Eclipse BIRT Plug-ins for Dynamic Piecewise Constant and Event Time-Series

by

Savitha Sundaramoorthi

A Thesis Presented in Partial Fulfillment
of the Requirements for the Degree
Master of Science

Approved November 2014 by the
Graduate Supervisory Committee:

Hessam S. Sarjoughian, Chair
Ross Maciejewiski
Georgios Fainekos

ARIZONA STATE UNIVERSITY

May 2015

ABSTRACT

Time-series plots are used in many scientific and engineering applications. In this thesis, two new plug-ins for piecewise constant and event time-series are developed within the Eclipse BIRT (Business Intelligence and Reporting Tools) framework. These customizable plug-ins support superdense time, which is required for plotting the dynamics of Parallel DEVS models. These plug-ins are designed to receive time-based alphanumerical data sets from external computing sources, which can then be dynamically plotted. Static and dynamic time-series plotting are demonstrated in two settings. First, as standalone plug-ins, they can be used to create static plots, which can then be included in BIRT reports. Second, the plug-ins are integrated into the DEVS-Suite simulator where runtime simulated data generated from model components are dynamically plotted. Visual representation of data sets can simplify and improve model verification and simulation validation.

To my parents

ACKNOWLEDGMENTS

I would like to offer my sincere gratitude to my thesis advisor Dr. Hessam Sarjoughian for introducing me to the area of Modeling and Simulation and providing me with this research opportunity. His patience and knowledge throughout the course of research has been invaluable for the successful completion. His professional guidance and encouragement gave me strength to learn and appreciate the challenges faced in this research.

I would like to thank my thesis committee members Dr. Ross Maciejewiski and Dr. Georgios Fainekos for their time and efforts in reviewing this thesis.

I would like to thank all the members in ASU-ACIMS (Arizona Centre for Integrative Modeling and Simulation) for their help in discussing my research and their support during difficult times.

Finally, I would like to thank my parents and my friends for their endless support.

TABLE OF CONTENTS

Page

LIST OF FIGURES……viii

CHAPTER

1 INTRODUCTION ……..…….1

1.1 Research Objectives And Approach ……...…….3

1.2 Contributions To This Study ……...……..….4

1.3 Organization of Thesis……..……...….5

2 BACKGROUND AND RELATED WORK..……...….7

2.1 Plotting Tools.……..……...……...7

2.1.1 Overview Of Charting Tools……..……..7

2.1.2 Inference From Charting Tools...……..…….10

2.2 Eclipse……..……...…....12

2.2.1 Overview Of Eclipse Framework……...……...…….12

2.2.2 Eclipse Platform Runtime And Plugin Architecture...……...……13

2.3 Business Intelligence And Reporting Tools (BIRT)……...……..….16

2.3.1 Overview Of BIRT Framework……...……...…...16

2.3.2 BIRT Architecture And Functionalities.…..……...……...……16

2.3.3 BIRT Process Flow……...……...……..19

2.3.4 BIRT Chart Component And Its Extension Points…..……....…….20

3 DESIGN SPECIFICATIONS AND IMPLEMENTATION OF TIME- SERIES PLUG-INS……....…….21

3.1 Trajectories For Mixed Linear And Superdense Time Bases…….21

CHAPTER Page

3.2 Requirements For Piecewise Constant Plugin……..…..…….21

3.2.1 Plotting Requirements……..……...23

3.2.2 Requirements For User Customization Of The Plot……..…….24

3.3 Design Specifications For Piecewise Constant Plot…..……..……25

3.3.1 Extension Of Schema Files For Piecewise Constant Chart Model…..…...26

3.3.2 Generation Of Code From Schema Files.……..…….29

3.3.3 Runtime Extensions……..……..33

3.3.4 Design Time Extensions……..……...39

3.4 Rendering Piecewise Constant Plots……...……..……...43

3.4.1 Pre-Computation Of Data Set And Creating Zero-Time Advance Models Models……...…….……...43

3.4.2 Render Series..…….……...…….…...45

3.4.3 Algorithms.……...……..…..……..45

3.5 Requirements For Org.Eclipse.Birt.Chart.Timeview.Event Plugin…….…....48

3.5.1 Plotting Requirements……..……..48

3.5.2 User Customization Of The Plots……..…….49

3.6 Design Specifications For Event Plot.……..…….……..50

3.6.1 Extension Of Schema Files For Event Constant Chart Model….....50

3.6.2 Runtime Extensions For Event Plots……..……....51

3.6.3 Design Time Extensions For Event Plots...……...52

4 INTEGRATION OF TIME-SERIES PLUG-INS WITH DEVS-SUITE…….....52

4.1 Timeview Component In DEVS-Suite 2.1…..……...54

4.2 Timeview Component In DEVS-Suite 3.0.……56

CHAPTER Page

4.2.1 Design Decisions For Integrating Time-Series Plugins With Devs-Suite……...56

4.2.2 Workflow Illustrating The Usage Of Time-Series Plug-Ins To Create, Render And Customize Plots…….59

4.3 Packaging Devs-Suite 3.0…....64

4.3.1 Distributing DEVS-Suite As JAR File……...…….64

4.4 Feature Comparison Between DEVS-Suite 2.1 And 3.0…….65

4.5 Performance Analysis…….66

4.5.1 Load Analysis On Extended BIRT Plug-Ins (Analysis 1)……..…….....66

4.5.2 Analysis Of Correctness Of Time View Component Under Different Simulation Speed…….……...……...69

5 DEMONSTRATION……...……..71

5.1 Demonstration Of Time-Series Plug-Ins With BIRT RCP…….……..71

5.2 Demonstration Of Time-Series Plugins With DEVS-Suite.…….77

6 CONCLUSIONS AND FUTURE WORK……...…….83

6.1 Conclusion…….……...83

6.2 Future Work….…….84

REFERENCES……...……..…85

APPENDIX

A DEPENDENCY ANALYSIS AND ENVIRONMNETAL SETUP FOR DEVS-SUITE 3.0……...……..87

CHAPTER Page

B SETUP OF ECLIPSE IDE FOR BUILDING, TESTING AND DEPLOYING TIME SERIES PLUG-INS IN APPENDIX B……...……..91

LIST OF FIGURES

Figure Page

1. Extended Time Series Plug-Ins With Possible Input Sources And Outputs……...4

2. Eclipse Plug-In Architecture ……12

3. Plug-Ins Extensions And Extension Points……14

4. BIRT Architecture……..17

5. Process Flow Of Rendering A BIRT Report ……19

6. Stacked Linear And Superdense Time Segment ……...19

7. Package Diagram For Piecewise Constant Plugin ……24

8. Design View For Piecewiseconstant.xsd File……...28

9. Design View Of Piecewiseconstantdata.xsd File……..29

10. Class Diagram For Org.Eclipse.Birt.Chart.Piecewiseconstant.Model Package..29

11. Class Diagram For Classes In Org.Eclipse.Birt.Chart.Piecewiseconstant.Data..32

12. Class Diagram For The Classes Involved In Datasetprocessor And Datapointdefinition Extension Point……..37

13. Piecewiseconstantrenderer Class And Its Relationship With BIRT Classes…...38

14. Class Diagram For Classes In Design Time Extensions……...……..39

15. Pre- Computation Of Main Plots' Data Sets – The Algorithm……....46

16.Creation Of Confluentmodel Instances And Attributes – The Algorithm……47

17.Rendering Piecewise Constant Series – The Algorithm……...48

18. Design View For Event.xsd File…….49

19. Package Diagram For Timeview Component In DEVS-Suite……55

20. Integration Of Time-Series Plug-Ins With DEVS-Suite……..57

21. Sequence Diagram For Creation Of Chart Objects……..60

Figure Page

22. Sequence Diagram For Rendering Chart……..62

23. Sequence Diagram For Chart Customization……...63

24.Packaging Of DEVS-Suite……64

25. Feature Comparison Between DEVS-Suite 2.1 And DEVS-Suite 3.0……65

26. Load Analysis Of Time-Series Plug-Ins……..67

27. Correctness Analysis Of Time View Component At Speed 1.0E-4……69

28. Correctness Analysis Of Time View Component At Speed 0.01……69

29. Initial Report Project Configuration Wizard……71

30. Chart Wizard For Properties Configuration…….72

31. Configuration Of Data Source For The Chart……..73

32. Selection Of File And Columns For The Data Set……...74

33. Setting The X And Y-Axes Data Sets …….75

34. Configuring Chart Series Properties...……..76

35. Report View Of The Piecewise Constant Chart……...77

36. Model Configuration And Tracking Wizard……78

37. Choosing The Components To Be Tracked, Plots To Be Rendered And Setting The Basic Chart Properties…….79

38. Plots Rendered With Zero-Time Advance..…….79

*39.*Customization Wizard For Advanced Chart Properties……81

40. Plots After Customization……82

CHAPTER 1

INTRODUCTION

Time-series charts are important in many scientific, engineering and statistical applications. They are useful for displaying the dynamics of systems in terms of time-varying variables in alternative two-dimensional charts. The time base for these charts can be real and integer numbers. Line charts can be used to render piecewise constant and event data sets, which are commonly used in simulation tools.

The DEVS-Suite Modeling and Simulation tool (ACIMS, 2015) (DEVS-Suite, 2009), supporting Parallel DEVS modeling formalism (Chow, 1994), offers basic support for generating runtime piecewise constant and event charts (or plots or trajectories). This simulator is built using the Model-View-Control-Façade architecture pattern. The View component receives notification from the Model and can generate input, output, and state trajectories in tabular and piecewise constant and event charts. The piecewise constant (or event) chart consists of piecewise constant (event) segments. The X-Axis for both chart types displays the passage of time. The time as a discrete or continuous quantity is defined to be linear and monotonically increasing. The Y-Axis represents a variable that symbolizes the input, output, or state of a model being simulated. The Y-Axis values can be strings, integers, and real numbers. Numeric values can range from negative to positive infinity inclusive. The string and first class objects are tabulated or displayed in terms of their "print" properties in the Java programming language. The piecewise constant chart is used for the phase and sigma- state variables of any atomic model component and the simulator's time of last event and the time to next event. An Event chart consists of

event segments that represent a sequence of events ordered in time. They are used for input and output trajectories.

Parallel discrete-time and discrete-event models may allow multiple inputs (or outputs) to occur at an instance of time due to a model being in one or multiple states for a zero-time duration (Chow, 1994). The functions responsible for processing these inputs, changing states, and producing outputs can be mathematically specified in terms of superdense time (Manna, Z, Pnueli, A, 1993). The DEVS-Suite simulator (DEVS-Suite, 2009) does not directly support visualizing the concept for superdense time (Manna, Z, Pnueli, A, 1993), which is necessary for segments that have zero-time duration. This major limitation is common to many classical and contemporary modeling and simulation tools like Ptolemy (ptolemyII, 2010). Analysis has been done on different plotting tools to provide improvised time-series plots overcoming the above-mentioned limitation. Some well-known plotting tools are Business Intelligent and Reporting Tools (BIRT, 2014), JFreeChart (JFreeChart, 2000), and d3.js (Data-Driven Documents, 2013).

While many plotting tools provide the APIs to render the piecewise constant and event plots, the Business Intelligence and Reporting Tools (BIRT) offers the most flexible infrastructure for extending or creating new kinds of charts. It enables reporting and business intelligence capabilities for rich client and web applications, especially those based on Java (JAVA, 2013) and J2EE (J2EE, 2013). It has two main components: a visual report designer for creating BIRT reports within the Eclipse IDE and a runtime component for generating reports that can be deployed in any Java application. The BIRT also includes a charting engine that can be integrated into the Report Designer and also be used to integrate charts into Java applications. BIRT

supports different kinds of charts such as Bar, Pie, Line, Scatter, and Gantt charts. One of its major capabilities is a flexible, rich built-in chart customization wizard. As BIRT is built using the Eclipse plug-in framework, one can extend the BIRT framework using extension points and use its API to develop new chart types.

1.1 RESEARCH OBJECTIVES AND APPROACH

The primary goal of this research is to develop time-series plug-ins – i.e., piecewise constant and event plug-ins extending the BIRT framework that could be integrated in Java and RCP applications (see Figure 1). In comparison with other time-series plots, the developed plug-ins offer the following new capabilities.

- The existing time-series plots in DEVS-Suite simulator render only the last simulated value when multiple events occur at zero-time steps. To visually render trajectory charts for a finite number of contiguous state changes at an instance of time (i.e., instantaneous state transitions) and thus multiple inputs and outputs occurring an instance of time, visual representations accommodating superdense time need to be developed using the BIRT plug-ins.
- The plug-ins support displaying positive and negative infinity values.
- The plug-ins support static and dynamic plotting with customized features for the DEVS-Suite simulator.

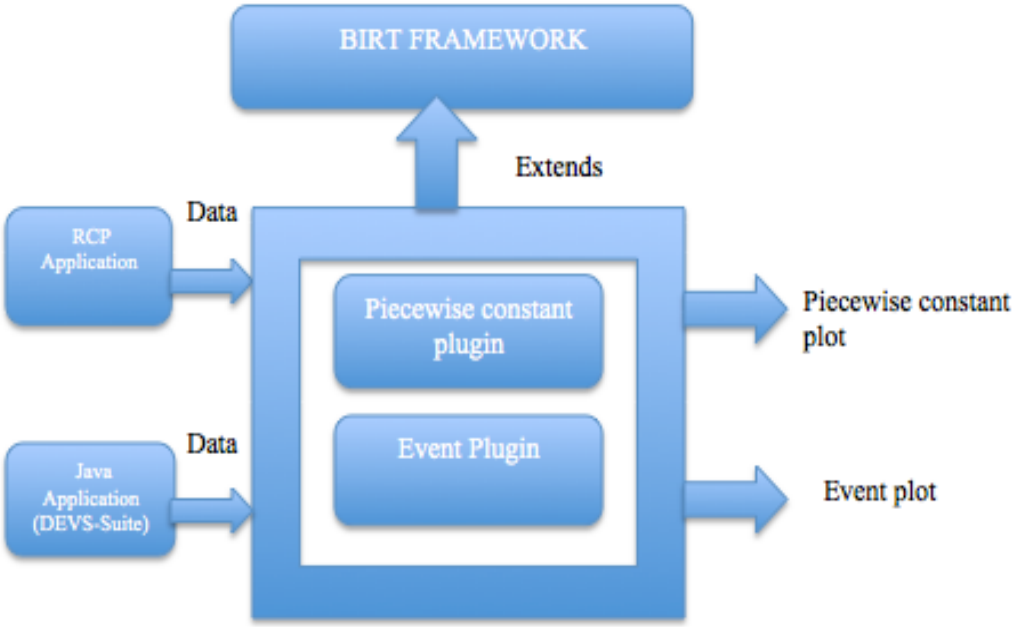

Figure 1. Concept for Developing Piecewise and Event Time-Series Plug-ins

The approach for developing the plug-ins is to understand the Eclipse plug-in framework and the extension points provided by BIRT. As BIRT does not have the entire necessary API to support the capabilities of the superdense time trajectory with negative and positive infinity values concept and formulation proposed in (Sarjoughian, Sundaramoorthi , 2015), the BIRT framework and in particular, its charting concept and design are studied in detail. With the developed plug-ins, integration with JAVA and RCP applications is also tested. Analysis on external dependencies for the application using the developed plug-ins is also done due to the plug-ins' dependency on Eclipse and BIRT frameworks.

1.2 CONTRIBUTION

The developed time-series plug-ins offer the following new capabilities:

- Plug-ins are developed as part of the Business Intelligence and Reporting Tools (a RCP application). Use of the plug-ins is demonstrated with the BIRT Report Designer Engine.
- The DEVS-Suite simulator supports generating runtime plots that have zero-time segments while allowing state, input, and output to have positive and negative infinity values.
- The capabilities of the charting engines are demonstrated using prototypical parallel DEVS models that have zero-time segments.

1.3 THESIS ORGANIZATION

Chapter 2 presents an overview and background on different plotting tools analyzed, the Eclipse Plug-in Architecture and the BIRT-charting API. It details the BIRT chart engine including its extension points and the process flow.

Chapter 3 describes the approach needed for plotting trajectories with a superdense time base. The software requirements and a design for piecewise and event trajectory plug-ins are developed using the UML class and XML schema diagrams.

Chapter 4 describes the implementation details for the piecewise and event chart extensions developed using the BIRT base time-series plug-ins. The integration process of these plug-ins with the DEVS-Suite simulator is detailed. All the new or modified visualization capabilities of the simulator are described in terms of class diagrams, sequence diagrams, and basic requirements. It also provides a performance analysis on the developed plug-ins.

Chapter 5 presents an example of a coupled model demonstrating the plotting capabilities for the piecewise constant and event chart plug-ins. The BIRT report

generated with the developed plug-ins is also demonstrated with an example. Chapter 6 discusses the conclusion and future work, such as supporting plots having multiple data sets, development of new chart types, and integration of the BIRT reporting engine into the DEVS-Suite simulator.

CHAPTER 2

BACKGROUND AND RELATED WORK

Background work for this thesis focuses on analysis of plotting tools and using BIRT as the visualization tool of choice. Explanations of BIRT architecture, Eclipse's plug-in framework that BIRT extends, the BIRT process flow, important chart concepts, terminologies, and the major extension points offered to extend chart follow.

2.1 PLOTTING TOOLS

Data visualization plays a crucial role in many science and engineering areas. Techniques including data normalization, filtering, zooming, etc. are important for creating useful visual data representations. A work that inspired and relates to our work can be used to create linear time charts and any finite segment of a trajectory can be zoomed in with arbitrary precision (Kincaid, R., 2010). In this type of chart, the time axis is linear, but the granularity of time can vary. Therefore, the spatial representation of a plot can be scaled uniformly unlike the superdense time axis and variables that may have positive and negative infinity values. To create the time-series charts described in Chapter 1, open source tools JFreeChart, D3.js, and BIRT were analyzed in terms of usability and performance.

2.1.1 Overview of Charting Tools

JFreeChart

JFreeChart is an open source chart library for the Java™ platform and is designed for use in Java applications, applets, and J2EE application (JFreeChart, 2013). It can generate pie charts, bar charts, line charts, scatter plots, time-series charts (including

moving averages, high-low-open-close charts and candlestick plots), Gantt charts, and more. It provides features such as data access from any implementation of the defined interfaces, the ability to export to PNG and JPEG image file formats or to any format with a Graphics2D implementation, tool tips, interactive zooming, chart mouse events, annotations, and HTML image map generation.

The time-series plots provided in JFreeChart is just a line chart using data obtained via the XYDataset interface with x-values displayed as dates on the domain axis (JFreeChart, 2013). The TimeSeries class can be used to represent values observed at other intervals (annual, daily, hourly, seconds etc.). It also provides built-in customization features to modify the chart attributes, plot attributes, and axis attributes. It also provides dynamic plotting through an event notification mechanism that allows it to respond to changes to any component of the chart.

Data Driven Documents (D3)

Data-Driven Documents or D3 is a JavaScript library for creating and manipulating documents using data sets (Bostock, M., Ogievetsky, V., 2011). It provides a set of libraries for transparent approach to visualization for the web. Rather than hide the underlying scenegraph within a toolkit-specific abstraction, D3 enables direct inspection and manipulation of document object models. With D3, one can selectively bind the input data to document elements applying dynamic transformations to modify and generate the content through data joins. D3 also supports rich user libraries for interaction and animation (Bostock, M., Ogievetsky, V., 2011).

Graphite

Graphite written in Python is a real-time graphing system, which performs storing numbers that change over time and plotting them (Graphite, 2011). This functionality is provided as a network service that is both easy to use and highly scalable. One writes application that collects numeric time-series data and send it to Graphite's processing backend, carbon, which stores in Graphite's specialized database, whisper. As whisper is a fixed size database, one needs to configure in advance on how much data we intend to store and the precision level, the lowesr being 1 second. For instance, one could store data with 1-minute precision (meaning one will have one data point for each minute) for 3 days (Graphite, 2011). This data can then be visualized through graphite's web interface, a Django webapp (Graphite, 2011) based on Cairo (Cairo, 2014), a 2D graphics library. Currently, it supports point plot, bar plot, pie plot, density plot, surface plot and Image plot (Graphite, 2011). There exist other client side dashboards some of which are based on Data Driven Documents.

BIRT

As explained in (Weatherby, Bondur, & Chatalbasheva, 2011), BIRT is a charting and reporting application and is tightly integrated with the Eclipse frameworks and platforms. BIRT is implemented as a set of Eclipse plug-ins and provides the functionality for all BIRT components, including BIRT applications, the engines that drive those applications, and supporting application programming interfaces (APIs). BIRT has a rich charting engine providing APIs to generate charts and associate them with data from a data source. The data source could be a(n) flat file, excel, database etc. This chart engine can be used not only in BIRT applications

but also can function independently by integrating with any Java application. Bar, area, bubble, cone, difference, Gantt, line, pyramid scatter, stock, and tube charts are supported by BIRT. One can make use of BIRT's plug-in architecture to create new types of charts through the plug-ins' extension points.

2.1.2 Inference from Charting Tools

Usability: Based on the background study done on the different charting tools, they do not offer support for creating the kinds of charts with the properties mentioned in Chapter 1. Tools such as JFreeChart, D3, Graphite and BIRT provide line or time-series charts without direct support for the superdense time trajectories with negative and positive infinity values proposed and defined in (Sarjoughian,, Sundaramoorthi, 2015). The time-series tools Graphite (Graphite, 2011), OpenTSDB (Deri, L., Mainardi, S. ,2012) supports storage of only real -time data, which is more useful for purposes such as monitoring and graphing the performance of computer systems. Moreover the visualization provided by the time-series plotting tools such as Graphite does not support the formulation mentioned in (Sarjoughian, Sundaramoorthi, 2015). There exist many client side dashboards for Graphite based on D3.js (Bostock, M., Ogievetsky, V., 2011), which is mainly intended for web-based visualization by speaking to the Document Object Model elements. The plots they provide are mainly intended for understanding patterns, trends, etc. Unfortunately for designing and experimenting with dynamical simulation models for concurrent systems, such charts are insufficient. These tools also support 3D charts where one axis is used for time variable, one for continuous variable, and one for discrete variable. One crucial difference is that 2D charts, unlike 3D charts, can be easily stacked together.

To support rendering of plots with superdense time and values including negative to positive infinity, we needed to extend the existing tools' framework to provide our own model, data set and rendering mechanism. As BIRT is based on Eclipse plug-in architecture, it offers extension points to define our own chart types by reusing the existing framework. BIRT also provides a chart wizard for building and customizing the chart. One could also add preferred customization features to the chart wizard through an extension point, unlike JFreeChart. Support for re-usability and a strong user base makes BIRT the tool of preference over the discussed charting tools.

Performance: Unlike JFreeChart, to improve performance, the BRT chart engine also uses an event object cache (Weatherby, Bondur, & Chatalbasheva, 2011). This cache stores previously created objects to be rendered for reuse in the cache. The device renderer provides a convenience method to retrieve objects from the cache. If the object does not exist in the cache, it is created, stored in the cache, and returned. Compared to other tools, a disadvantage observed with BIRT is adding a large collection of external JAR files to the class path because of its dependency on Eclipse plug-ins, which leads to an increase in the size of the application.

After considering the trade-offs, we considered BIRT to be our tool of choice because of its robust, extensible framework for plotting and customization and its increased level of performance for rendering.

2.2 ECLIPSE

2.2.1 Overview of Eclipse Framework

The Eclipse Software Development Kit (SDK) often referred to as Eclipse (see Figure 2), is both the leading Java™ integrated development environment (IDE) and the tool available for building products based on the Eclipse Platform. The Eclipse SDK has a layered structure and is a combination of several Eclipse projects, including Platform, Java Development Tools (JDT), and the Plug-in Development Environment (PDE).

In its entirety, the Eclipse Platform contains the functionality required to build IDEs. However, the Eclipse Platform is itself a composition. The Eclipse Rich Client Platform (RCP) is one such subset of components and by using it, is possible to build arbitrary applications in diverse domains. Some of the components in the Eclipse Platform, highlight the subset that makes up the RCP. JFace and SWT are packages native to Eclipse based on Java Swing API. Eclipse IDE itself (Workbench IDE) is built on top of the Rich Client Platform and JDT and PDE are tools plugged into the Workbench IDE.The manner in which each subsystem in the platform interacts with each other is shown in Figure 2. Another example is BIRT extending the Rich Client Platform framework to use the components and add its own features.

