tr>
<tr>
<td>MoC</td>
<td>Model of Computation</td>
</tr>
<tr>
<td>MoML</td>
<td>Modeling Markup Language</td>
</tr>
<tr>
<td>PELAB</td>
<td>Programming Environment Laboratory</td>
</tr>
<tr>
<td>ODE</td>
<td>Ordinary Differential Equation</td>
</tr>
<tr>
<td>OMC</td>
<td>OpenModelica Compiler</td>
</tr>
<tr>
<td>OMI</td>
<td>OpenModelica Interactive</td>
</tr>
<tr>
<td>OSI</td>
<td>Open Source Initiative</td>
</tr>
<tr>
<td>OSMC</td>
<td>Open Source Modelica Consortium</td>
</tr>
<tr>
<td>UI</td>
<td>User Interface</td>
</tr>
</tbody>
</table>
På svenska
Detta dokument hålls tillgängligt på Internet – eller dess framtida ersättare – under en längre tid från publiceringsdatum under förutsättning att inga extra-ordinära omständigheter uppstår.

Tillgång till dokumentet innebär tillstånd för var och en att läsa, ladda ner, skriva ut enstaka kopior för enskilt bruk och att använda det oförändrat för ickekommersiell forskning och för undervisning. Överföring av upphovsrätten vid en senare tidpunkt kan inte upphäva detta tillstånd. All annan användning av dokumentet kräver upphovsmannens medgivande. För att garantera äktheten, säkerheten och tillgängligheten finns det lösningar av teknisk och administrativ art.

Upphovsmannens ideella rätt innefattar rätt att bli nämd som upphovsman i den omfattning som god sed kräver vid användning av dokumentet på ovan beskrivna sätt samt skydd mot att dokumentet ändras eller presenteras i sådan form eller i sådant sammanhang som är kränkande för upphovsmannens litterära eller konstnärliga anseende eller egenart.

För ytterligare information om Linköping University Electronic Press se förlagets hemsida http://www.ep.liu.se/

In English
The publishers will keep this document online on the Internet - or its possible replacement - for a considerable time from the date of publication barring exceptional circumstances.

The online availability of the document implies a permanent permission for anyone to read, to download, to print out single copies for your own use and to use it unchanged for any non-commercial research and educational purpose. Subsequent transfers of copyright cannot revoke this permission. All other uses of the document are conditional on the consent of the copyright owner. The publisher has taken technical and administrative measures to assure authenticity, security and accessibility.

According to intellectual property law the author has the right to be mentioned when his/her work is accessed as described above and to be protected against infringement.

For additional information about the Linköping University Electronic Press and its procedures for publication and for assurance of document integrity, please refer to its WWW home page: http://www.ep.liu.se/

© [Mana för- och Mirzaei]