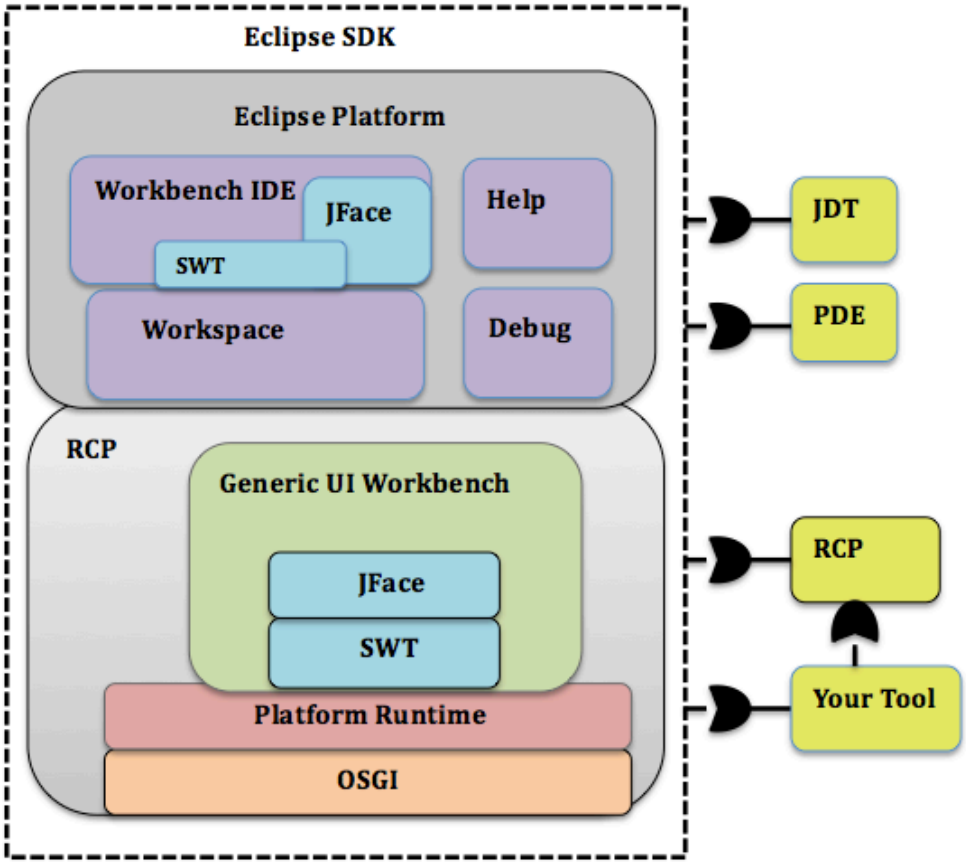

Figure 2. Eclipse Plug-in Architecture

2.2.2 Eclipse Platform Runtime and Plug-in Architecture

As mentioned in (D'Anjou, Fairbrother, Kehn, Kellerman, & McCarthy), Eclipse is a collection of loosely bound yet interconnected pieces of code. How these pieces of code are "discovered" and how they extend and discover each other captures the fundamental principles of the Eclipse architecture. These functional units are called plug-ins. A small kernel known as the platform runtime runs on top of OSGI (Open Services Gateway Initiative) framework, as shown in Figure 2. This framework provides a dynamic modular architecture, which has been used in many applications such as Eclipse Equinox. Except for the platform runtime, all of Eclipse Platform's functionality is located in plug-ins.

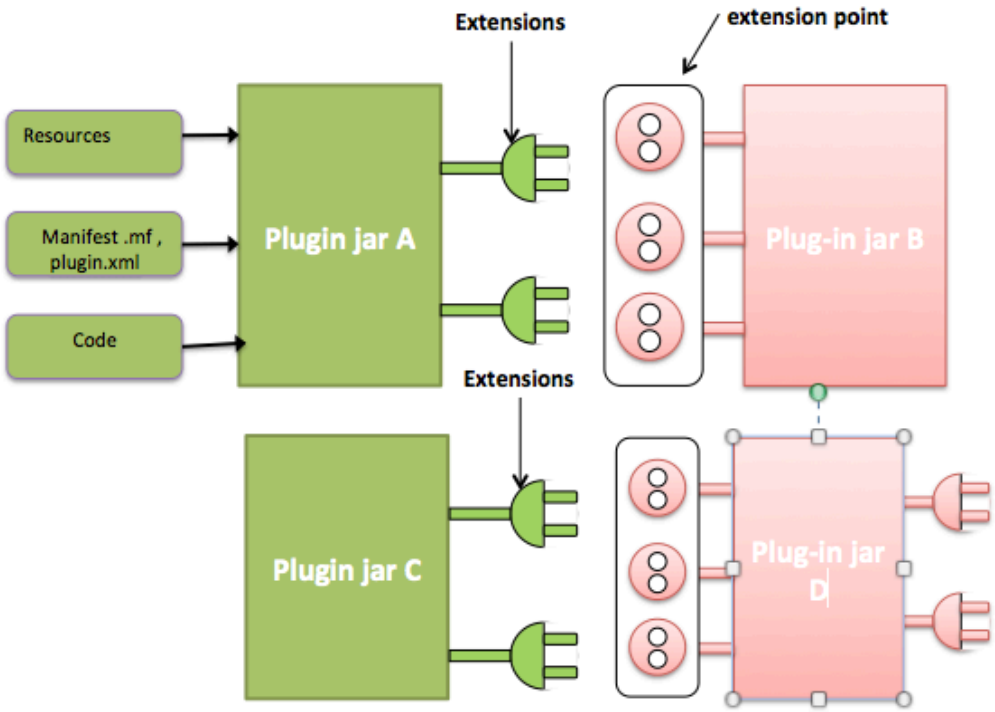

Figure 3. Plug-ins Extensions and Extension Points

A plug-in is a self-contained bundle and it contains the code and resources that it needs to run. A plug-in is self-describing; has its own ID and says what it contributes to other plug-ins through extensions and what it requires from other plug-ins through extension points. This information is described in plug-in manifest, which has a pair of files - manifest.mf and plugin.xml files. The manifest.mf file is an OSGI-bundle manifest that describes the plug-in runtime dependencies; the plugin.xml file is an XML-based description of the plug-in's extensions and extension points. The plug-in containing the code, resources, and manifest file are packaged and delivered as a JAR file, as shown in Figure 3 as Plug-in JAR A.

Extension points: As described above, an extension point is the mechanism by which, the application's current functionality can be extended. The Eclipse workbench UI is itself an example of a plug-in that defines a number of extension points where other plug-ins can contribute toolbar and menu actions, dialogs, editors, etc. (Silva, 2009)

Extension points are used to:

- Create new functionality (a new action set, view, or editor) or increase the functionality of an existing plug-in (contribute a new menu choice).
- Define data-only configuration extensions (help contribution).

Extensions: A plug-in provides an extension to contribute to another plug-in based on the contract defined by an existing extension point. Contributions can be code or data. A plug-in can provide only extensions or extension points or provide both. As shown in Figure 3, Plug-in A only has extensions by contributing to Plug-in B, which only provides extension points. Plug-in D provides extension points and has extensions by contributing to Plug-in C.

On start-up, the Platform Runtime discovers the set of available plug-ins, reads their manifest files, and builds an in-memory plug-in registry. The Platform matches extension declarations by name with their corresponding extension point declarations. Any problems, such as extensions missing extension points, are detected and logged. The resulting plug-in registry is available via the Platform API. Plug-ins can also be added, replaced, or deleted after startup. At runtime it can determine the extension points and extensions supplied without running the plug-in(s); this declarative approach provides a small memory footprint and fast performance. Therefore, several

plug-ins could be installed, but is not activated unless the user or application requests a particular function (Silva, 2009).

2.3 BUSINESS INTELLIGENCE AND REPORTING TOOLS (BIRT)

2.3.1 Overview of BIRT Framework

As described in (Weatherby, Bondur, & Chatalbasheva, 2011), BIRT is implemented as a set of Eclipse plug-ins and provides the functionality for all BIRT components. Figure 4 shows BIRT extending the Eclipse framework. The relationships between BIRT and the Eclipse components can be viewed as a stack. Each tier in the stack depends upon, extends, and integrates with the tier below it.

2.3.2 BIRT Architecture and Functionalities

BIRT Applications: As shown in Figure 4, the applications are present at the topmost tier. BIRT core component plug-ins extended from Eclipse frameworks, the RCP platform, and other BIRT plug-ins that extend core component plug-ins are grouped into different applications. The grouping is based on the functionalities they provide. There are four main BIRT applications:

- BIRT RCP Report Designer: This is an RCP Application with a simplified report design interface.
- BIRT Report Designer: This runs as a set of Eclipse plug-ins and is almost the same as the BIRT RCP Report Designer. This allows us to develop reports and is used within the Eclipse workbench IDE.
- BIRT Report Viewer: It is a web application and provides all the necessary files to deploy to most of the J2EE Application servers.

- Chart Builder: It provides a chart wizard to expedite the process of chart development and can be integrated into Java applications.

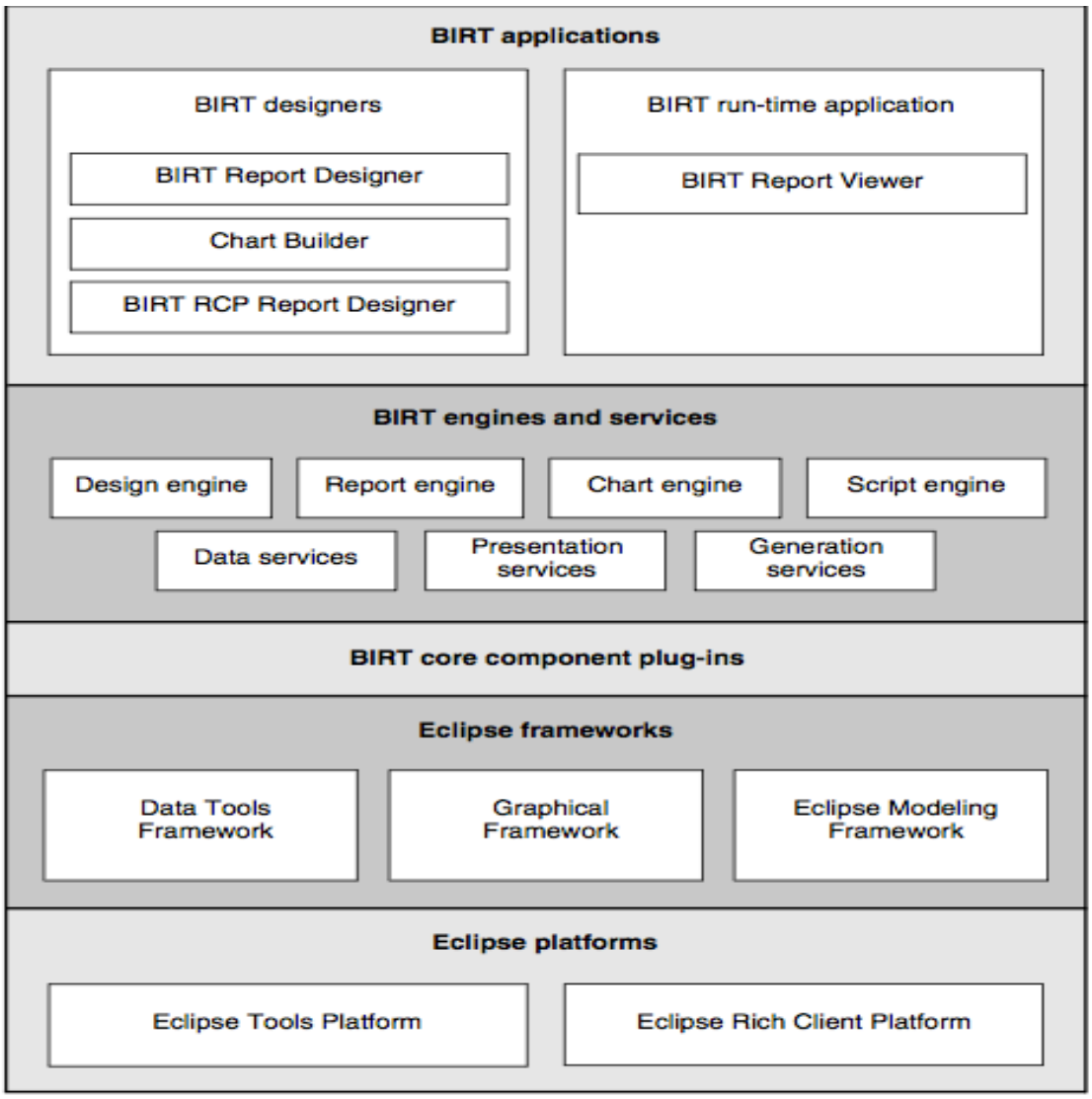

Figure 4. BIRT Architecture (adapted from (Weatherby, Bondur, & Chatalbasheva, 2011))

BIRT Engines and Services: BIRT has several engines like report engine, design engine, and chart engine, which are a set of Java APIs that provide basic functionality. Different services are offered using the Java APIs provided by these engines .

- Design Engine: It contains APIs to validate and generate a report design file

of the extension .rpt. These APIs validate the elements and structure of the design file against Report Object Model (ROM) specification provided by BIRT.

- Report Engine: It contains APIs that support integrating the runtime part of BIRT into Java applications and it provides the ability to view the output in many formats like PDF, DOC, HTML, etc. It also offers extension points to support custom output formats.
- Chart Engine: It contains the APIs to generate charts and associate them with data from a data source. The data source could be a(n) flat file, excel, database, etc. Use of this chart engine is not restricted to a BIRT application. It can function independently by integrating with any Java application. In this research, we use the chart engine APIs to integrate the developed plug-ins with the DEVS-Suite application.
- Script Engine: It contains an API that provides a powerful scripting capability that enables a report developer to create custom code to control various aspects of report creation.

BIRT services are a set of Java classes that provide functionality using the API provided from different engines (Weatherby, Bondur, & Chatalbasheva, 2011).

- Data services: The data services communicate with the ODA framework to retrieve data. This ODA framework is provided by the Eclipse Platform to manage the native drives, open connections, and manage data requests. The data service also offers functionalities such as sorting, grouping, aggregating, and filtering on the retrieved data.

- Generation services: It is present within the report engine. Its main functionality is to connect to the data sources specified in a report design, to retrieve and process the data using the data engine, to create the report layout, and to generate the report document.
- Presentation services: It process the report document created by the generation services and renders the report to the requested format and the layout specified in the design.

2.3.3 BIRT Process Flow

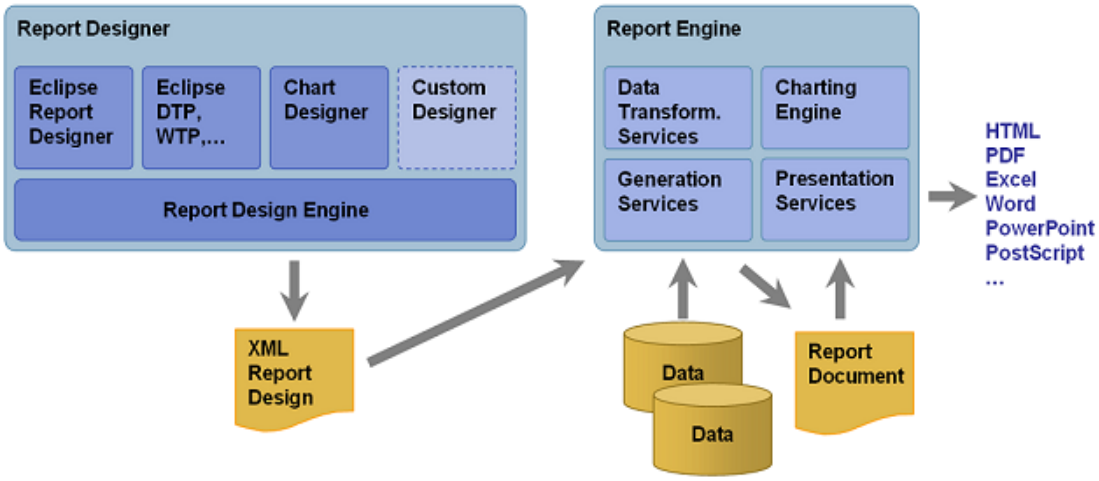

Figure 5. Process Flow of Rendering a BIRT Report (adapted from (Actuate, 2013))

As explained in (Weatherby, Bondur, & Chatalbasheva, 2011), the BIRT Report Designer application provides facilities like drag and drop for easy report design. Once the report is designed, the design file is consumed by the Report Design Engine and produces the XML report design files. The BIRT Report Engine consumes the XML Report design files, fetches the appropriate data from the source and emits the output in the desired format (Figure 5). The functionalities to fetch the data and to

generate the output are provided by the data, generation, and presentation services. BIRT provides an integrated charting component, which supports a wide variety of charts and can be used in the report.

2.2.4 BIRT Chart Component and Its Extension Points

BIRT provides various components to create a report and one among them is BIRT chart components. The BIRT chart components can be used in the report or in standalone java applications. As the thesis focuses on developing time-series chart plug-ins, we focus on chart component. BIRT chart supports charts with axes and charts without axes. Bar, area, bubble, cone, difference, Gantt, line, pyramid scatter, stock, and tube charts are supported by ChartWithAxesImpl and pie and radar charts are supported by the ChartWithoutAxesImpl class. Each chart can have its own sub type. The key visual chart components are movable areas called blocks that include plot, title, and legend areas. The other important component of chart is the series, which controls the set of values to be plotted and other attributes related to rendering the data points like data labels, markers, line thickness, color type, etc. Charts display the values from the series as data points in charts with axes. There are two kinds of series; Category series are values that category or the x-axis display and orthogonal series are values that y-axis displays are created for charts with axes.

BIRT provides extension points to add a new chart type to the BIRT Chart Engine. These extension points are classified as Runtime and Design time extension points and are required to extend the time-series plots. This is described in detail in Chapter 3.

CHAPTER 3

DESIGN SPECIFICATIONS AND IMPLEMENTATION OF TIME-SERIES PLUG-INS

This chapter focuses on the conceptualization of trajectories for mixed linear and superdense time bases, requirements, design specification, and the rendering logic for the time-series plug-ins—piecewise constant plug-in and event plug-in.

3.1 CONCEPTUALIZATION OF TRAJECTORIES FOR MIXED LINEAR AND SUPERDENSE TIME BASES

States, inputs, and outputs of any dynamical model are defined in terms of linear (Kim, Sarjoughian, 2009) and superdense time (Manna, Z, Pnueli, A, 1993). Time is represented by a variable **t**. We specify every time-dependent input, state, or output of a model, represented in terms of a variable **v** and variable **t**. Linear trajectory, which is a concatenation of one or more segments (piecewise constant or event segments), is conceptualized (Zeigler, Praehofer, & Kim , 2000) and implemented in DEVS-Suite 2.1. Concepts for mixed linear and superdense time trajectories are explained as follows.

In contrast to the duration of linear time segments that are defined to be greater than 0 and less than infinity, the superdense time segments are defined to have a duration of zero (measured in terms of the model and simulation time). This can be defined to have a finite spatial length, which can be in pixels and are labeled as t[0], t[1],..,t[n]. Given the segment between $t_1[0]$ and $t_1[1]$ with $\Delta t = 0$ (i.e., $t_1[0] \preccurlyeq t_1[1]$) they can be mapped to x_2 and x_3 with $\Delta x = \chi$. The x_2 and x_3 spatial values are strictly ordered (i.e., $x_2 \prec x_3$) even though their corresponding time values are weakly

simultaneous. For details refer to (Sarjoughian, Sundaramoorthi 2015).

Composing linear and superdense time segments as just described has few shortcomings (Sarjoughian, Sundaramoorthi 2015). Adding or removing superdense time segments causes the time axis of the piecewise constant and event charts to expand or shrink. This is undesirable when the charts are being plotted dynamically. To overcome these problems, a new visualization concept for superdense time axis has been formulated as shown in Figure 6 (Sarjoughian, Sundaramoorthi, 2015). In this approach, separate linear and superdense time axes are defined. This is important as it simplifies visualization of the charts. Another benefit of this approach is that charts can be scaled. When there are several series of superdense time segments, they are separated and therefore simpler to plot and view.

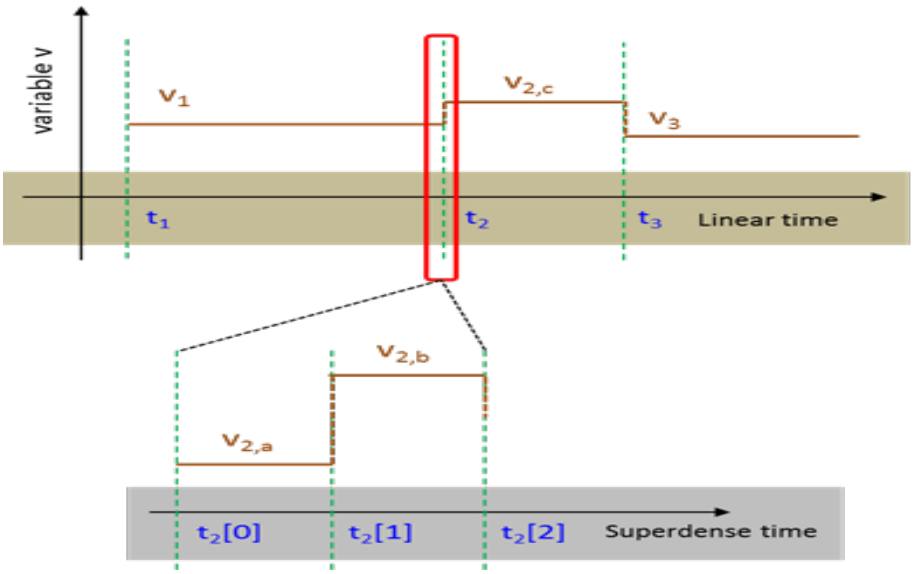

Figure 6. Stacked Linear and Superdense Time Segments.

3.2 REQUIREMENTS FOR PIECEWISE CONSTANT PLUG-IN

The following requirements for piecewise constant and event chart plug-ins are derived from the superdense time charts with infinity values that are defined and formulated in (Sarjoughian, Sundaramoorthi 2015). In the following, the description provided for the piecewise constant chart type equally applies to event chart type.

3.2.1 Plotting Requirements

Stacked linear and superdense time segments are plotted as a main plot with zero-time advance plots, respectively.

Axes and data values: The piecewise constant chart renders the values that are sequentially ordered with time. So, the X-Axis of the main plot represents time instances, which are positive real numbers, and the ordering is monotonic and displayed in linear space. The X-Axis for zero-time advance plots represents the indexed time t[0], t[1], ..,t[n] and is text based. The values of Y-Axis for both the plots are integers and can vary from positive infinity to negative infinity and the scale could be linear or logarithmic. Numerical mapping should be introduced if string values needs to be rendered as piecewise constant. For example the phase of the discrete event model such as {active, busy, active} is mapped to {1, 2, 1}. The Y-axis values for segments are labeled if they are string. The plug-ins should also provide logic to separate the zero-time advance data values from the main data values to plot them as separate piecewise constant charts below the main chart.

Visualization of Infinity: Since infinity needs to be plotted relative to a finite value set, the plug-in must provide logic for infinity value to be mapped to a number larger than the largest number in the set. If the value set is known a-priori, it is relatively

simple to assign a finite value to infinity, which can be visually rendered well relative to the numbers (or strings) in the set. However, in a dynamical setting, the largest number cannot be known a-priori. Therefore, the (positive or negative) infinity value can be defined to be a percentage larger than the largest (positive or negative) value in the value set. This is represented as a horizontal band that occupies 10% (default) of the plot's positive and negative portions of Y-Axis, respectively. The upper bound of the infinity is represented by a red dotted line and labeled. The data value set including the assigned finite value to infinity can be mapped to the spatial axis rendered linearly in pixels.

Dynamic Plotting: As discussed in Chapter 1, one of the main features of the plot is to support dynamic plotting in the form of scrolling. So, the plug-in should remember the last scrolled orthogonal (Y-axis value) value to maintain the continuity of the plot.

3.2.2 Requirements for User Customization of the Plot

The customization features of the chart's properties, which are common to chart with axes, are already provided by BIRT. In addition to this, the plug-in should provide the necessary customization features that are special to the piecewise constant chart. The user should be able to customize the main- and zero- time advance plots' series line properties and series labels. The user should be allowed to enable or disable the positive and negative infinity bands and adjust the width of the infinity band by specifying the desired percentage. The user should also be allowed to enable or disable the viewing of zero-time advance plots.

3.3 DESIGN SPECIFICATIONS FOR PIECEWISE CONSTANT PLOT

In order to create the piecewise constant plug-in, the first step was to determine the extensions this plug-in uses based on our requirements. Based on the major extension points that BIRT provides to create custom charts (Weatherby, Bondur, & Chatalbasheva, 2011), the extensions of this plug-in can be classified into Runtime and Design Time Plug-in extensions.

The package diagram (see Figure 7) shows the packages involved in piecewise constant plug-in and the important relationships between them. The packages org.eclipse.birt.chart.piecewiseconstant.model and org.eclipse.birt.chart. piecewiseconstant.data, that are generated by BIRT, extending the XSD files using EMF must be present before using the other extension points. Let us have a detailed analysis about the XSD files used, runtime and the design time extensions.

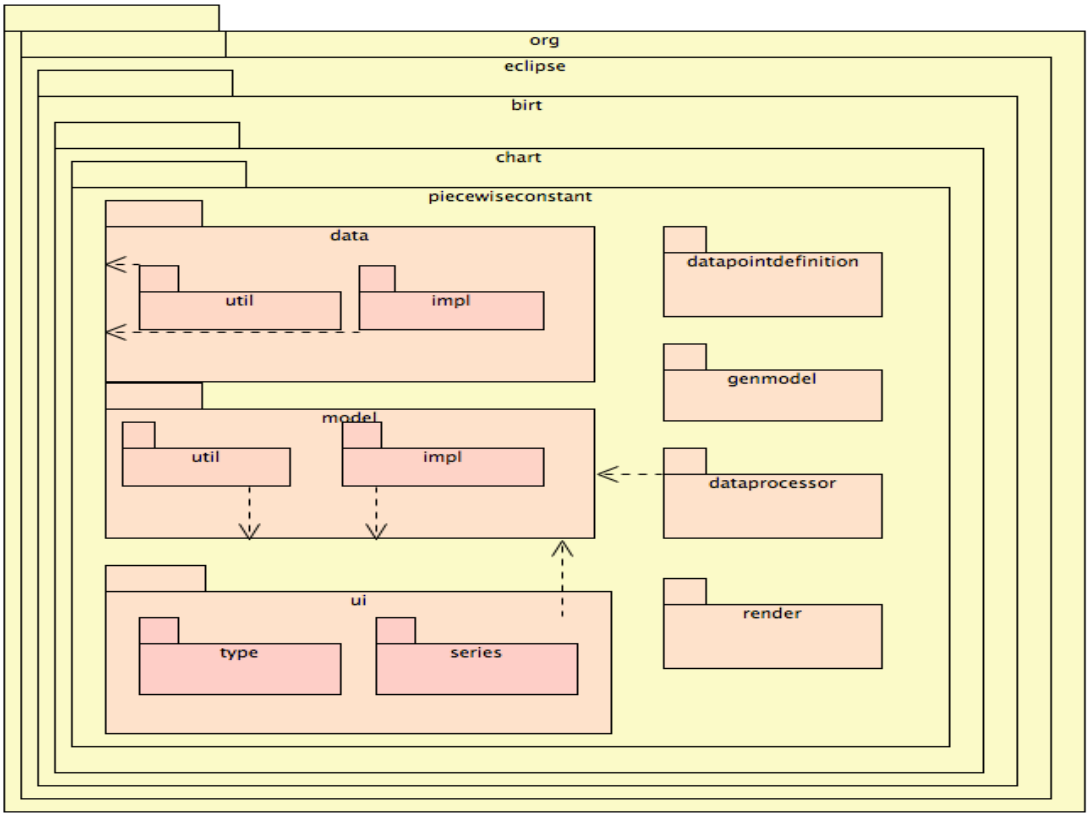

Figure 7. Package Diagram for Piecewise Constant Plug-in

3.3.1 Extension of Schema Files for Piecewise Constant Chart Model

Piecewise constant chart has its unique properties to enable and disable infinity, zero-time advance plots, etc. So, a new chart model was defined using XSD files. The different kinds of XSD files provided by BIRT are attribute.xsd, component.xsd, layout.xsd, data.xsd, type.xsd, and model.xsd.

attribute.xsd: This defines attributes, which are associated with various components of the chart, such as line attributes, Axis Type, Tick Style, data points, markers, etc.

layout.xsd: This defines the layout of the various blocks that make up a chart, including Legend, Plot, and Title blocks.

model.xsd: This defines the chart model and common properties of a chart, which currently include chartWithAxes and chartWithoutAxes extending from chart model.

data.xsd: This defines the data types used in the chart, including basic data types and extended data types such as NumberDataSet, TextDataSet, GanttDataSet, etc.

component.xsd: This defines the components of a chart, such as scales, axis, grid, label, and base-level series. Figure 8 represents the base-level series, which is a complex type in the component.xsd file.

type.xsd: This defines each series type supported by the engine. Each type extends from the base-series type from the component.xsd, which includes the series types like BarSeries, GanttSeries, LineSeries, etc.

PiecewiseConstant.xsd: Though LineSeries in type.xsd file has properties close to the PiecewiseConstant series, we needed to add more elements for zero-time advance plots & positive and negative infinity bands. So, we created a new PiecewiseConstant series by extending the base-level series from the component.xsd file. This includes the following additional elements (see Figure 8):

- LineAttributes: This defines the properties of the lines to be connected in piecewise constant and event graphs.
- showPositiveInfinityBand: This accepts a Boolean value, which tells if the positive infinity band should be displayed in the graph.
- showNegativeInfinityBand: This accepts a Boolean value, which tells if the negative infinity band should be displayed in the graph.
- positiveInfinityPercent: This accepts a double value, which tells the percentage of space allocated for the positive infinity band with respect to the positive orthogonal axis.

- negativeInfinityPercent: This accepts a double value, which represents the percentage of space allocated for the negative infinity band with respect to the negative orthogonal axis.
- showConfluentPlot: This accepts a Boolean value, which enables or disables the zero-time advance plots (confluent plots).
- confluentAxesLineAttributes: This defines the properties of the axis line such as color and thickness for zero-time advance plots.

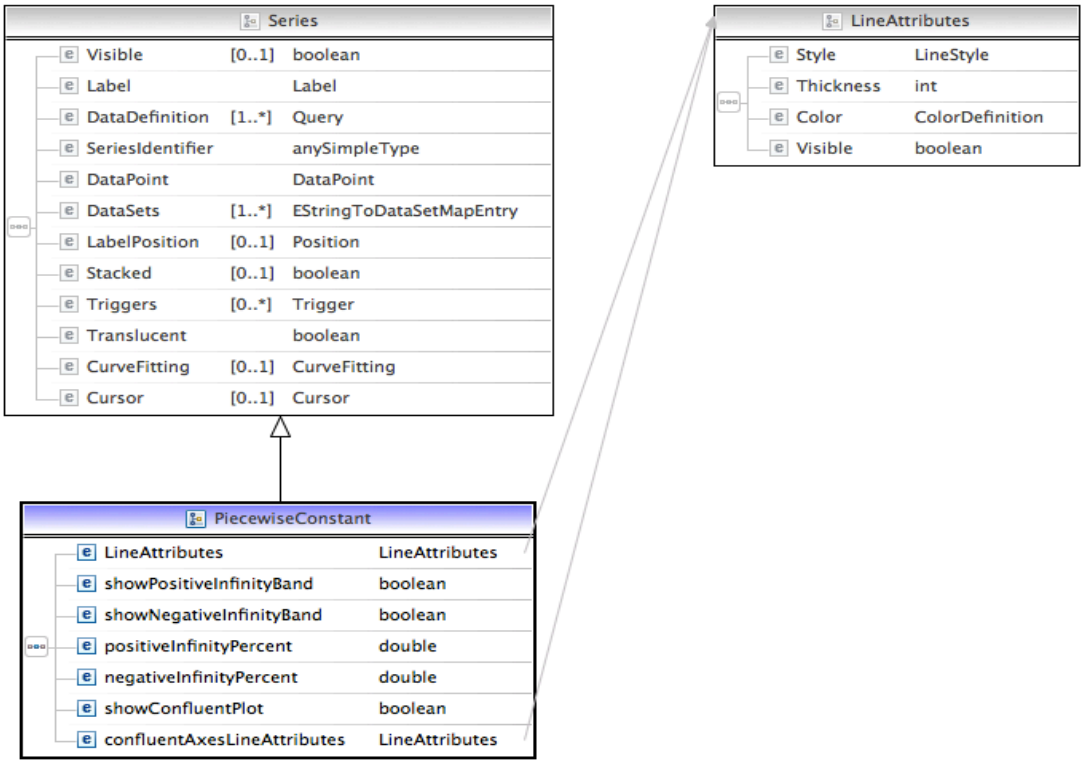

Figure 8. Design View for PiecewiseConstant.xsd File

PiecewiseConstantData.xsd: Piecewise constant chart requires a new data set for series associated with Y-Axis. So we created a new PiecewiseConstantDataSet as given in Figure 9 by extending the Dataset from the data.xsd file. We did not add new

elements to the PiecewiseConsatntDataSet schema file but the extension was done for future behavioral contribution in the generated class.

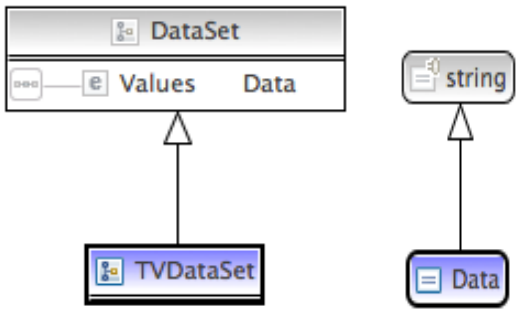

Figure 9. Design View of PiecewiseConstantData.xsd File

3.3.2. Generation of Code from Schema Files

After extending the schema files PiecewiseConstant.XSD and PiecewiseConstantData.xsd, BIRT uses EMF to generate the packages related to PiecewiseConstant series and the PiecewiseConstant dataset from the schema files.

The org.eclipse.birt.chart.piecewiseconstant.model, org.eclipse.birt.chart.piecewiseconstant.model, org.eclipse.birt.chart. piecewiseconstant.model.impl, and org.eclipse.birt.chart.piecewiseconstant.model.util packages were generated using the PiecewiseConstant.xsd file and must be present before extending the runtime time extension points. Package org.eclipse.birt.chart. piecewiseconstant.model contains the interface PiecewiseConstant, PiecewiseconstantPackage and PiecewiseConstantFactory. PiecewiseConstant interface contains the getter and setter method declarations for the attributes of the PiecewiseConstant series and is inherited by org.eclipse.birt.chart.piecewiseconstant. model.impl.PiecewiseConstantImpl. Although EMF provided most of the required code, there were few methods in PiecewiseConstantImpl class that were written manually.

- getDisplayName(): Returns a user friendly string name for the chart.
- create(): Allows an instance of PiecewiseConstantImpl class to be created and initialized.
- Initialize(): Sets several default attributes such as line attributes, confluentAxesLineAttributes.
- copyInstance(): Copies an instance of PiecewiseConstant series and is called when the chart engine makes a copy of the PiecewiseConstant. This method should also be defined in the PiecewiseConstant interface.

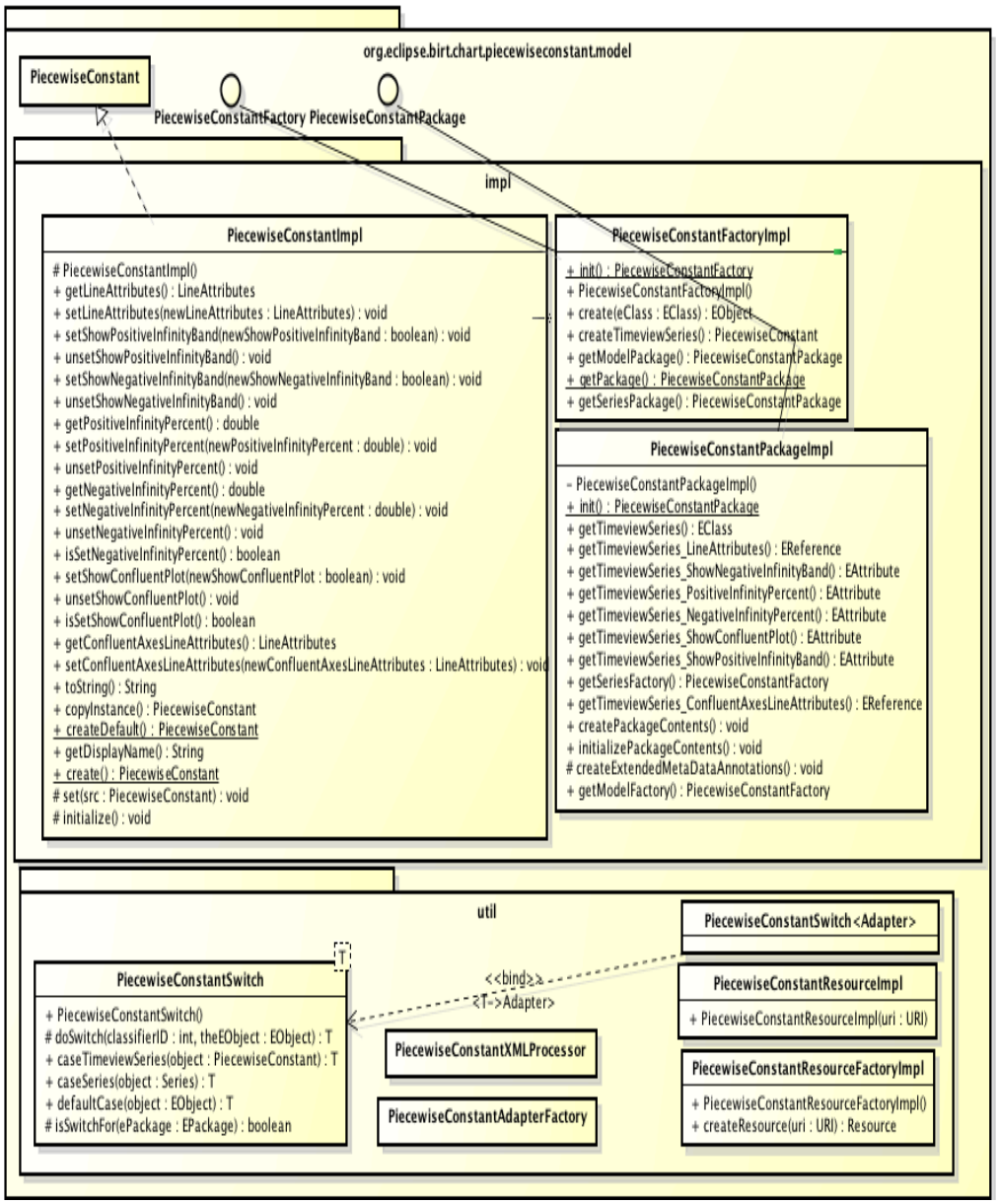

Figure 10. Class Diagram for org.eclipse.birt.chart.piecewiseconstant.model Package

Other classes in the org.eclipse.birt.chart.piecewiseconstant.model.impl and all the classes in org.eclipse.birt.chart.piecewiseconstant.model.util package are related to EMF. See Figure 10, for the classes and the important relationships between them.

The org.eclipse.birt.chart.piecewiseconstant.data, org.eclipse.birt.chart. piecewiseconstant.data.impl, and org.eclipse.birt.chart.piecewiseconstant.data.util

packages were generated using the PiecewiseConstant.xsd file and must be present before extending the runtime time extension points. The PiecewiseConstantDataSetImpl class in the org.eclipse.birt.chart. piecewiseconstant. data.impl package has get and set methods for the instantiation of data elements in the data sets. We also provided a create method for convenient instantiation of data sets. The other classes in the package are related to EMF. See Figure 11 for the important relationships between the classes related to the PiecewiseConstant data set.

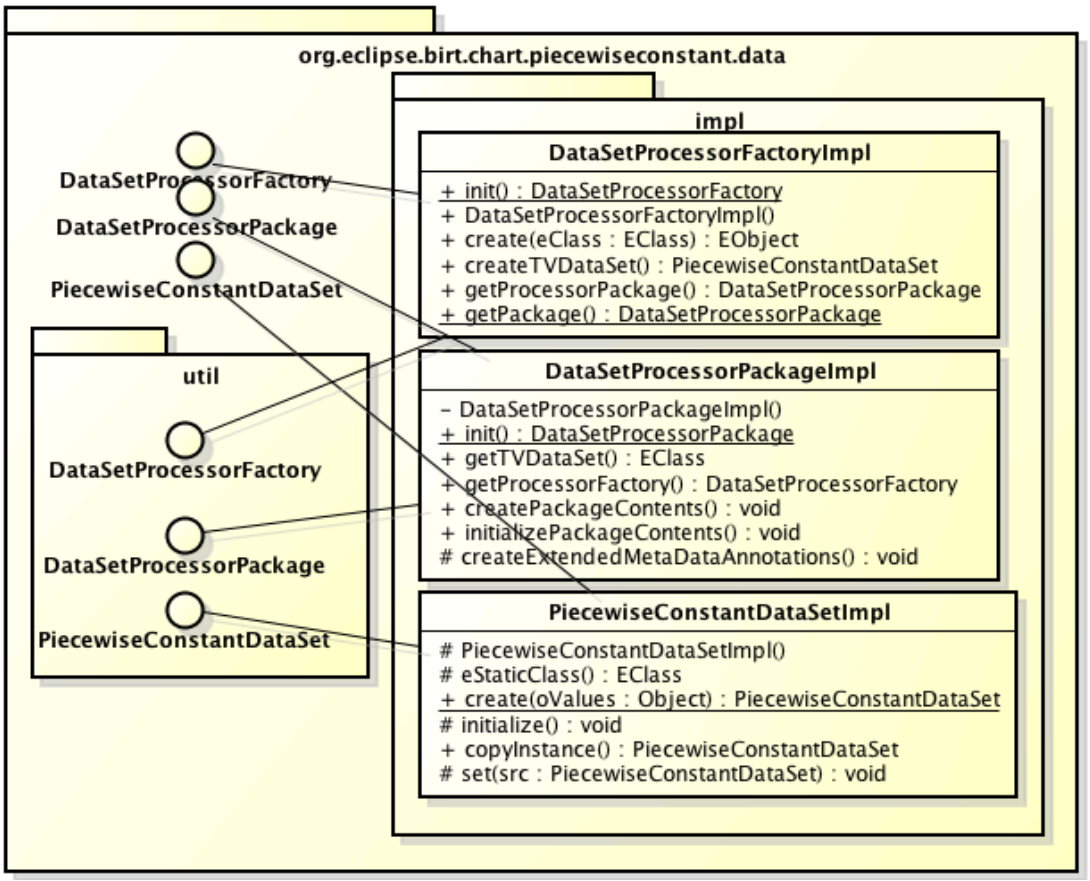

Figure 11. Class Diagram for Classes in org.eclipse.birt.chart.piecewiseconstant.data

After generating the model and data packages, we created the runtime and design time extensions for the org.eclipse.birt.chart.timeview.piecewiseconstant plug-in.

3.3.3 Runtime Extensions

Piecewise constant plot supports its own rendering mechanism and data set for orthogonal axis in order to render piecewise constant segments, to visualize infinity, to render zero-time advance plots along with the main plot, and to support dynamic plotting by scrolling. We made use of the following runtime extension points to create piecewise constant plot.

1) The extension point org.eclipse.emf.ecore.generated_package was used to register a generated Ecore package against a namespace URI (Uniform Resource Identifier) in EMF's global package registry, EPackage.Registry.INSTANCE. When a model instance is loaded from its serialized form, this registry will be consulted in order to obtain the correct metamodel to be instantiated. The package attribute of this extension point contains three properties. URI that uniquely identifies an Ecore package, class attribute, which mentions the fully qualified Java class name of the generated package, and genModel attribute that mentions the absolute or plug-in relative URI of the generator model resource for the org.eclipse.birt.chart.piecewiseconstant.model package.

2) The extension point org.eclipse.emf.ecore.extension_parser was used to define the resource factory that handles a specific file (or URI) extension. The parser element has two attributes. Type attribute denotes a file extension that identifies the filenames and URIs to be handled by a resource factory. Class attribute represents the fully qualified name of the Java class org.eclipse.birt.chart.piecewiseconstant. util.PieceWiseConstantResourceFactoryImpl implementing org. eclipse.emf.ecore. resource.Resource.Factory.

3) The org.eclipse.birt.chart.engine.charttypes extension point was used to load the custom model for a new chart type. The chart engine framework uses the charttypes extension point to load the custom model. Piecewise constant plug-in implements this extension point with the PiecewiseConstantModelLoader class located in the org.eclipse.birt.chart.piecewiseconstant.genmodel package (see Figure 7). This class has only one method getChartTypePackage(). This method creates an instance of org.eclipse.birt.chart.piecewiseconstant.model.PieceWiseConstantPackage, which has a reference to org.eclipse.birt.chart.piecewiseconstant.model.PieceWiseConstant.
4) The org.eclipse.birt.chart.engine.datasetprocessors extension point was used to process each row of data to form a data set for the PiecewiseConstant series. This extension point is used only if the chart requires a unique data-point type. This extension point has two attributes; processor – represents the fully qualified class name for DataSetProcessor, and series – represents the org.eclipse.birt.chart. piecewiseconstant.model.impl. As PiecewiseConstant chart needed to support dynamic scrolling, the chart had to remember the last scrolled value of the PiecewiseConstant series (Orthogonal value) to maintain the continuity of the segments. So, each data-point of the PiecewiseConstant series represents four values, label – name of the segment or PiecewiseConstant series labels, level – Y-Axis value or the height of the segment, previous label and previous level – represents the previous label and previous value of height of the segment respectively.

The package org.eclipse.birt.chart.piecewiseconstant.dataprocessor (see Figure 7) is associated with this data point and has two classes—PiecewiseConstantEntry and PiecewiseConstantDataSetProcessor. PiecewiseConstantEntry class was created (see Figure 11) to define attributes such as level, label, prevLabel, and prevLevel that are

required to plot the PiecewiseConstant segments. PiecewiseConstantDataSetProcessor class was created to populate a unique set of values for the series type using a query result set. The methods in PiecewiseConstantDataSetProcessor class are explained below (see Figure 12):

- populate(): This method supports setting up a processor class to handle row navigation and to populate values in PiecewiseConstantDataSet. If the plug-in reads the values to be plotted from an external file like .csv, we need to mention only the label values and the corresponding time values in the file. The logic in this method takes care of assigning the level (height of the segment), previous level, and label attributes of the PiecewiseConstantEntry object and populates the PiecewiseConstantDataSet. If the labels have numerical value, the same value is used for level attribute. In case of labels with non-numerical values, numerical mapping is done for the level. A hash map is maintained with the label as the key and the corresponding numerical mapping as the value. The height of the segment starts from one and is incremented by one for every new label that is not present in the hash map. The value for the positive and negative infinity band is calculated based on the percentage mentioned by the user and is mapped to the same value, if the label contains positive or negative infinity.
- validatePiecewiseConstantEntryData(): This method is used to validate the values in the data set.
- getMinimum(): This method gets the minimum value from the PiecewiseConstant data set.

- getMaximum() - This method gets the maximum value from the PiecewiseConstantDataSet.

5) As the datapoint (PiecewiseConstantEntry) in the plug-in has more than one value, the org.eclipse.birt.chart.engine.datapointdefinitions extension was used. This extension has two attributes, series – defines org.eclipse.birt.chart.piecewiseconstant.model.impl, and definition – defines the class org.eclipse.birt.chart.piecewiseconstant.datapointdefinition.PiecewiseConstantDataPointDefinition that implements org.eclipse.birt.chart.datafeed.IDataPointDefinition. The PiecewiseConstantDataPointDefinition class is responsible for defining the data types of a datapoint entry and its display text and it has the following methods (see Figure 12)

- getDataPointTypes(): This method returns the datapoint types for the PiecewiseConstant series. In our case, the datapoint types are Text, Numerical, Text, and Numerical for Label, Level, prevLabel, and prevLevel respectively.
- getDisplayText(): This method returns the DisplayText for the datapoint types. It returns Label value as the Display Text.
- getCompatibleDataType(): This method returns the data type for the corresponding data point type.

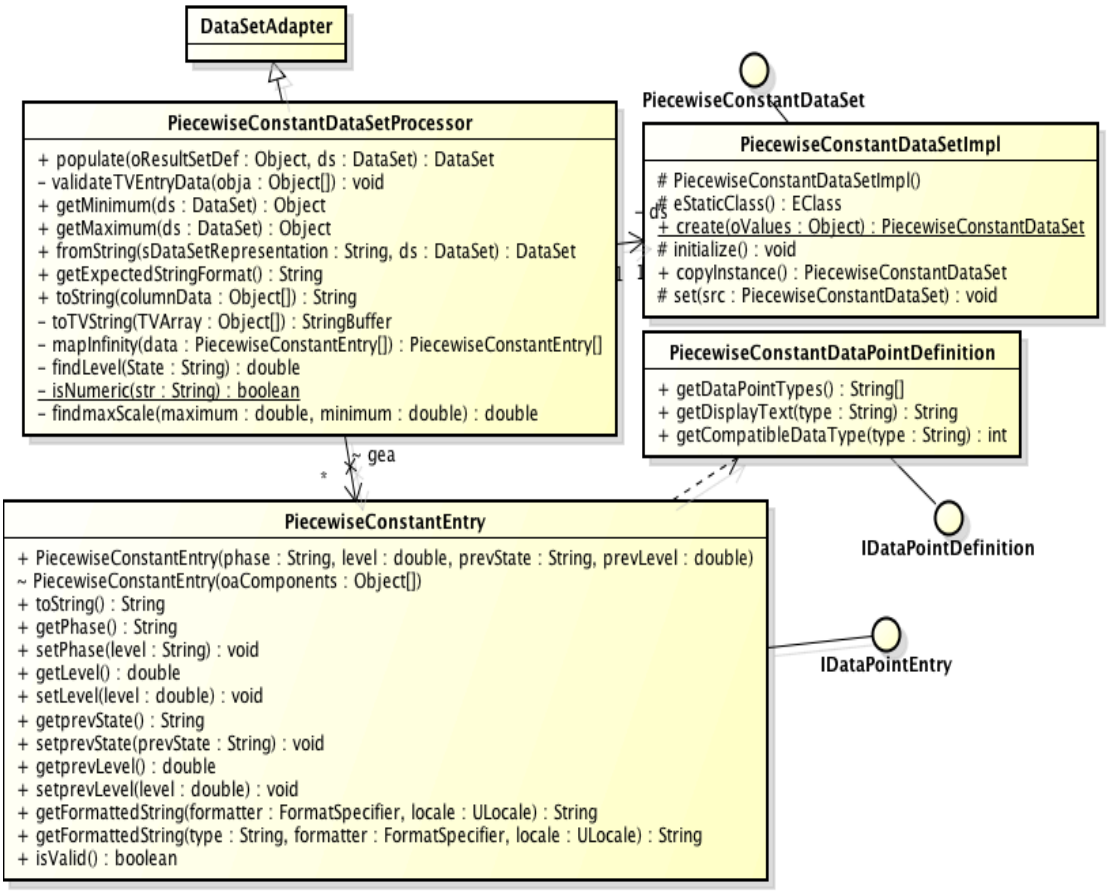

Figure 12. Class Diagram for the Classes Involved in datasetprocessor and datapointdefinition Extension Point

6) The piecewise constant plug-in needs its own logic to draw the piecewise constant segments, to render infinity band and the zero-time advance plots. So, The extension point org.eclipse.birt.chart.engine.modelrenderers was used to render the PiecewiseConstant series (Y-Axis dataset) and to focus on the actual mechanics of drawing the series. The piecewise constant plug-in has the package org.eclipse.birt.chart.piecewiseconstant.render, which is used by the above extension point and it has the PiecewiseConstantRenderer class. This class is extended from the abstract class AxesRenderer, which implements the ISeriesRenderer interface,

providing a set of convenience methods for use by a model renderer (see Figure 13). The following methods were implemented in the PiecewiseConstantRenderer class.

- Compute( ): Called by the chart engine framework in the chart engine build method just before returning the GenerateChartState object and used for rendering the chart. This method gives the series renderer the opportunity to do pre-processing on the data to be rendered.
- renderSeries(): Called by the chart engine framework for each series when rendering the series graphics for the series type. The main logic for rendering the PiecewiseConstant segments is explained in Section 3.4.

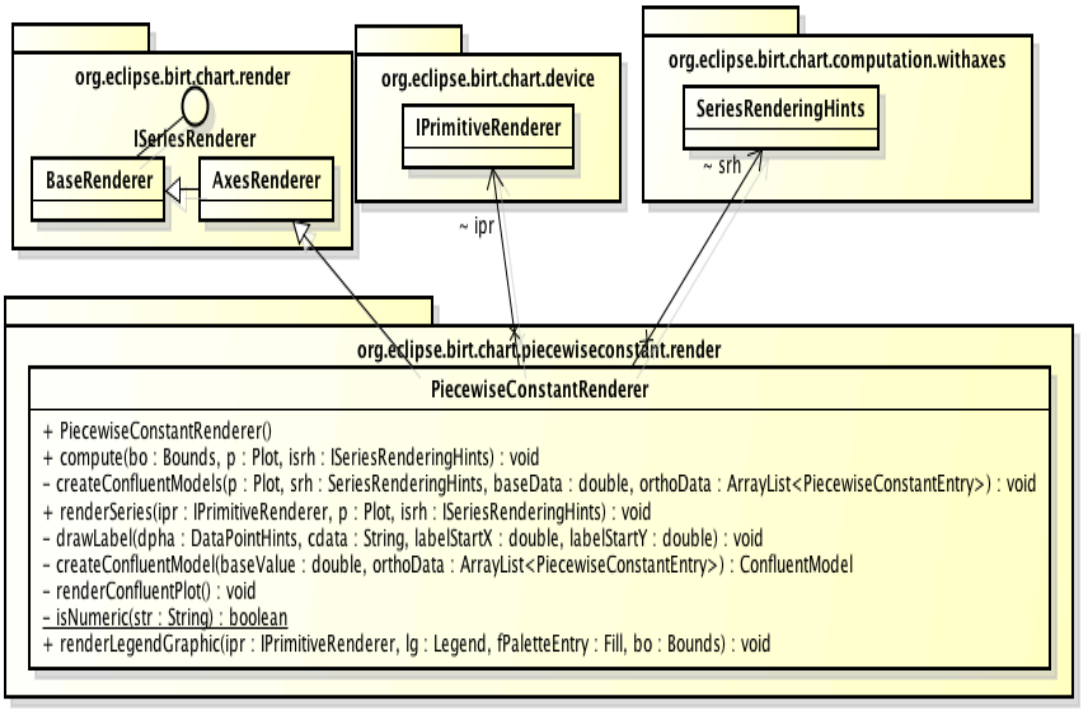

Figure 13. PiecewiseConstantRenderer Class and Its Relationship with Other BIRT classes

3.3.4 Design Time Extensions

As we were needed to register the piecewise constant plot with the chart wizard and provide user customization in addition to the capabilities provided by BIRT, we used the following design time extensions.

We needed to register the PiecewiseConstant chart as a new chart type. So, we used the extension point org.eclipse.birt.chart.ui.type, which is involved with the org.eclipse.birt.chart.piecewiseconstant.ui.type package. It has two attributes, ClassDefinition – mentioning the fully qualified class name for PiecewiseConstantChart that provides UI details about this chart type, and Name – to uniquely identify the chart defined by this extension. To use this extension point, we then created PieceWiseConstantChart class that implements the org.eclipse.birt.chart.ui.swt.interfaces.IChartType interface. The class diagram in Figure 13 shows the methods implemented in the PieceWiseConstantChart class and some of the important ones are explained below.

- getName(): Returns the name of the Chart.
- getDisplayName(): Returns the name to display in the chart builder wizard.
- getImage(): Returns the icon image to display in the chart builder for the plot.
- getChartSubtypes(): Returns a collection of chart subtypes that contains an image and name of each subtype that the plot supports. In our case, as there are no sub plots for PieceWiseConstant, the main plot is considered a sub plot.
- getModel(): Returns the model for the current plot under development. This method is called, when the user selects a different chart type, to

handle converting between chart types. For example, when the user selects a pie chart in the first tab, then selects a piecewise constant chart, this method is called to convert from a chart without axes to a chart with axes.

- isDimensionSupported(): Determines if a chart type supports a specific dimension. Piecewise constant plot supports only two-dimensional charts.
- getSeries(): This method creates an initial series type for piecewise constant plot.

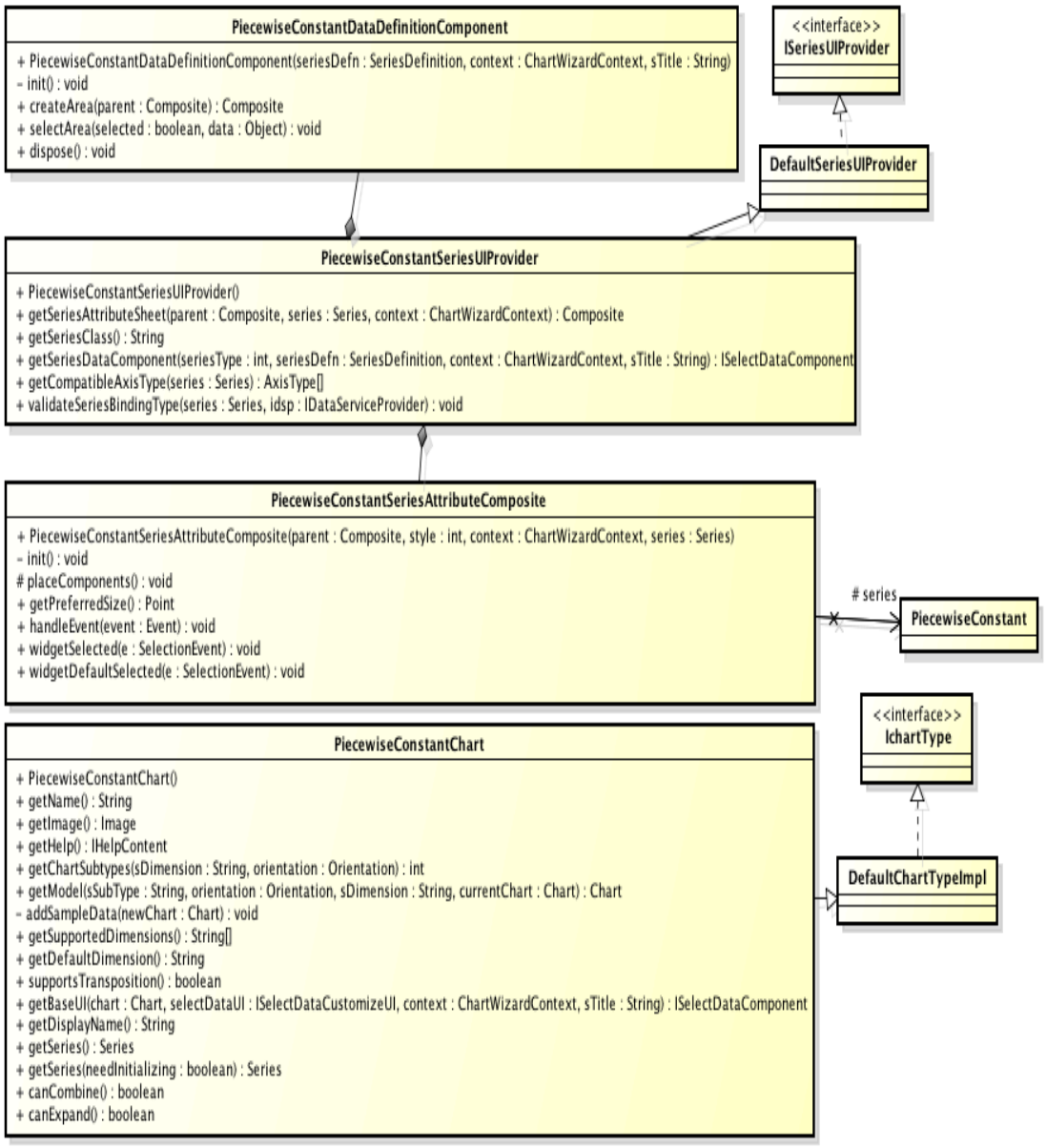

Figure 14. Class Diagram for Classes in Design Time Extensions.

The extension point org.eclipse.birt.chart.ui.seriescomposites is used to construct custom user interface elements to provide user customization of PiecewiseConstant series. The series composite element for this extension point has two attributes, seriesType and seriesUIProvider. The seriesType attribute specifies the fully qualified class name of the Series implementation class, org.eclipse.birt.chart.piecewiseconstant.model.impl.PieceWiseConstantImpl for which

this extension provides UI components. The seriesUIProvider attribute specifies the PieceWiseConstantSeriesUIProvider class that implements the ISeriesUIProvider interface. This extension point is implemented in the second and third tabs of the chart builder. Figure 14 shows the important methods involved in the class PieceWiseConstantSeriesUIProvider:

- getSeriesAttributeSheet(): This method constructs the composite used to manipulate the piecewise constant plot's series attributes by returning the created object for PieceWiseConstantSeriesAttributeComposite. This composite sheet is used in the third tab of the chart builder. The PieceWiseConstantSeriesAttributeComposite class contains the logic to provide user customization such as enabling and disabling infinity band, setting up percentage of space taken up by the infinity band, series line attributes for main and confluent plots, enabling and disabling confluent plots, etc.
- getSeriesDataComponent(): Builds elements that appear in chart builder by returning user interface items for the value series and the optional value series grouping.
- getSeriesClass(): Returns the series class for which the ISeriesUIProvider is implemented.
- validateSeriesBindingType(): Verifies that the series has the proper data type.
- getCompatibleAxisType(): Determines if a specific axis type is supported for a value series. In our case, the value series supports the linear axis type.

3.4 RENDERING PIECEWISE CONSTANT PLOTS

The rendering of axes, grid lines, and the title for the piecewise constant plot is handled by BIRT. This section deals with the PiecewiseConstantRenderer class, a runtime extension that deals with rendering the PiecewiseConstant series.

BIRT API does the mapping of the data values to the pixel values and is passed as a parameter to renderSeries method. Using these pixel values, the segments are rendered. As BIRT does not support plotting the concept of stacked linear and superdense time segments (see Figure 6) out of the box, logic should be provided for separating the superdense time values from main data set and adjusting the pixel values provided by the BIRT for zero-time advance plots to be rendered below the main plot.

3.4.1 Pre-computation of Dataset and Creation of Zero-time Advance Models

The compute method is called by BIRT before rendering the series and is used to do pre-computation with the available base and orthogonal series data. The base data set with indexed zero-time advance value and its corresponding orthogonal data set is formed from the main data sets for each zero-time advance plot to be rendered (see Algorithm 3.4.3.1). An instance of zero-time advance model is created for each sub plot with the data sets and the mapped pixel values as attributes. As BIRT does not provide the mapping of data set values to the pixel values for the zero-time advance plots, the following logic was used to calculate the pixel value relative to the main plot.

Let the number of zero-time advance plots to be rendered be *n.* One fourth of the total plotting area is allocated for these sub plots, which is divided amongst them

and are rendered horizontally. For each plot, base (x-axis) and orthogonal (y-axis) data points need to be calculated for rendering the series.

The *baseDataPoints* gives the X coordinate pixel value and the following attributes must be calculated. *plotStart* and *plotEnd* represents the start and end co-ordinates of the main plot. This is used to calculate the *xStepSize*, which represents the pixels between major ticks in the X-Axis and remains the same across all sub plots. *totalDataSize* represents the sum of the size of all data sets used to render the sub plots, which is helpful for calculating the *xStepSize*. *xStart and xEnd* represents the coordinate of the start and end of the X-Axis for each sub plot. The formula is as follows:

baseDataPoints[i] = xStart+(xStepSize(i))* , where

i = the count of the data value whose pixel mapping is calculated,

xStepSize = |(*plotStart*)-(*plotEnd*)| / (*totalDataSize* +*n*) ,

The *orthogonalDataPoints* gives the y co-ordinate pixel value and the following attributes must be calculated. *posPixels* and *negPixels* represent the pixels needed for the positive and negative Y-Axis respectively. *yValue* represents the Y-Axis data value to be plotted. *maxValue and minValue* represent the maximum and minimum orthogonal value amongst all the data sets for the zero-time advance sub plot. The pixel values are scaled based on the *maxValue* and *minValue* to make sure all sub plots have the same Y-Axis scale.

$$orthogonalDataPoints[i]= \begin{cases} plotEnd - (negativePixels + |yValue/maxValue| * posPixels)), & if\ yValue > 0 \\ (plotEnd - negativePixels) + (|yValue/minValue| * negPixels)), & if\ yValue < 0 \end{cases}$$

3.4.2 Render Series

The render series method is called by BIRT after rending the axes for the main plot. The ISeriesRenderingHints is passed as a parameter to renderSeries method, which contains the mapped data set values to pixel values computed by BIRT. Below are the steps followed for rendering piecewise constant series:

- Rendering the positive and negative infinity band if enabled.
- Rendering the series in the main plot using the pixel values provided by the attribute DataPointHints of ISeriesRenderingHints instance.
- Rendering the zero-time advance plot, if any, using attributes of the ConfluentModel instance stored in the confluentModels ArrayList.

3.4.3 Algorithms

Figure 15 explains the pre-computation of the values in the base and orthogonal data set of the main plot to form separate data sets for the zero-time advance sub plots. Figure 16 explains the creation of ConfluentModel instance and assigning attribute values using the above formula in section 3.4.1. Figure 17 explains rendering of PiecewiseConstant series, which includes logic for rendering series in the main plot, infinity bands, and the sub plots.

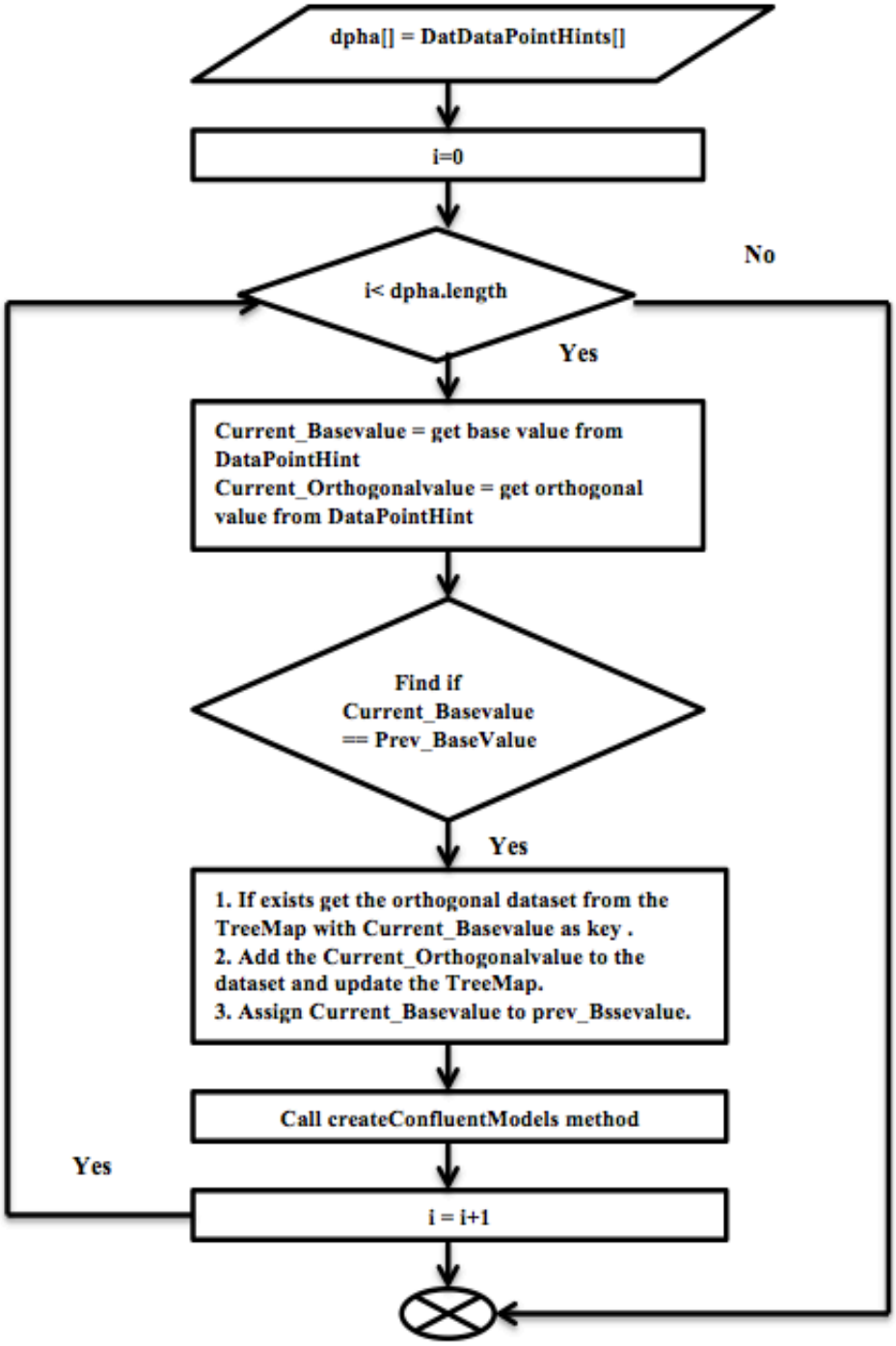

Figure 15. Pre-computation of Main Plots' Data Sets – The Algorithm

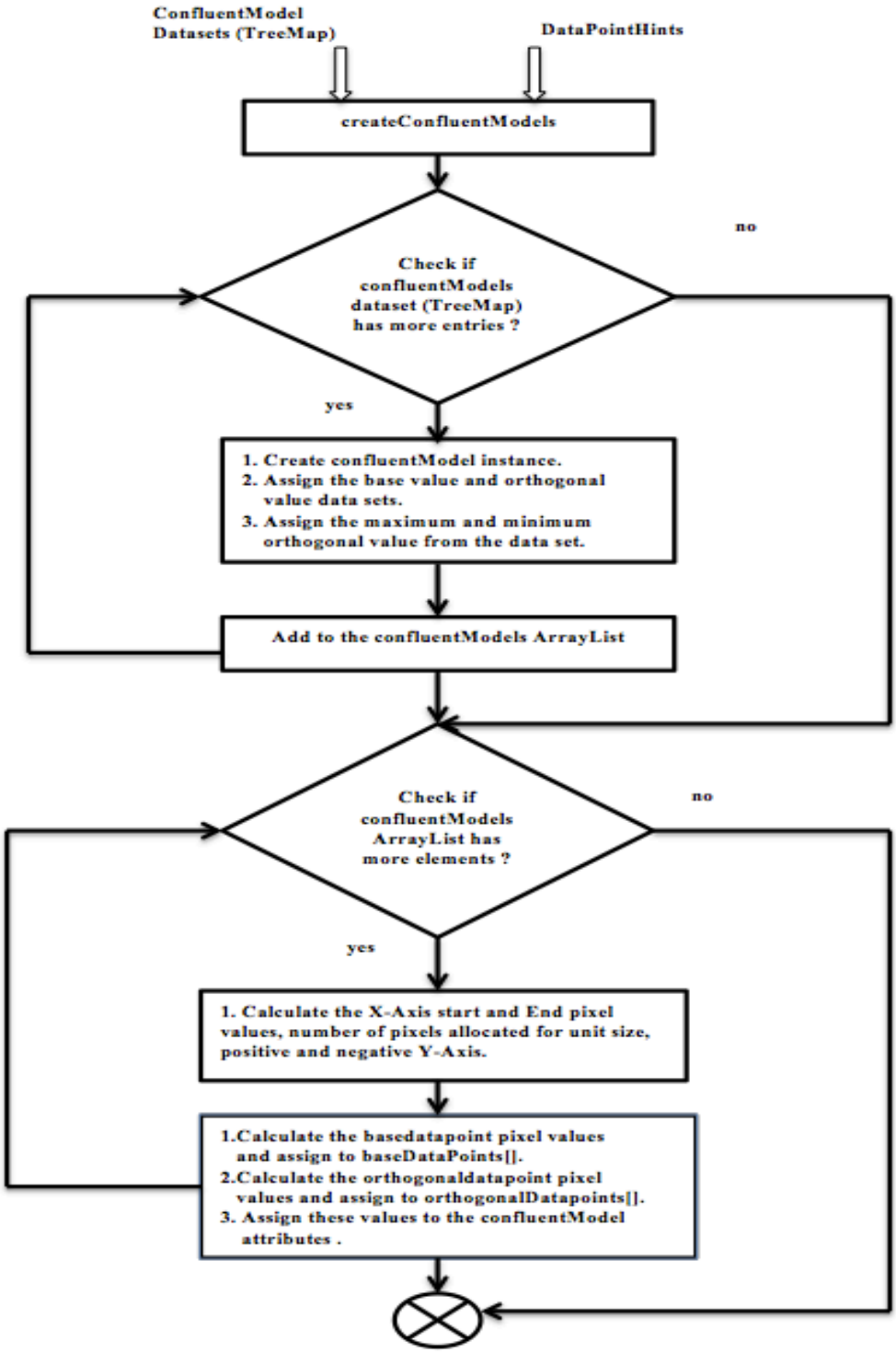

Figure 16. Creation of confluentModel Instances and Attributes – The Algorithm

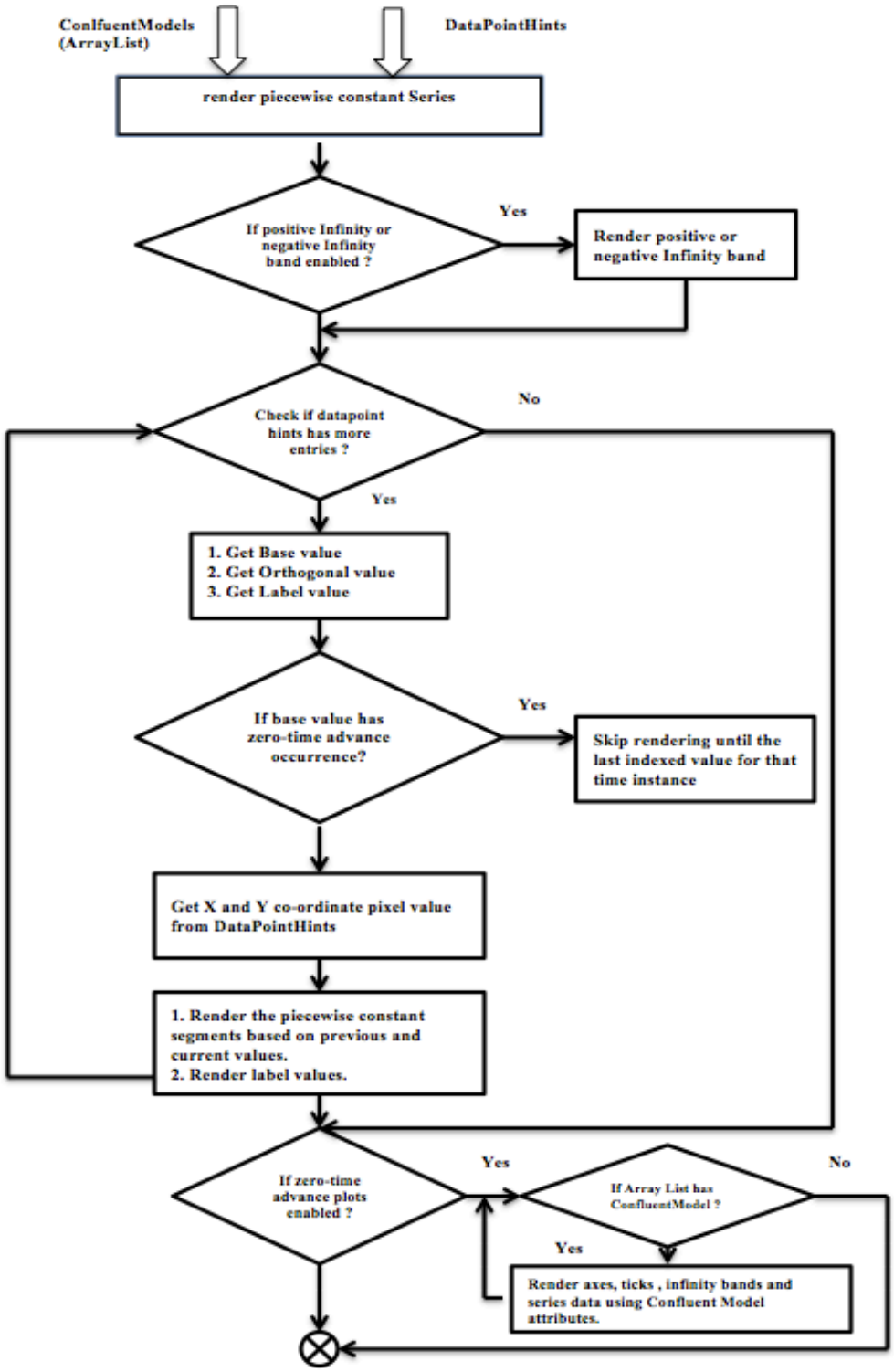

Figure 17. Rendering PiecewiseConstant Series – The Algorithm

3.5 Requirements for org.eclipse.birt.chart.timeview.event Plug-in

An event chart consists of event segments that represent a sequence of events ordered in time. It is mainly used to plot the input and output event from the discrete

event models. Based on the properties of the event chart defined in (Sarjoughian, Sundaramoorth, 2015), the following requirements are defined for the event plug-in.

3.5.1 Plotting Requirements

Axes and data values: The event chart renders the values that are sequentially ordered with time. So, the X-Axis of the chart represents time, which is a positive real number, and the ordering is monotonic and displayed in linear space. It should also support rendering of stacked linear and superdense time segments plotted as main plot and zero-time advance plots respectively. The values of the Y-Axis are integers and can vary from positive infinity to negative infinity. Since the Y-Axis is also linear, a numerical mapping should be introduced if string values need to be rendered as events. For e.g., the output of the discrete event model such as {job1, job2, job 3} is mapped to some numerical constant {2,2,2}. The segments are labeled, especially if it is a string.

The plotting should support dynamic scrolling, visualization of positive, negative infinity, and zero-time advance plots similar to PiecewiseConstant plots.

3.5.2 User Customization of the Plot

In addition to the customization features provided by BIRT, the plug-in should provide the necessary customization features that are special to the event chart. The user should be able to customize the main and zero-time advance plots' series attributes like line properties, series labels, arrow color, etc. The user should be allowed to enable or disable the viewing of zero-time advance plots, the positive and negative infinity bands, and also adjust the width of the infinity band by specifying the desired percentage.

3.6 Design Specifications for Event Plot

As the major requirements and all the extensions of piecewise constant and event plug-in remain the same, our initial design decision was to have a single plug-in. The piecewise constant and event plots were considered as sub plots in the main plug-in. But considering the differences between these two plots and the future contributions, the decision to have separate plug-ins for piecewise constant and event plot was made.

3.6.1 Extension of Schema Files for Event Constant Chart Model

Event chart has its unique properties to enable and disable infinity, zero-time advance plots and data set. So, we defined our own chart model using XSD files Event.xsd: New event series by extending the base-level series from the component.xsd file was created. This includes the additional elements (see Figure 18) such as LineAttributes, showPositiveInfinityBand, showNegativeInfinityBand, positiveInfinityPercent, negativeInfinityPercent, showConfluentPlot, confluentAxesLineAttributes, ArrowColor, and confluentAxesLabel, confluentAxesLabelPosition.

The org.eclipse.birt.chart.event.model, org.eclipse.birt.chart.event.model.impl, org.eclipse.birt.chart.event.model.util packages were generated using the Event.xsd file and the methods such as getDisplayName, Initialize, create, copyInstance were written manually as described in PiecewiseConstantImpl.

After generating the model and data packages, we created the runtime and design time extensions for org.eclipse.birt.chart.timeview.event plug-in.

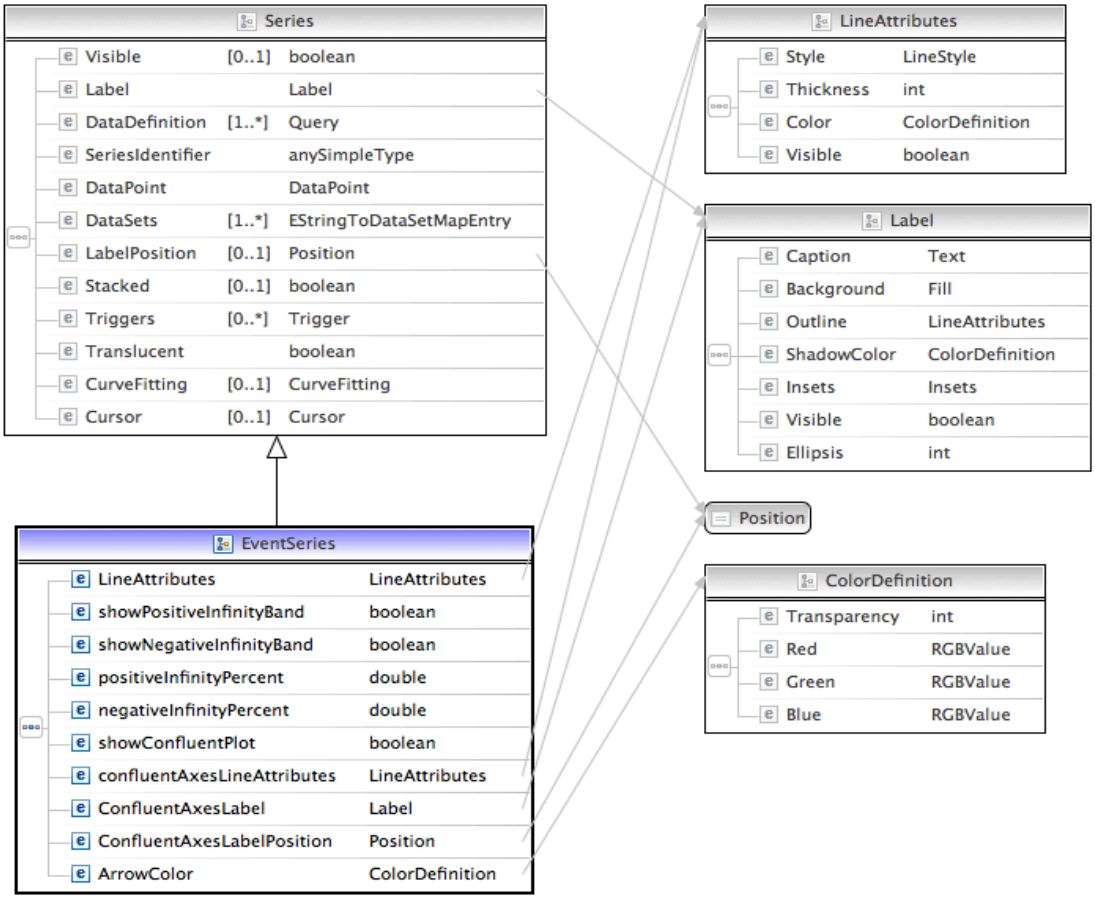

Figure 18. Design View for Event.xsd File

3.6.2 Runtime Extensions for Event Plot

The extension point org.eclipse.emf.ecore.generated_package, org.eclipse.emf.ecore.extension_parser, org.eclipse.birt.chart.engine.charttypes, and the logic associated with the classes of these extensions remain the same with respect to the piecewise constant plug-in.

The package org.eclipse.birt.chart.event.dataprocessor is associated with org.eclipse.birt.chart.engine.datasetprocessors extension point and has two classes—EventEntry and EventDataSetProcessor. The major difference is that EventEntry

requires only two attributes, Level and Label. Because of the nature of the event plot segment, prevLevel and prevLabel attributes used in PiecewiseConstantEntry does not serve the same purpose here. The EventDataSetProcessor has the same methods as in PiecewiseConstantDataSetProcessor and the logic of these methods remain the same.

As the datapoint (EventEntry) in our plug-in has more than one value, the org.eclipse.birt.chart.engine.datapointdefinitions extension was used. The package org.eclipse.birt.chart.event.datapointdefinition is associated with it. The EventDataPointDefinition class is responsible for defining the data types of a datapoint entry and display text. The main difference lays in the getDataPointTypes method, which returns the datapoint types Text, Numerical for Label and Level respectively.

The extension point org.eclipse.birt.chart.engine.modelrenderers was used to render the Event series (Y-Axis dataset). This extension is associated with the package org.eclipse.birt.chart.event.render. The implementation of all the methods in the EventRenderer class remains the same with respect to the piecewiseConstantRenderer class except for renderSeries. This method focuses on the logic of rendering the event segments. The pre-computation of the data sets and logic used for rendering zero-time advance plots remains the same.

3.6.3 Design Time Extensions for Event Plots

As we needed to register the event plot with the chart wizard and provide user customization, in addition to the capabilities provided by BIRT, we used the org.eclipse.birt.chart.ui.type and org.eclipse.birt.chart.ui.seriescomposites extensions. These extensions are involved with packages org.eclipse.birt.chart.event.ui.type and

org.eclipse.birt.chart.event.ui.series respectively. The implementation of all the methods involved in these packages remains the same as that of the piecewise constant plug-in except for the EventSeriesAttributeComposite class. This class contains the logic to provide UI for user customization and is based on the series attributes defined in the org.eclipse.birt.chart.event.model.EventSeries.

Because of the above differences, we created a separate event plug-in for a cleaner design and implementation. This will also make the future contribution to plug-ins a lot easier.

CHAPTER 4

INTEGRATION OF TIME-SERIES PLUG-INS WITH DEVS-SUITE

This chapter explains about the improvisation of the time view component in DEVS-Suite by integrating the created time-series plug-ins to render the plots and to provide advance customization features to configure the chart properties through the chart wizard.

4.1 TIMEVIEW COMPONENT IN DEVS-SUITE 2.1.0

The time view component in DEVS-Suite 2.1.0 supports dynamic plotting and basic customization capabilities of piecewise constant and event charts written from scratch using Java Swing API. DEVS-Suite follows MVFC architecture (Sarjoughian, 2003), where the package view.timeview in the view layer is associated with generating time-series plots.

Understanding the existing design of the time view component is crucial for the integration of the developed time-series plug-ins (Kim, 2009). The time view component is capable of tracking phase, sigma, TL, TN, and input and output ports for each component of the model in a separate scroll pane. The TimeView class is responsible for creating this scroll component and integrates into the Tracking Environment. The TimeGraph class is responsible for rendering all the plots for the tracked component in the scroll pane. This class supports dynamic scrolling of the plot by maintaining three queues—next, current and previous—that store the events. When the data generated by the simulator is available in the façade layer, an instance of event class is created and stored in the next queue. The background thread in the TimeView class keeps updating the queues every few milliseconds and calls the

drawGraphData method in TimeGraph class to render plots. The TimeGraphAttributeSet class has the attributes necessary for the plots such as axes, labels, etc. Figure 19 shows the higher-level overview of time view component in DEVS-Suite 2.1 through the package diagram.

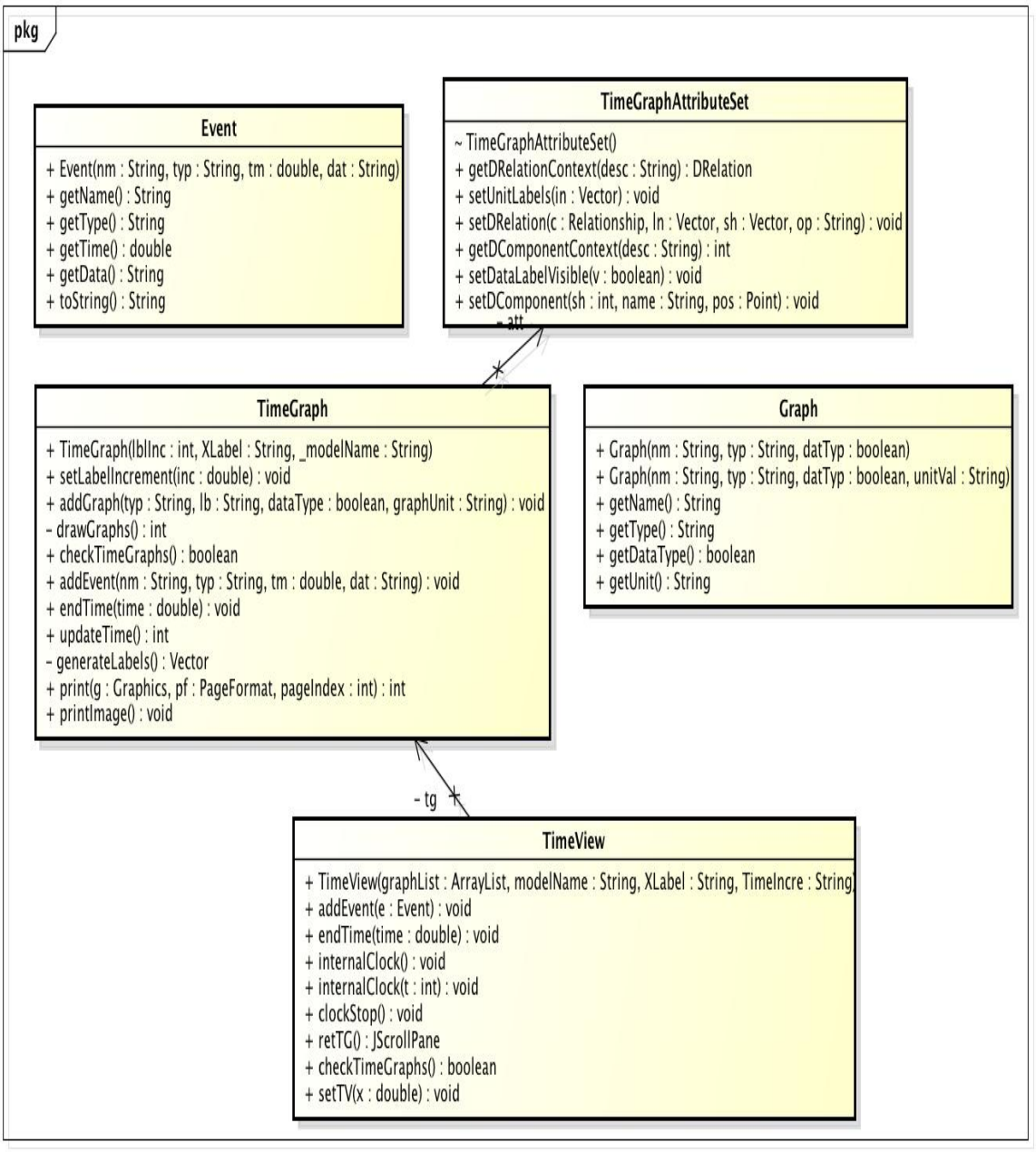

Figure 19. Package Diagram for Time View Component in DEVS-Suite

4.2 TIMEVIEW COMPONENT IN DEVS-SUITE 3.0

The time view component in DEVS-Suite 3.0 is improvised by using the extended BIRT time-series plug-ins to render and provide advanced customization features. Understanding the BIRT Chart Engine API is important before integrating the time-series plug-ins with any Java application, which can be found in (BIRT ChartEngine API, 2013). The rest of this section explains the design decisions taken for integration and the workflow involved.

4.2.1 DESIGN DECISIONS FOR INTEGRATING TIME-SERIES PLUG-INS WITH DEVS-SUITE

As seen in Chapter 3, the developed piecewise constant and event plug-ins provide getter and setter methods for the chart series attributes, rendering logic, and the customization features for the time-series plots. The key concept of the integration lies in the following steps:

- Creating the chart objects to be rendered using the BIRT application programming interface (API).
- Converting data simulated to a data set understandable by the time-series plug-ins.
- Updating the created chart object with the data set and customized attributes values.
- Rendering this chart object using BIRT API to a swing device.

For integration, numerous changes were made to the time view component in DEVS-Suite 2.1 except for its interaction with the façade layer via the controller to get the simulated data and updating the queues for scrolling. Currently, DEVS-suite

supports only time-series plots. But decisions were made accounting for the future support of other kinds of charts and the easy integration of them. Figure 20 shows the high-level architecture overview of DEVS-Suite integrated with extended BIRT plug-ins and the classes involved in time view component.

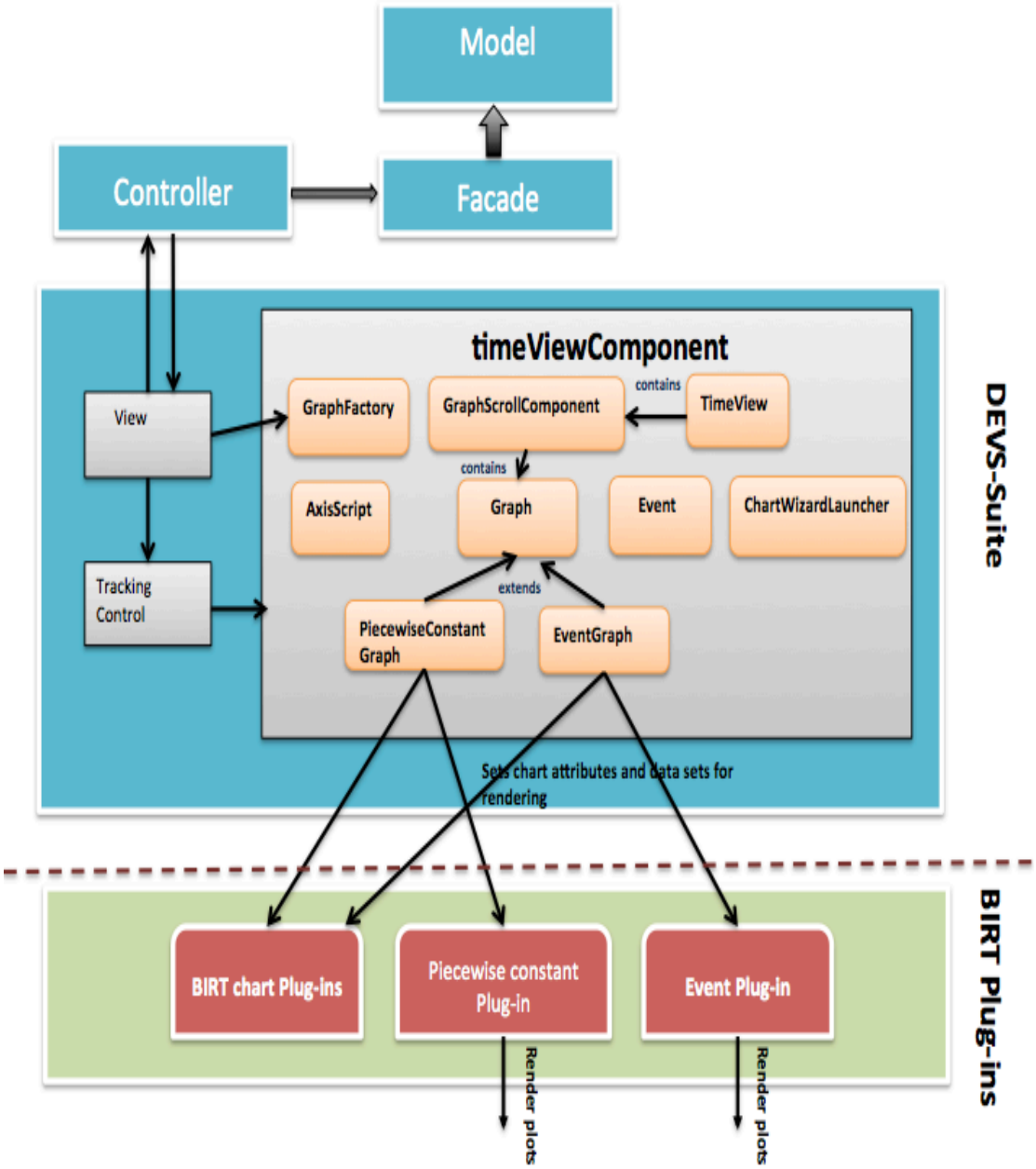

Figure 20. Integration of Time-Series Plug-ins with DEVS-Suite

The functionalities of the classes in time view component are explained as follows:

- The Graph class, which is the super class, provides the abstract functionalities for the concrete graph classes to extend and implement. The abstract methods' definition includes filling the data set for the chart, updating the data set for every simulation cycle, and setting the axes attributes such as scale and unit name. The Graph class also extends JPanel to render the chart to a swing device, which is accomplished by overriding the paint method. The paint method includes the logic of passing the created chart object to the BIRT chart engine to be generated and rendered.
- The Piecewise Constant Graph and Event Graph class extends the Graph class and implements the abstract methods.
- The GraphFactory class uses Factory Pattern to get the corresponding chart object based on the type of chart to be created. This decouples the code and provides flexible implementation in case there is an addition of a new chart like bar or stack chart for visualization.
- For each tracked component, an instance of GraphScrollComponent class is created. It has all the Graph objects to be rendered on the scroll pane and the logic to update the properties and data sets for each graph object.
- The TimeView class will create the TimeView component and integrate into the Tracking Environment. This class has a background thread, which is called every few seconds to update the time start and time end of the GraphScrollComponent objects based on which data sets of each plots are updated and re-rendered.

- The ChartWizardLauncher class has the API to invoke the chart builder API, which provides customization features that can be configured before and during plotting.
- The plots display +Infinity and –Infinity as the axes labels for the positive and negative infinity bands. By default, BIRT displays numeric values for linear axes labels. The AxisScript class has the logic for replacing the appropriate numeric value labels with +Infinity and –Infinity while rendering the axes labels. The ExternalScriptVariable serves as the helper class for the AxisScript class and the AxisScriptLoader class loads the script.

4.2.2 Workflow Illustrating the Usage of Time-Series Plug-ins to Create, Render and Customize Plots

The following sequence of diagrams give a detailed analysis on the steps involved in creating a chart object, rendering the created object, and in the customization of the chart's attributes.

4.2.3.1 Creating Chart Object

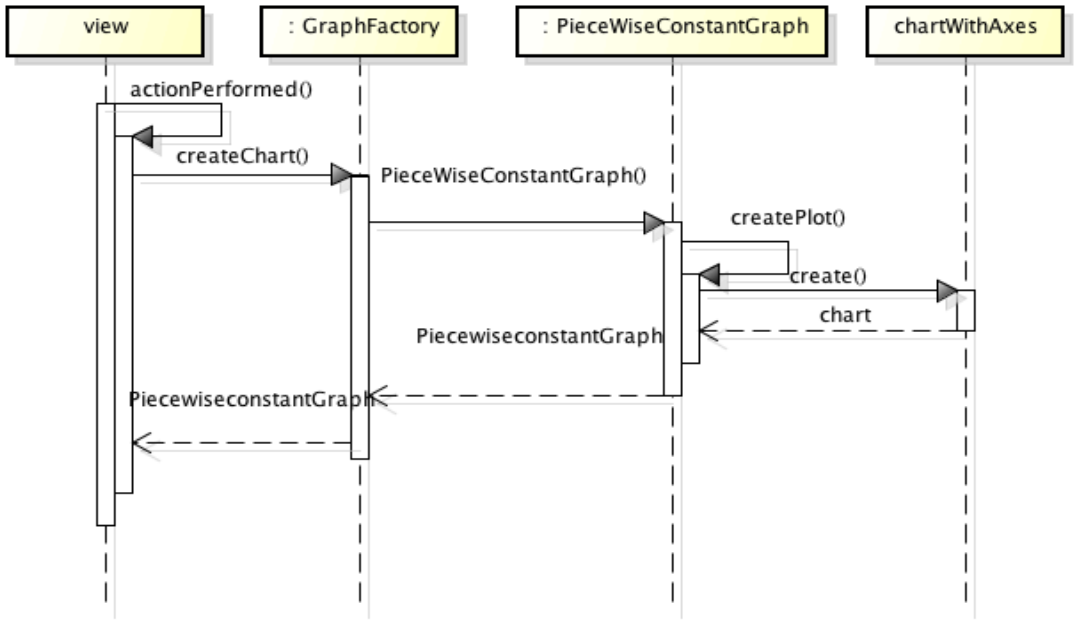

Figure 21. Sequence Diagram for Creation of Chart Objects

The sequence diagram in Figure 21 shows the control flow involved in the creation of chart objects and initialization of the basic chart properties.

The model configuration and tracking wizard has listeners to keep track of which component's plots the user wants to view. When the user clicks the track button in the wizard, actionPerformed method in View class is invoked. This calls the createChart method in the GraphFactory class and the type of chart to be created is passed as a parameter in the method. Based on the type, the respective chart objects are created.

As shown in Figure 21, the createChart method returns an instance of PiecewiseConstantGraph class. During the instance creation, the constructor of the PiecewiseConstantGraph class invokes the createPlot method, which is responsible for creating a chartWithAxes object and setting the basic chart properties using the

BIRT API. With the created PiecewiseConstantGraph object, one can update the chart's data set and properties and render the chart on a swing device.

4.2.3.2 Rendering the Chart Object

The created chart object is rendered on a Swing device, JPanel. The chart object's dataset is updated with the new data available in the façade layer and re-rendered on the JPanel. The sequence diagram in Figure 22 explains the mechanism involved in rendering the plot.

GraphScrollComponent's updateTime method is invoked every few milliseconds to update the time start and time end of the plots in GraphScollComponent. This in turn calls the drawGraphData method for each plot to be updated.

The drawGraphData method uses the chart engine and time-series plug-ins' API to update the chart's data set and render the updated chart object.

- The setAxesScale method in PieceWiseConstantGraph class calculates and sets the chart's minimum and maximum X-Axis and Y-Axis values. The calculation takes into account the infinity percent specified by the user if enabled.
- The updateDataSet method creates a new data set for the orthogonal axis with the PiecewiseConstantEntry values and a numberDataSet for the category axis with the double array representing time. The created data set is updated with the X-Axis and Y-Axis of the chart.
- Once the chart is updated with new values, the repaint method is invoked, which calls the paint method to re-render the chart. The paint method gets the generator instance from the chart engine API. Build method is invoked using

this instance, which returns the generated chart state. This method builds and computes preferred sizes of various chart components offscreen using the provided display server. The render method is invoked to display the previously built chart into the swing device.

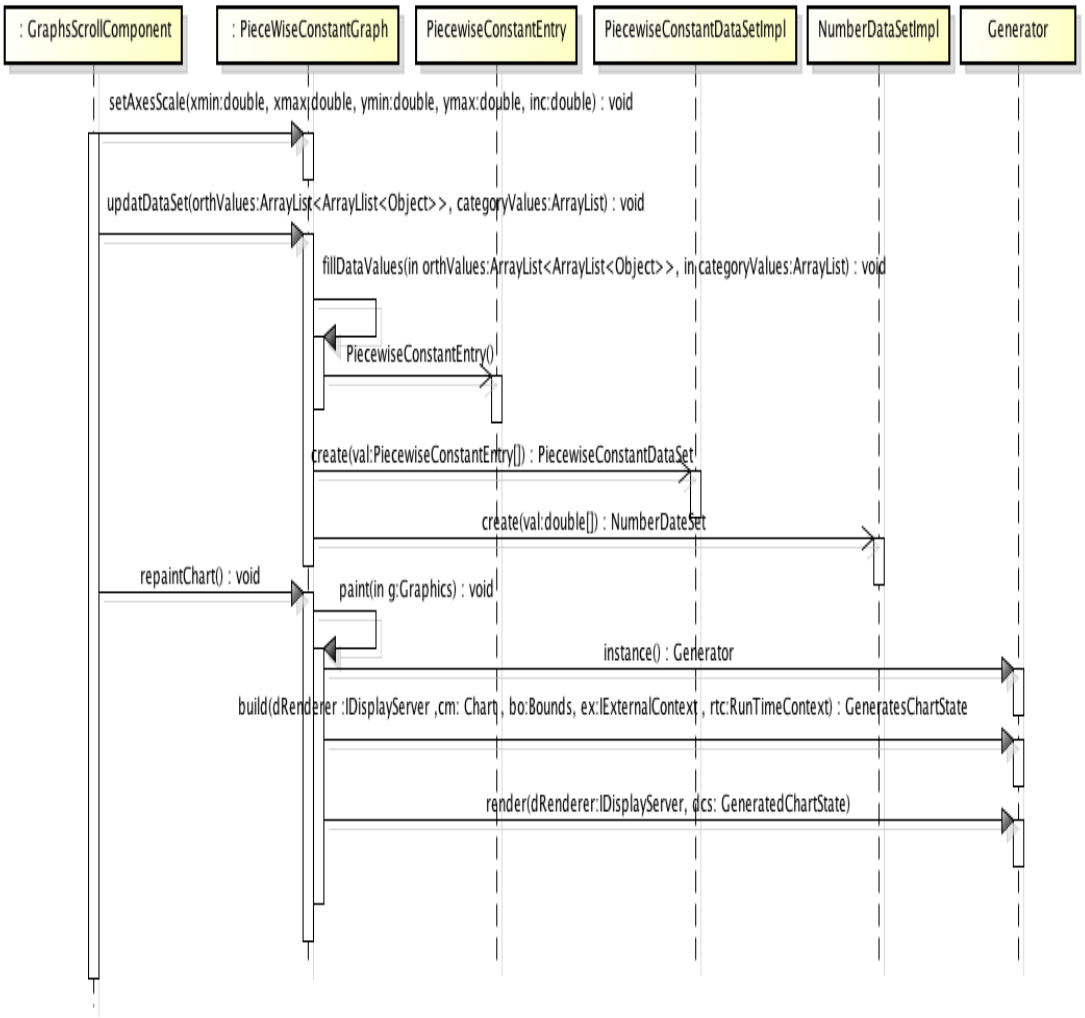

Figure 22. Sequence Diagram for Rendering Chart

4.2.3.3 Customization

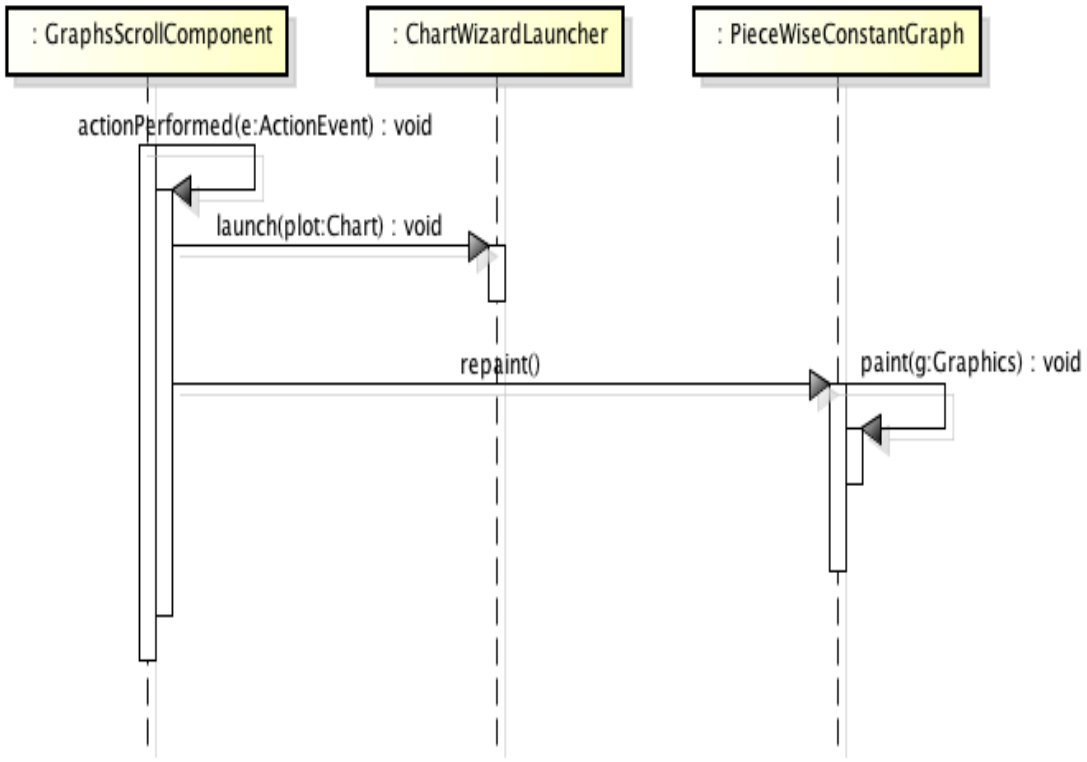

Figure 23. Sequence Diagram for Chart Customization

The user can customize the chart properties before and during the rendering of the plot. A customize button is provided in the time view scroll window. When the button is clicked, the launch method is invoked, which opens the chart wizard. The chart object to be customized is passed in the method. As described in Chapter 3, the piecewise constant and event plug-ins provides customization features for series properties like enabling and disabling the infinity band and zero-time advance plots as well as changing the series label. The repaint method is called, which in turn invokes the paint method to regenerate the chart to reflect the change in the chart's properties as shown in Figure 23.

4.3 PACKAGING DEVS-SUITE 3.0

As DEVS-Suite 3.0 uses the extended time-series BIRT plug-ins and BIRT chart API for plotting, one should make sure the necessary external dependencies are added to the class path. Detailed explanation on setting up the environment for DEVS-Suite and the process used for analyzing the external dependencies are given in Appendix B.

4.3.1 Distributing DEVS-Suite as JAR file

The final deliverable for DEVS-Suite contains the DEVS-Suite.zip of the following structure (Figure 24). The created zip archive contains the application JAR file, DEVS-Suite.jar, and the external libraries needed in the DEVS-suite_lib folder. The relative paths to these libraries are added in the class path entry of the JAR's manifest file.

```
DEVS-Suite.zip
|
|----DEVS-Suite_lib/
|    |--External jars (plug-ins: SWT, RCP, BIRT, …)
|---DEVS-Suite.jar
|   |-- Classes
|   |----MANIFEST/
|        |----MANIFEST.MF
|
```

Figure 24. Packaging of DEVS-Suite

The JAR file can be used as the following:

As Executable: The users can unpack the zip file making sure that the JAR file and libraries of the JAR files are in the same folder. Run the application JAR file. The SimpArcMod package is included in the DEVS-Suite.jar. While configuring the

DEVS-Suite, the full path to the location of the JAR file must be given in, path to packages of model classes.

As Non-Executable: In the non-executable version, the SimpArcMod package is not included in the JAR file. This is mainly used to add the JAR file as a dependency for the users to create their own models using an IDE. After creating the Java project, the user should add DEVS-Suite.jar located in the zip folder. The dependencies are automatically included in the project's class path.

4.4 FEATURE COMPARISON BETWEEN DEVS-SUITE 2.1 AND 3.0

The Figure 25 compares the features offered by the time view component in the previous and current version of DEVS-Suite.

Features	DEVS- Suite 2.1	DEVS- Suite 3.0
Model Configuration and Tracking wizard		
Basic Configuration settings	Yes	Yes
Tracked components shown in configuration wizard	No	Yes
Options to track phase, sigma, TL, TN, input and output ports	Yes	Yes
Plotting and Customization Capabilities		
Rendering piecewise constant and event plots for positive and string values	Yes	Yes
Rendering piecewise constant and event plots for negative values	No	Yes
Zero-time advance plots	No	Yes
Positive and negative infinity bands	No	Yes
Plot resizing	No	Yes
Customization wizard for advance chart properties	No	Yes
Easy integration of other charts	No	Yes

Figure 25. Feature Comparison Between DEVS-Suite 2.1 and DEVS-Suite 3.0

4.5 PERFORMANCE ANALYSIS

Once the extended BIRT time-series plug-ins were integrated with DEVS-Suite, performance analysis was done on the entire system to monitor how well the time view component performed with the integrated BIRT plug-ins. We performed two kinds of analyses.

1. The first analysis was to discover the robustness of the time-series plug-ins by increasing the load.
2. When the simulator runs at maximum speed, the majority of the rendering is done after the simulation is completed. The second analysis was done to establish the correctness of the time view component in DEVS-suite 3.0 and to determine if the BIRT plug-in contributed to an initial delay.

To do the analyses, we needed a way to measure the time taken by the program. Rather than rely on a manual stopwatch we developed a special class called Stopwatch that could be used to track the time elapsed from an arbitrary time point in the execution. We also developed a class called PerformanceMeasure, which contains the logic needed to do the analysis. Below are detailed explanations of the experimental setup, trials, observation, and inference for Analysis 1 and 2 mentioned above.

4.5.1 Load Analysis on Extended BIRT Plug-ins (Analysis 1)

The load analysis was performed to test the robustness of time-series plug-ins under two scenarios—zero-time advance plots disabled and enabled. The load on the plug-ins is increased by scaling the number of components tracked for the model EFP with 30 processors. The time taken to render is calculated by the PerformanceMeasure

class, which calculates the time from the moment the user clicks “run simulation” until the time-series plug-ins render the last event. The diagram below shows the scatter plot of the load analysis. The experimental set up and the observations are also explained.

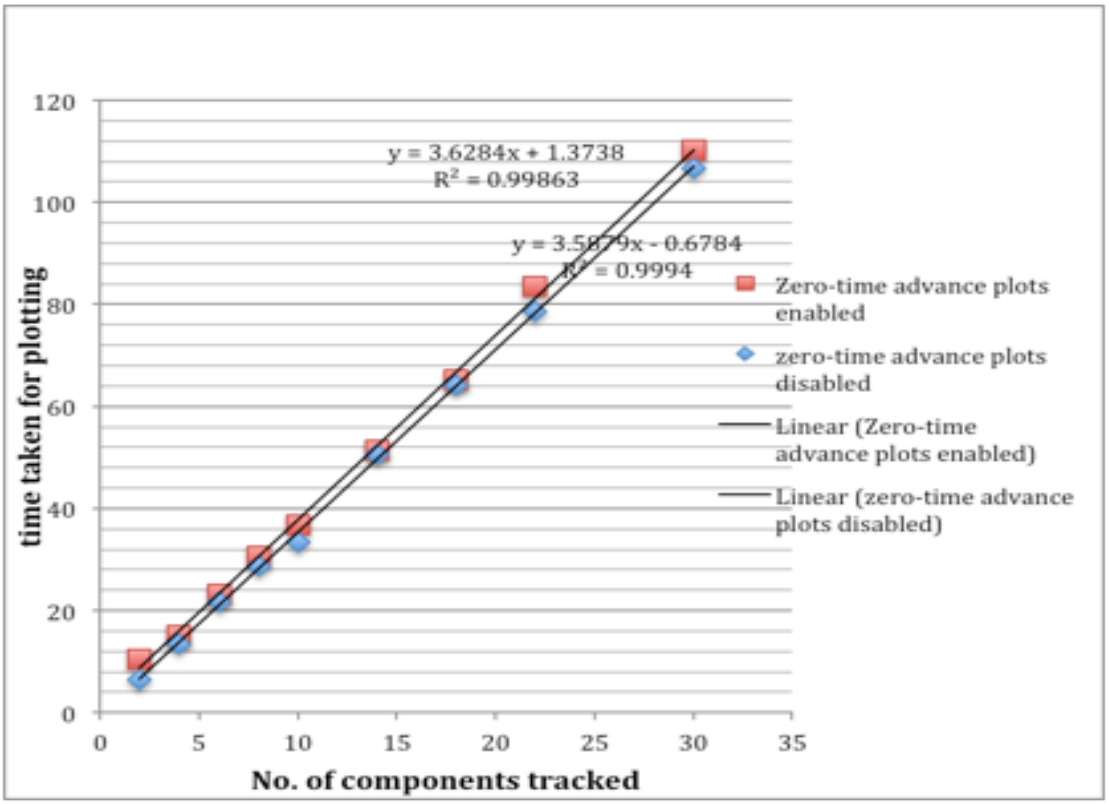

Figure 26. Load Analysis of Time-Series Plug-ins

4.5.1.1 Experimental Setup

The plots—phase, sigma, TL, TN, and output—were monitored for each component and the components tracked were scaled from 2 to 30 for the model EFP. This information is plotted along the X-Axis and the total time taken in seconds for rendering was plotted on the Y-Axis. The same experiment was conducted for two scenarios—plots with zero-time advance disabled and zero-time advance enabled. The number of jobs generated and the simulator speed are kept constant, in this case 20

jobs and 1.0E-4 respectively. Also, the SimView was disabled and the TimeView was tracked in separate windows. Each trial was run 3 times and the mean was taken.

During these trials, the computer was allowed to function normally, operating all programs and background processes that it would during normal use. The DEVS-Suite was executed from the JAR file and DEVS-suite was restarted for each trial. However, there was no user interaction while the experiments were in progress. What is important is how long the time view takes to complete the plotting when the components are scaled, which demonstrates the stability of the extended plug-ins.

4.5.1.2 Observation

- One can observe from the given plot (Figure 26) that there is a linear increase in time taken for plotting with increase in components tracked for both series.
- The linear equations represent the relation between the components tracked and the time taken for both series. The equation for plotting with the zero-time advance plot disabled and enabled is y= 3.5879x – 0.76784 and y= 3.6284x+ 1.3738 respectively, where y represents the time taken for plotting and x represents the components tracked. This linear increase in time and the regression factor being close to 1 represents the robustness of the extended plug-ins.
- The difference in time taken for plotting is minimal between enabling and disabling of zero-time advance plots, which explains that there is less overhead in plotting zero-time advanced plots and the difference almost remains constant when the components are scaled. This also demonstrates the robustness of the time-series plug-ins.

4.5.2 Analysis of Correctness of Time View Component Under Different Simulation Speeds

This analysis was done to check the correctness of the improvised time view component. To perform the above analysis, the growth rate of the queues—next, current, and previous—in the time view component and the jobs rendered was measured every few seconds. The updateInfo and the updateJobsrendered methods were added to the class PerfromanceMeasure to measure the growth rate of the queues and the correctness of the time view component. The updateJobsrendered method was called to update the number of jobs plotted after the BIRT plug-in renders the event for each plot. The updateInfo method updates the next, current, and previous queues' size and the jobs rendered by BIRT for every few seconds that pass. This gives a good picture of the growth rate of the queues and the jobs rendered under various simulation speeds.

4.5.2.1 Experimental Setup

The experiment was done with a gpt model by tracking the generator component and rendering the phase plot. The number of jobs generated was kept constant for all the trials. The monitored time in seconds was plotted on the X-Axis and the number of jobs in the queues and the jobs rendered at the monitored time are plotted as different series along the Y-Axis. The experiment was performed under various simulation speeds.

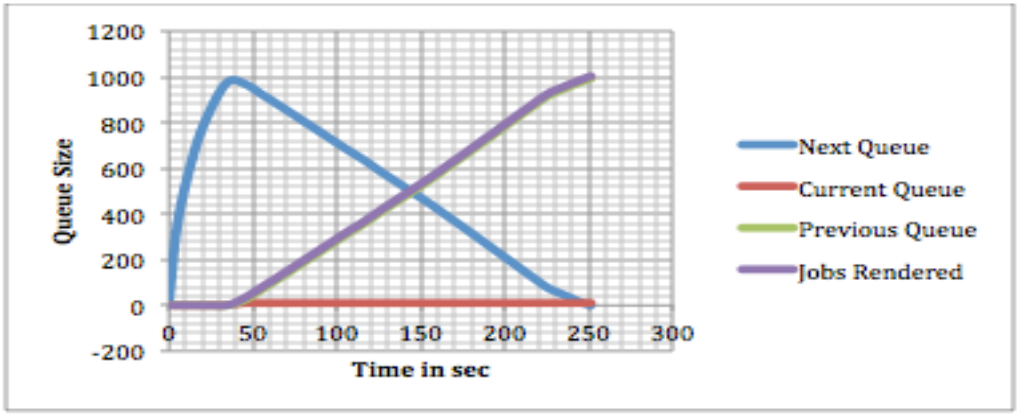

Figure 27. Correctness Analysis of Time View Component at Speed 1.0E-4

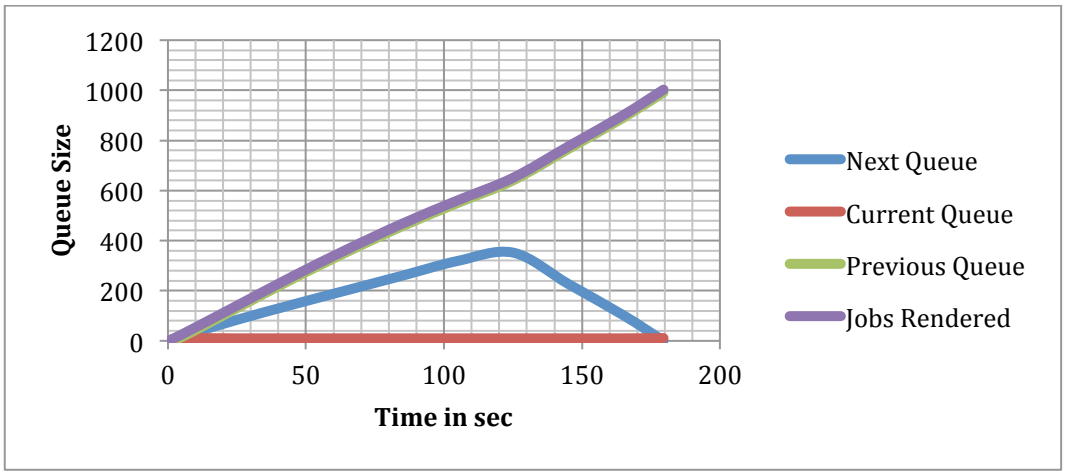

Figure 28. Correctness Analysis of Time View Component at Speed 0.01

4.5.2.1 Observation

From the Figures 27 and 28, one can observe that the sum of jobs in the next, current, and previous queues and the number of jobs rendered are equal for the last time instance monitored. This establishes the correctness of the time view component, meaning all the simulated events are rendered.

CHAPTER 5

DEMONSTRATION

The capabilities of the developed time-series plug-ins—piecewise constant and event plug-ins—are demonstrated by adding them to the BIRT RCP application and integrating with the DEVS-Suite application.

5.1 DEMONSTRATION OF TIME-SERIES PLUG-INS WITH BIRT RCP

For this demonstration, the environment required was setup and the extended time-series plug-ins were deployed as explained in Appendix A. This demonstration explains creating a BIRT report using the extended time-series plug-ins.

Step 1: After the initial set up, switch to the report design perspective in Eclipse. Create a new BIRT project by clicking File→ New→Project. A wizard appears as shown in Figure 29. Select Business Intelligence and Reporting Tools →Report Project.

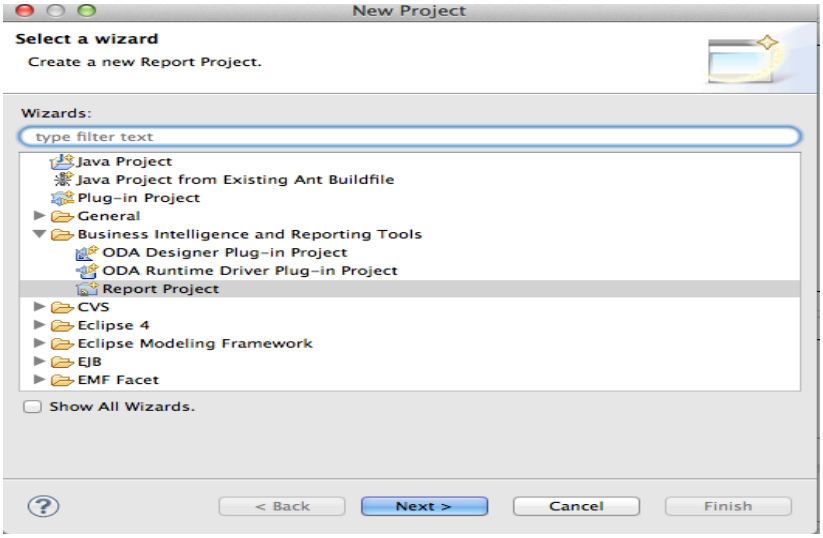

Figure 29. Initial Report Project Configuration Wizard

Step 2:

Click Next. Give the Project Name and the location where the Project should be created. Click Finish.

Step 3:

Create a blank report by clicking File→New→Report. The wizard appears. Choose the project folder and give the report name in the style of sample.rptdesign. Click Next. In the next screen, choose Blank Report and click Finish.

Step 4:

A blank report will be created. Switch back to the Palette, select a chart, drag and drop it onto your report. The extended chart types—piecewise constant and event charts—appear with the other BIRT charts. For the demonstration, we choose a piecewise constant chart as shown in Figure 30.

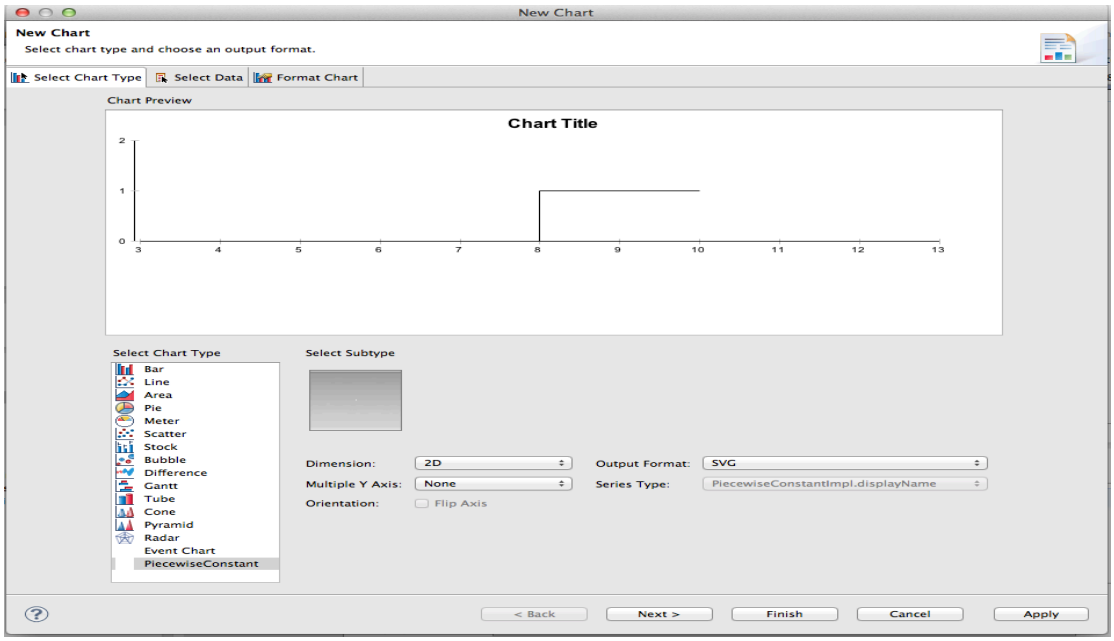

Figure 30. Chart Wizard for Properties Configuration

Step 5:

Click Next and choose the data source and your data set. Click “use data from” and choose “flat file” as the data source. A dialog box appears. Select the folder where the CSV files are located. The first row and the second row in the CSV files represent the column names and their data types respectively. Make sure “use second line as data type indicator” is checked as shown in Figure 31.

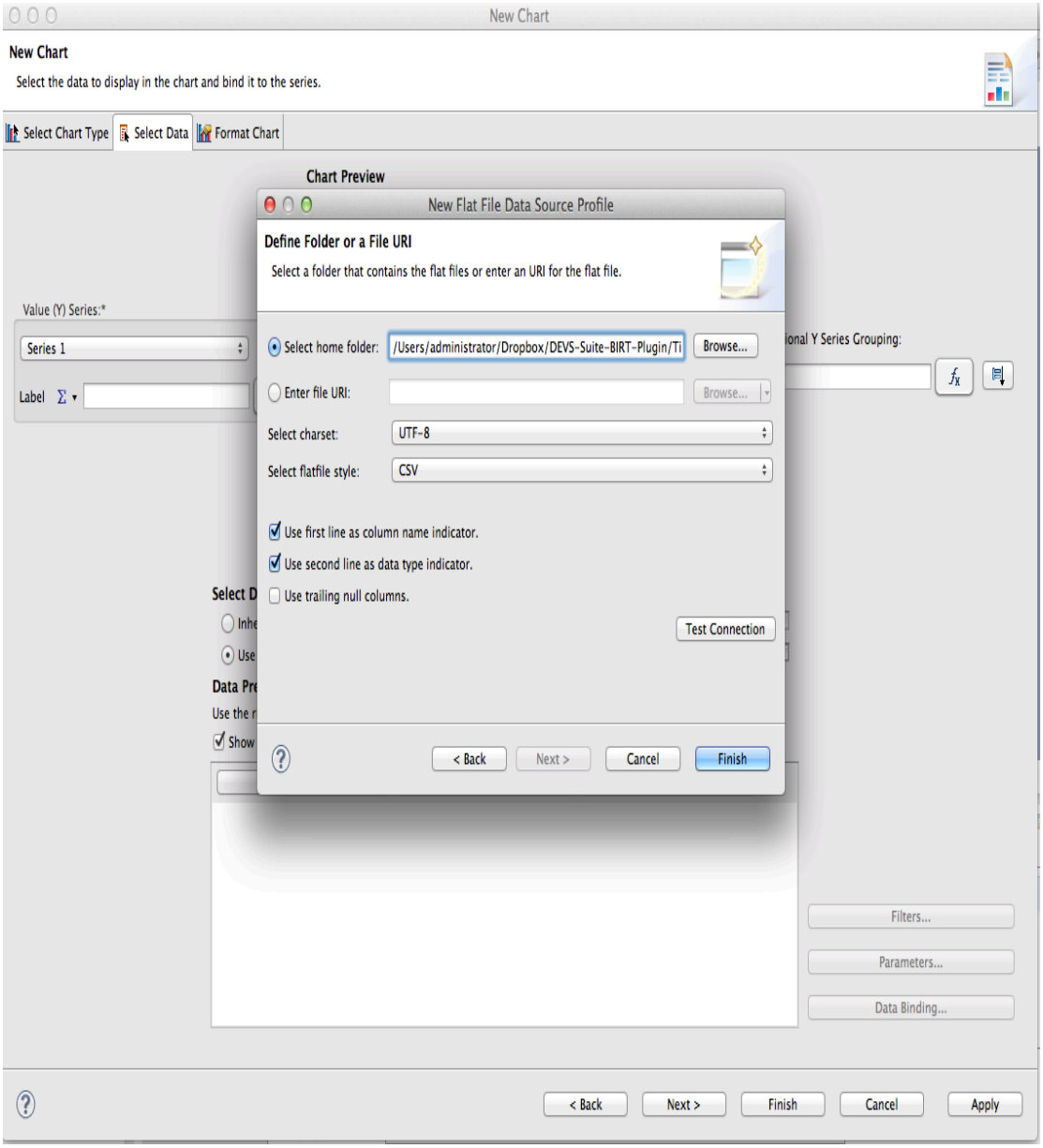

Figure 31. Configuration of Data Source for the Chart

Step 6:

Click Finish. Another screen appears to configure the new data set. Give the data set a new name and click Next. Choose the CSV file from the combo box. The columns in the file will be displayed. Select the columns by clicking on the arrow as shown in Figure 32 and click Finish.

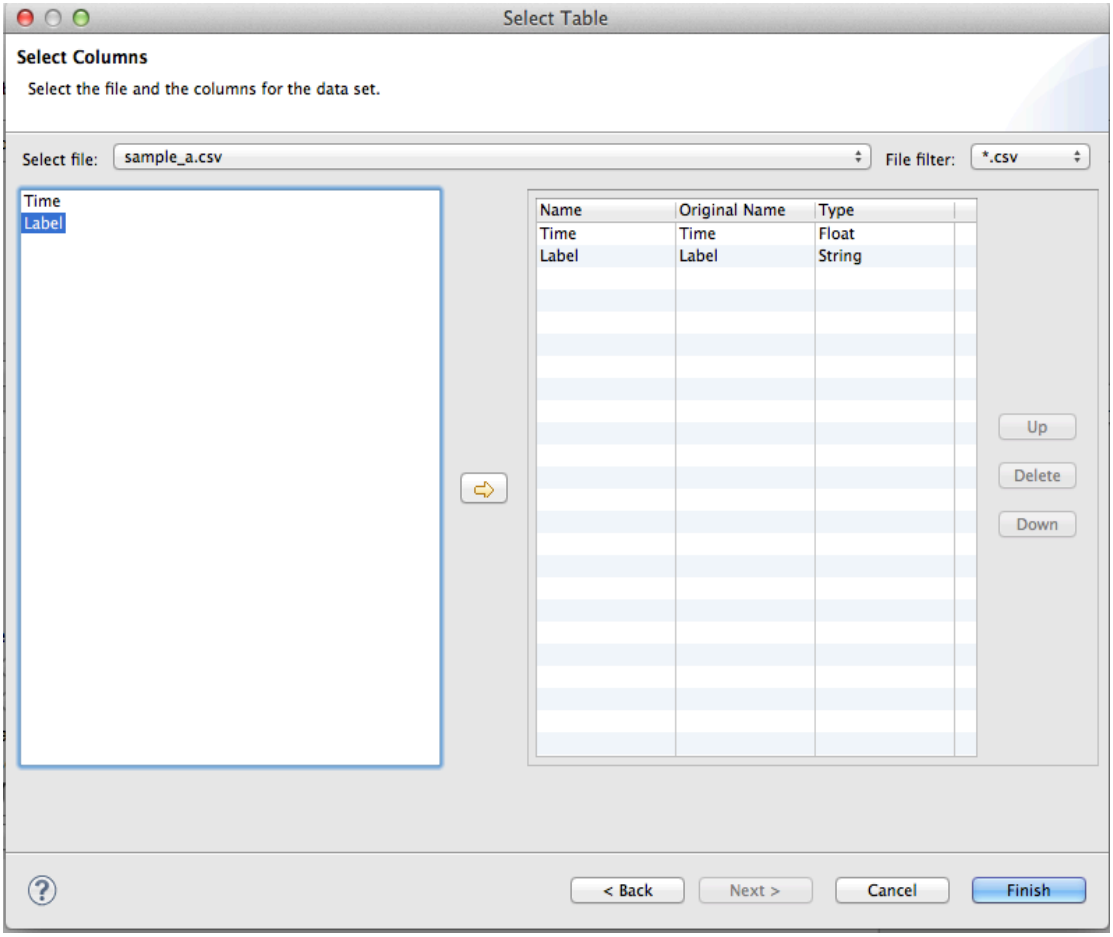

Figure 32. Selection of File and Columns for the Data Set

Step 7:

The columns appear in the data preview. Assign the Time column for X series and Label column for Y series as shown in Figure 33. If the repeated values in the category series do not need to be grouped, disable "grouping enabled" by clicking the second button next to Category (X) Series. If the labels have string values, numerical mapping is done by the plug-in and the plot is rendered.

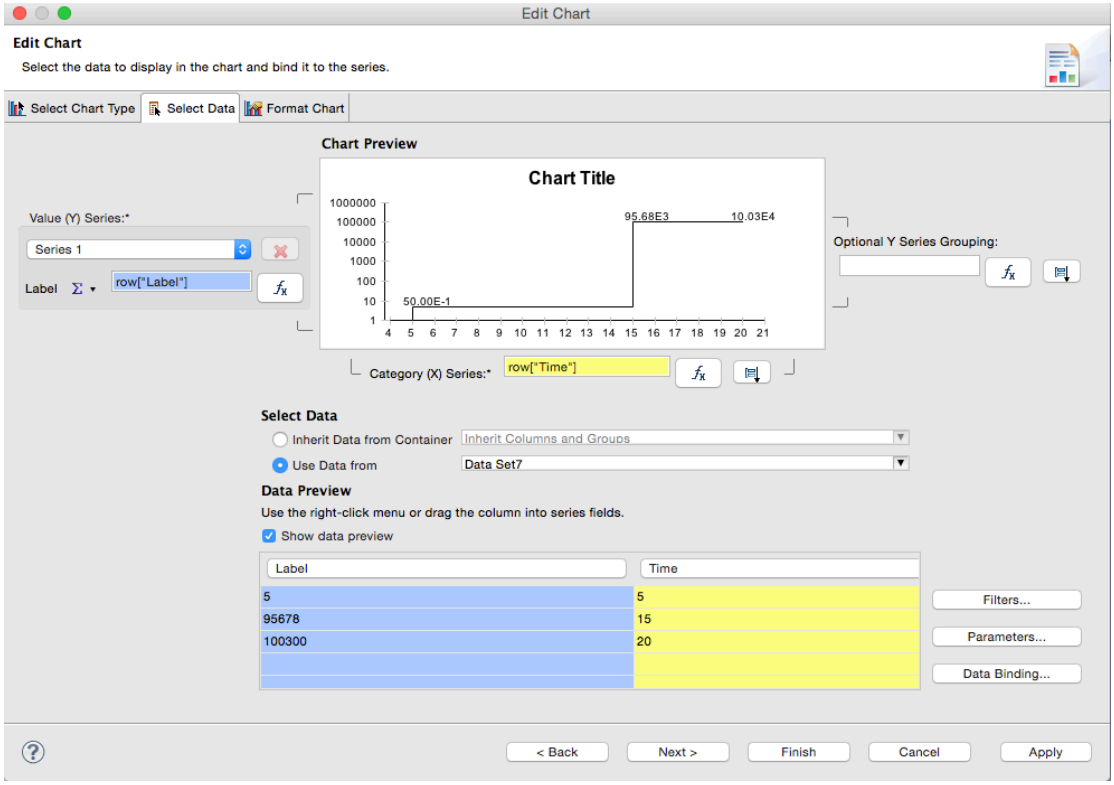

Figure 33. Setting the X and Y-Axes Data Sets

Step 8:

Click Next to go to the Format chart tab. Under the Series tab, choose Value (Y) Series. It has the newly added customization properties for this plug-in, like configuring the infinity bands, confluent plots, series line attributes, and labels as shown in Figure 34. For plots with repeated values for category series, instead of grouping, they could be viewed as separate sub-plots below the main plot. This concept is useful in simulation tools. Option ShowConfluent provides this feature. Other properties related to axis, title of the chart, plot, and legend are modified here.

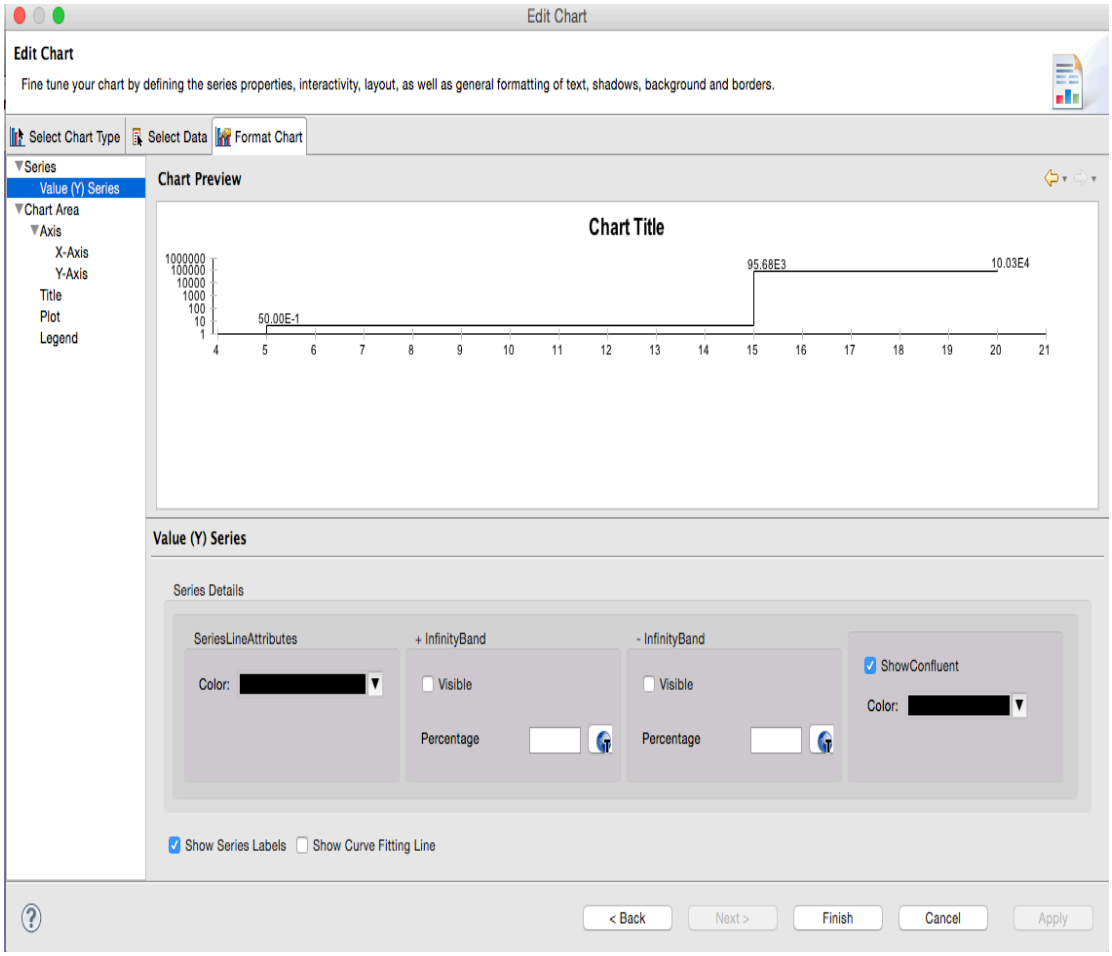

Figure 34. Configuring Chart Series Properties

Step 9:

In the report design page, click Preview to view the chart. This shows the piecewise constant chat with the super dense time trajectories rendered in logarithmic scale.

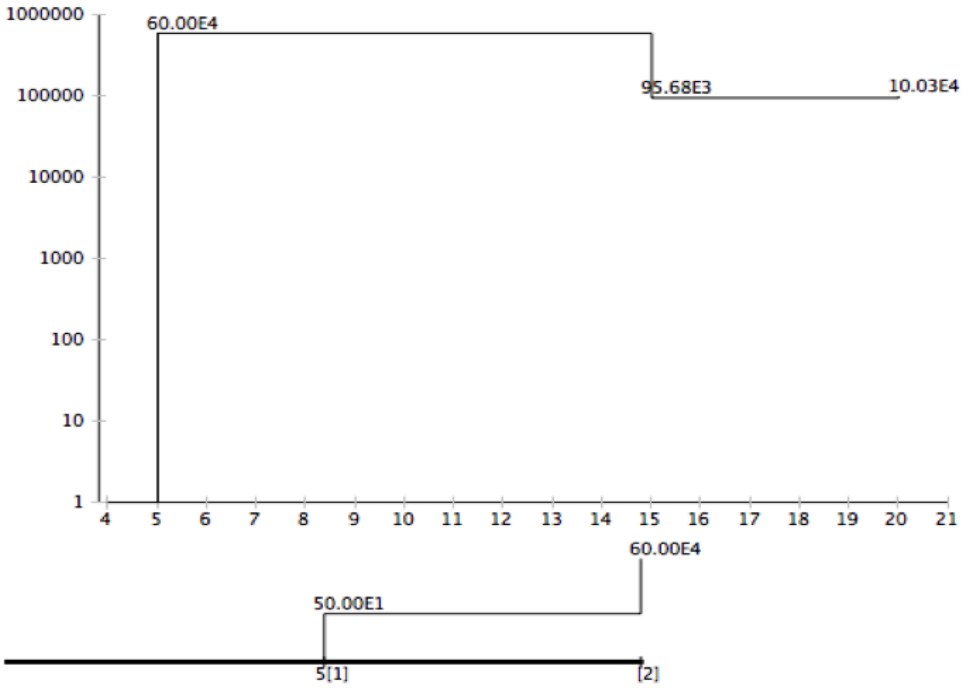

Figure 35. Report View of the Piecewise Constant Chart

5.2 DEMONSTRATION OF TIME-SERIES PLUG-INS WITH DEVS-SUITE

The developed piecewise constant and event plug-ins were integrated with DEVS-Suite as explained in Chapter 4. This demonstration shows the simulation of a basic EFP model and tracks the simulation data using the BIRT integrated time view component. Apart from the integration of time-series plug-ins, a new configuration wizard is introduced, which has steps to configure and choose the model, select the components to be tracked, and set the initial chart properties.

Step 1: Loading of Model

Select the package and the model (EFP) to be simulated from the drop down menu. If the package is not available, it can be configured by giving full path to the

package classes, source files, and the package name by clicking the configure button (see Figure 36). Choose the tracking option, if the simulation data needs to be tracked by plots and/or log files. Click Next.

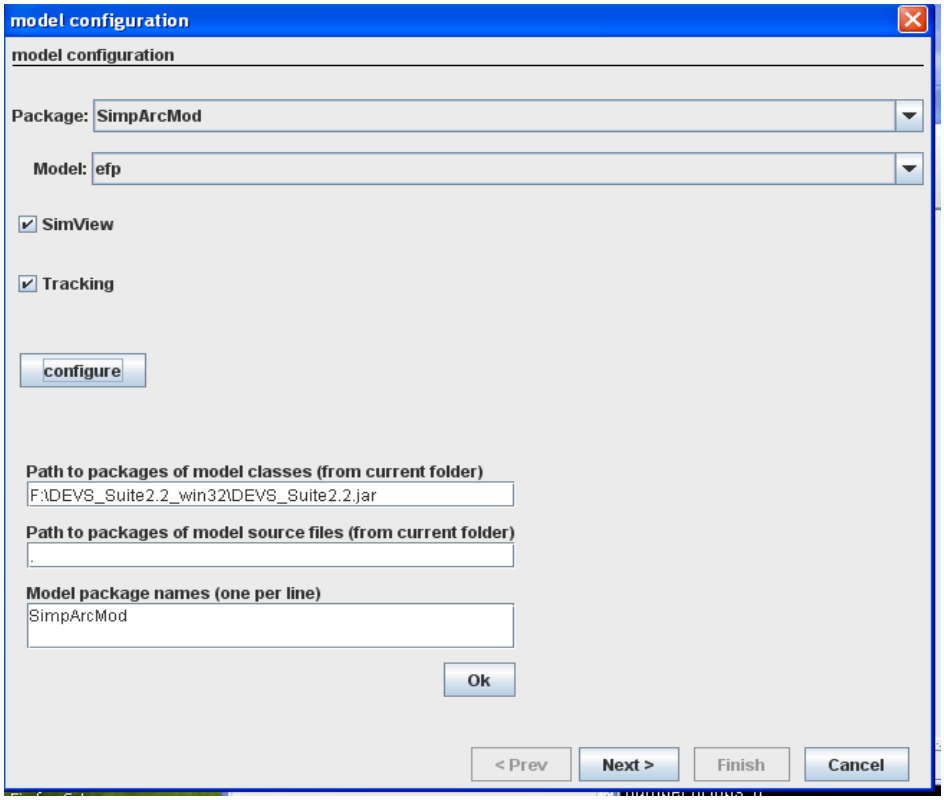

Figure 36. Model Configuration and Tracking Wizard

Step 2: Tracking of Components

The components in the model are displayed in JTree and it provides convenient access to view the tracked components. Choose the components to be tracked. Basic properties of the chart like axes units, increment of the time axis, and enabling the zero-time advance plots can be set in this step. Click the track button to confirm tracking of the component. All the tracked components are displayed in green, as shown in Figure 37. Click Finish.

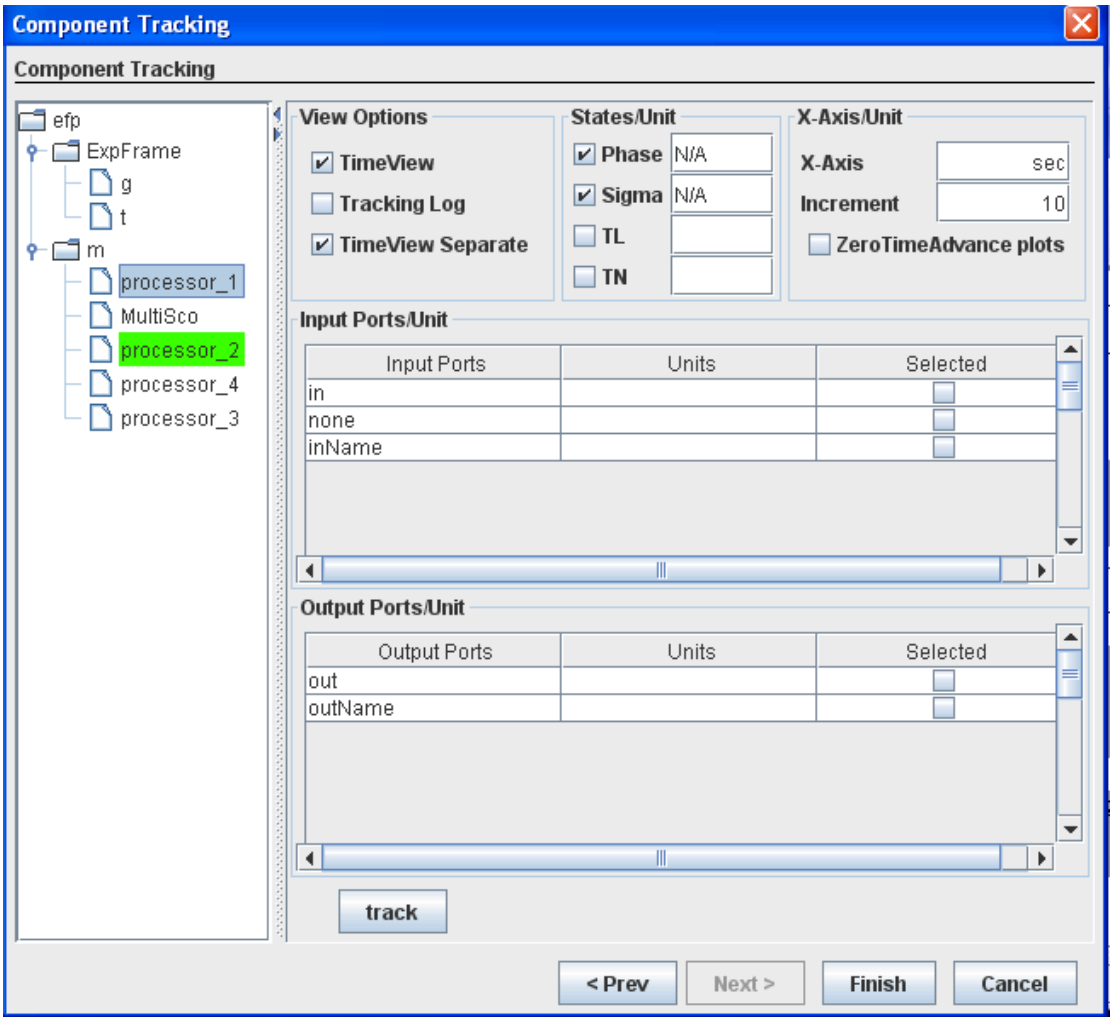

Figure 37. Choosing the Components to be Tracked, Plots to be Rendered and Setting the Basic Chart Properties

Step 3: Rendering the Plots in Time View Scroll Panel

The Processor_1 component is displayed in a separate time view scroll pane. Phase and sigma graphs are displayed. Positive infinity band is displayed by default for the sigma graph. As multiple events take place in zero-time steps, the zero-time advance plots are displayed. As seen in Figure 38, main plot displays the last event that occurred and the zero-time advance plots render all the events for those time instances with the category axis representing time in an indexed order. The integrated plug-ins have the capability to render string, positive and negative values.

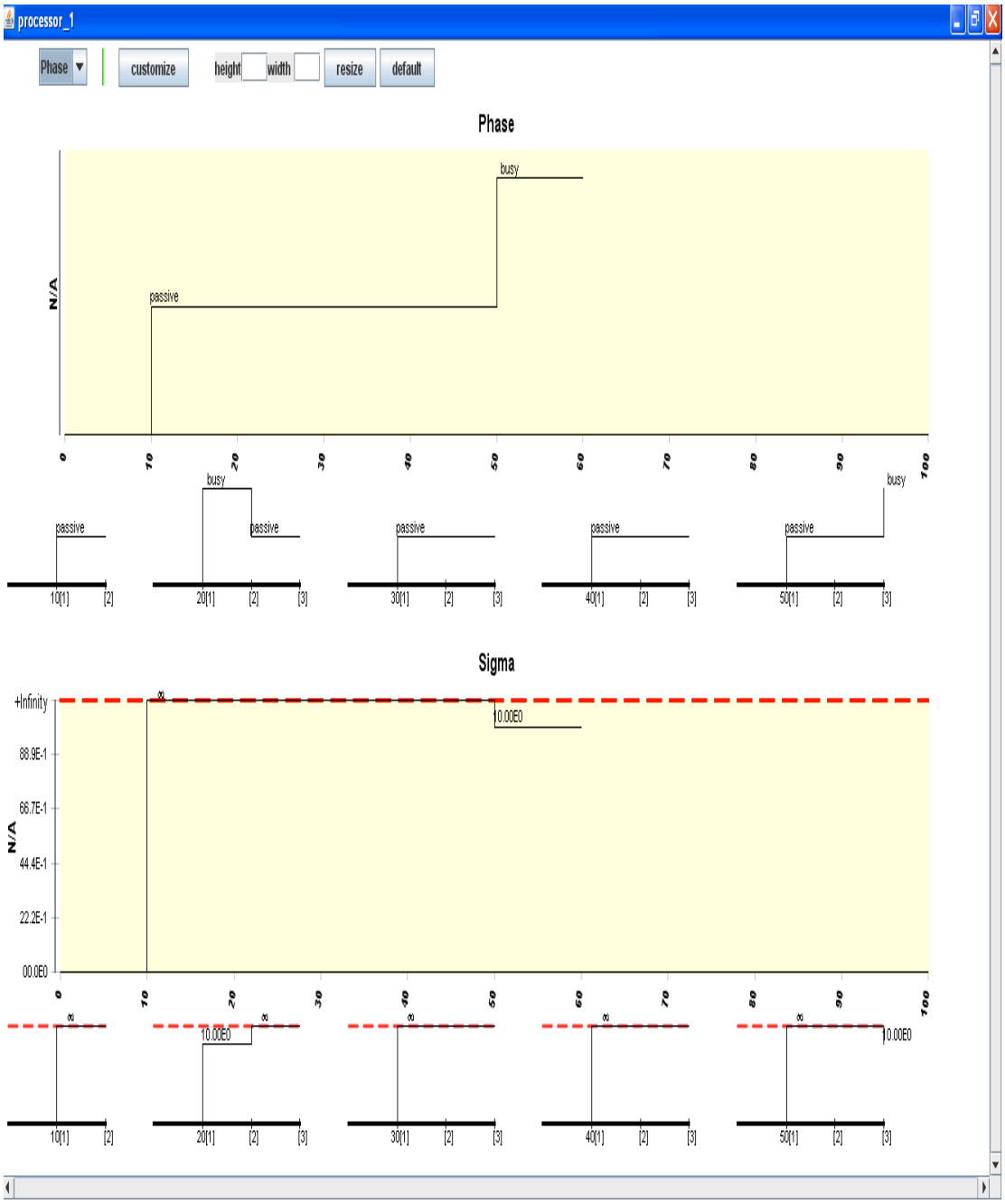

Figure 38. Plots Rendered with Zero-Time Advance

Step 4: Customization

Customization of the plots can be performed during simulation by choosing the plot to be customized form the combo box and clicking the customize button. The wizard is launched as shown in Figure 39. The integrated Time-Series plug-ins

provides the following customization features.

- Series label properties like font size, color, alignment, enabling and disabling the infinity band, and allocating the width of the band in percentage, series line color, enabling and disabling of zero-time advance plots, and its axes colors.
- X-Axis and Y-Axis properties like line color, thickness, label size, label color, and minimum and maximum values of the axes.
- Title of the plot and legend properties.

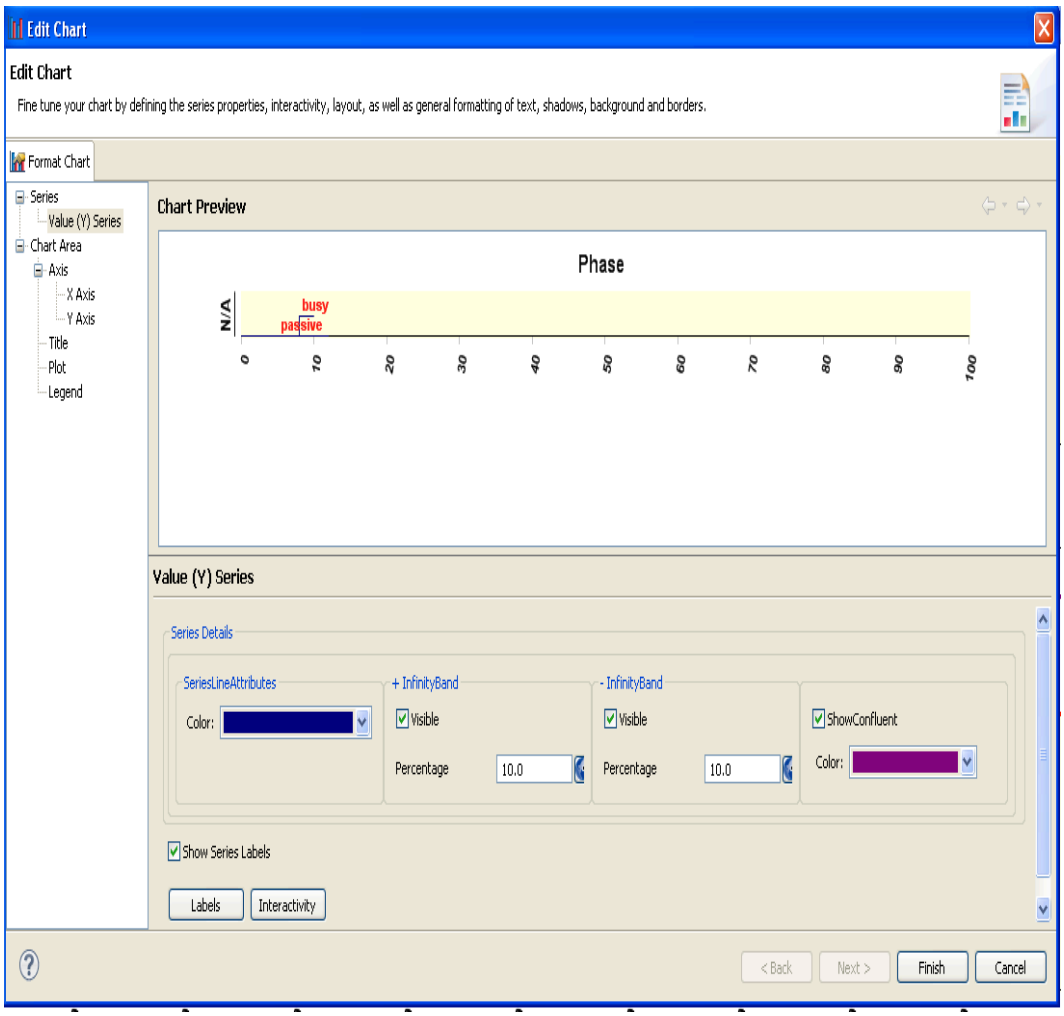

*Figure 39.*Customization Wizard for Advanced Chart Properties

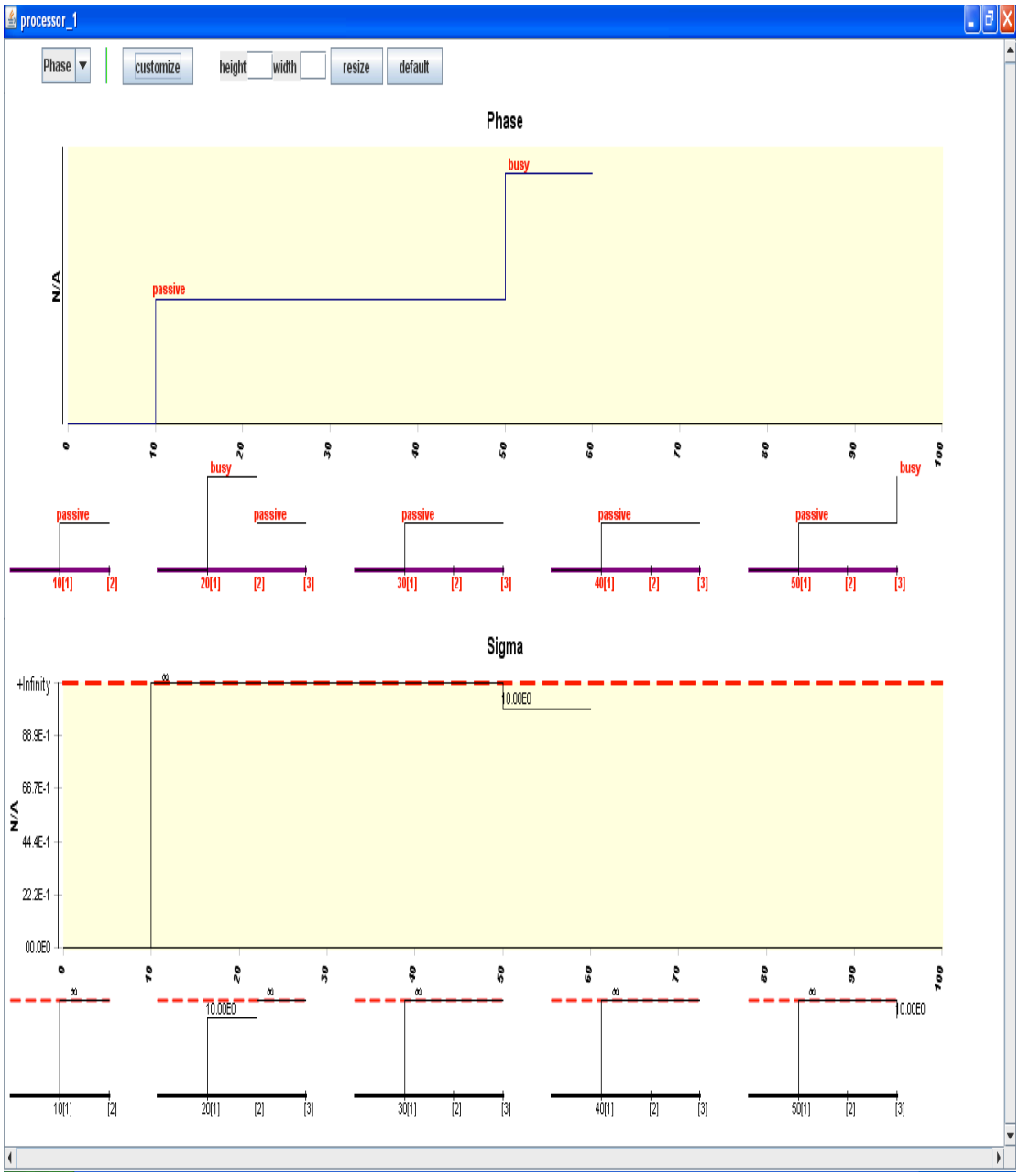

Figure 40. Plots After Customization

After the properties of the charts are modified, click Finish to close the wizard. The changes are reflected in the phase plot as shown in Figure 40. The height and width of the plot can be modified as desired by entering the values in the text box. The change is reflected across all the plots in the time view plots.

CHAPTER 6

CONCLUSIONS AND FUTURE WORK

6.1 CONCLUSION

In this thesis, we have developed two new 2-D time-series with superdense time segments based on its mathematical specification using the BIRT framework. The design specification support linear and superdense time piecewise constant and event chart types with positive and negative infinity bands. The plug-ins are integrated into the DEVS-Suite simulator to support displaying input, output, and state trajectories for parallel atomic DEVS models. Example DEVS models were developed to handle different scales of linear and logarithmic values for Y-axis. Further time-series plug-ins—piecewise constant and event plug-ins—support advanced visualization and customization capabilities by extending BIRT based on Eclipse plug-in architecture. The developed plug-ins were tested in these applications.

Developing the time-series plug-ins and testing in the above scenarios has led to many discoveries about the software architectures especially through BIRT's extension capability for reuse. BIRT, based on Eclipse plug-in architecture, proved to be the appropriate tool of choice amongst other plotting tools mainly because of its component reusability through its extension points, its API design providing flexibility to integrate the plug-ins with other JAVA applications, and its industrial strength update system. The robust framework that allows focusing on the core business logic proved to be another huge advantage. Though Eclipse plug-in architecture offered these advantages, a few downsides were experienced.

- Understanding the eclipse plug-in architecture and figuring out the appropriate extension points to be used has a huge learning curve for developers new to this environment.
- Though the extension point concept of plug-in based architecture proved to be helpful, the restriction it imposes from extending the class of our choice did not help for all the use cases. Zero-time advance plots has a unique requirement where multiple plots with different X-Axis scales need to be rendered. BIRT did not offer a suitable extension point and a work around had to be found to solve this problem. Designing for extensibility to meet all the use cases takes good software engineering practices.
- Setting up the environment for developing and testing the extended BIRT plug-ins can be complex as it has a dependency on Eclipse Modeling Framework and other BIRT plug-ins.
- Though BIRT API offered support for integrating the extended plug-ins with JAVA application, analyzing their dependencies proved to be a tedious task.

Though the above-mentioned tradeoffs exist in extending plug-in based architecture for developing the time-series plots, it performed well in terms of usability and robustness.

6.2 Future Work

Other charts in BIRT can be integrated with DEVS-Suite 3.0 to visualize the simulation data. The design of the new time view component also supports easy integration of the new charts, if added. Further plans exist for rendering plots with multiple data sets and integration of the BIRT Report engine with DEVS-Suite 3.0.

REFERENCES

D'Anjou, J., Fairbrother, S., Kehn, D., Kellerman, J., & McCarthy, P. (YEAR).The Java Developer's Guide to ECLIPSE (2nd ed.). Addison Wesley.

Silva, V. (2009). Practical eclipse rich client platform projects.

Weatherby, J., Bondur, T., & Chatalbasheva, I. (2011). Integrating and Extending BIRT (3rd ed.).

Zeigler, B.P., Praehofer, H., & Kim , T. (2000). Theory of Modelling and Simulation (2nd ed.).

Sarjoughian, H.S., Singh, R. (2003). Building simulation modeling environments using systems theory and software architecture principles. Advanced Simulation Technology Conference, SCS, Wash. DC.

Kim, S., Sarjoughian, H. S., Elamvazuthi, V. (2009). DEVS-suite: A Simulator Supporting Visual Experimentation Design and Behavior Monitoring. Proceedings of the 2009 Spring Simulation Multiconference. San Diego, California: Society for Computer Simulation International.

Chow, A.C , Zeigler, B.P. (1994). Parallel DEVS: a parallel, hierarchical, modular modeling formalism. *Simulation Conference Proceedings, 1994. Winter* (pp. 716 - 722). IEEE.

Zeigler, B.P., Sarjoughian, H.S., and Au, V., (1997) .Object-oriented DEVS: object behavior specification, Proceedings of Enabling Technology for Simulation Science, Orlando, FL.

Kincaid, R., (2010). SignalLens: Focus+Context Applied to Electronic Time Series. IEEE Transactions on Visualization and Computer Graphics, vol.16, no.6.

Wymore, A.W., (1993). Model-Based Systems Engineering, CRC Press, Boca Raton.

Sarjoughian, H. S., Sundaramoorthi, S., (2015). Superdense Time Trajectories for DEVS Simulation Models. Symposium On Theory of Modeling and Simulation (TMS'15), Virginia, U.S.A.

Manna, Z., Pnueli, A., The Temporal Logic of Reactive and Concurrent Systems, Springer, Berlin, 1993

Lee, E.A., (2014), *System Design, Modeling, and Simulation using Ptolemy II*, http://ptolemy.org.

Bostock, M., Ogievetsky, V., & Heer, J., (2011). D3 data-driven documents. Visualization and Computer Graphics, IEEE Transactions on, 17(12), 2301-2309.

Deri, L., Mainardi, S., & Fusco, F., (2012). tsdb: A compressed database for time series (pp. 143-156). Springer Berlin Heidelberg.

Department of Information Engineering, University of Pisa, Pisa, Italy 4 IBM Zürich Research Laboratory, Rüschlikon, Switzerland.

Actuate (2011). (Actuate) Retrieved 2013, Retrieved from http://developer.actuate.com/be/documentation/ihub2-web/birtos/ie21/index.html - page/Int-Birt/images/ArchitecturalOverview_1_02_1.html

BIRT ChartEngine API(2013). Retrieved 2013, Retrieved from http://www.eclipse.org/birt/release20specs/BPS39 1.0.1 - Chart Engine Usage and Integration.pdf

BIRT 4.4. (2014). (Actuate) Retrieved 2014, Retrieved from http://www.eclipse.org/birt: http://www.eclipse.org/birt/documentation

ACIMS, 2015, Arizona Center for Integrative Modeling and Simulation, http://acims.asu.edu/software

DEVS-Suite simulator 2.1.0 (2009). (ACIMS) Retrieved 2011, Retrieved from http://devs-suitesim.sourceforge.net/

JFreeChart (2013). Retrieved 2013, Retrieved from www.jfree.org: http://www.jfree.org/index.html

Graphite (2011). Retrieved 2014, Retrieved from http://graphite.readthedocs.org/en/latest/index.html

MathWorks , Simulink (2011). Retrieved from http://www.mathworks.com/products/simulink/

PtolemyII 8.0. (2010). Retrieved from http://ptolemy.eecs.berkeley.edu/ptolemyII/

JAVA 7. (2011). (Oracle) Retrieved from http://www.oracle.com/technetwork/java/javase/downloads/index.html

J2EE 7. (2013). (Oracle) Retrieved from 2013, from https://docs.oracle.com/javaee/

Cairo. (2014). Retrieved from http://cairographics.org/

APPENDIX A

SETUP OF ECLIPSE IDE FOR BUILDING, TESTING, AND DEPLOYING TIME-SERIES PLUG-INS IN

1. *SET-UP FOR BUILDING TIME-SERIES PLUG-INS*

Following points explains in detail the environment required for building the BIRT time-series plug-ins. The plug-ins were developed using Eclipse Juno and tested in Juno, Kepler, and Luna versions.

- Download Eclipse Version: Eclipse classic 4.2.2 (Juno) from the following link: http://www.eclipse.org/downloads/packages/release/juno/sr1. Eclipse SDK is the default. JDK Version 1.7 is used.
- Go to Install New Software in the Help menu of Eclipse IDE and install Business Intelligence, Reporting and Charting Plug-ins (Version 4.2.x) from the link: file://localhost/Juno - http://download.eclipse.org:releases:juno. Or you can add the plug-ins from the following BIRT repository: http://download.eclipse.org/birt/update-site/4.2
- Import the time-series plug-ins into a new workspace and test it by clicking on the link "Launch an Eclipse Application" in the plugin.xml of time-series plug-ins—org.eclipse.birt.chart.piecewiseconstant or org.eclipse.birt.chart.event.
- The chart engine uses an XSD SDK to generate an EMF model. To modify the source code or xsd files of the time-series plug-ins, the EMF plug-ins are required.
- Go to Install New Software in the Help menu of the Eclipse IDE expand Modeling and select EMF (Eclipse Modeling Framework SDK) and XSD (XML Schema definition) SDK items from the following link: file://localhost/Juno - http://download.eclipse.org:releases:juno.

- The plug-in with the source code can be obtained from the source download. http://www.eclipse.org/downloads/download.php?file=/birt/downloads/drops/R-R1-4_2_2-201302161152/birt-source-4_2_2.zip. Technically, only the XSD directory and the existing EMF model are required for modification, although it is useful to have the chart engine source for debugging purposes.

2. *SETUP FOR TESTING THE DEVELOPED PLUG-INS*

The following points explain in detail how to test the extended time-series plug-in by the developer.

- To test the time-series plug-ins, run the org.eclipse.birt.chart.piecewiseconstant or org.eclipse.birt.chart.event package as an Eclipse application. To run the plug-in as an eclipse application, follow these steps.
    - When testing for the first time, create a run configuration Eclipse Application and choose the tab Plug-ins to make sure that the chosen time-series plug-ins from the workspace and the dependent plug-ins from the target platform are included.
    - Choose Run->Run As->Eclipse Application. Eclipse builds a new default workspace. Eclipse opens a new instance of the Eclipse workbench that displays the welcome page.
- Open the Report Design perspective. Choose Window→Open Perspective→Report Design. If Report Design is not available, choose Other. On Open Perspective, select Report Design, then choose OK. Create a blank

report and drag the chart type from the Palette. Time View chart will be included in the chart wizard.

The following points explain in detail how to export the time-series plug-ins and deploy them in BIRT by the user.

- To deploy the time-series plug-ins along with other eclipse plug-ins, go to Export→Plug-in Developments→Deployable plug-ins and fragments in Plug-in Development Environment (PDE) perspective. An export wizard appears.
- In Available Plug-ins and Fragments, select the org.eclipse.birt.chart.timeview plug-in to export.
- In Export Destination, specify the Archive file or Directory where the plug-in JAR file should be placed.
- In Export Options, select one of the following if necessary: 1. Include source code. 2. Package plug-ins as individual JAR archives. 3. Save as Ant script.
- Choose Finish to export the plug-in to the specified destination.
- Copy the create JAR file into the Eclipse Application's plug-ins folder.
- Restart Eclipse so it will find the time-series plug-in and include it in the chart wizard.

APPENDIX B

DEPENDENCY ANALYSIS AND ENVIRONMNETAL SETUP

FOR DEVS-SUITE 3.0

1. ***EXTERNAL DEPENDENCY ANALYSIS OF DEVS-SUITE***

The following gives an analysis on the external JAR files dependencies of DEVS-Suite 3.0. The external dependencies were analyzed by a combination of the JBOSS tattletale tool and brute force.

This tool generates reports that will show dependencies and general information that can help identify areas that need attention such as minimizing the number of dependencies or eliminating duplicated class files from the class path. As the tool analyzes static dependencies from the JAR file archives, elimination of many unused external JAR dependency is made easy. This tool does not detect runtime dependencies. By using brute force the narrowed down JAR files were eliminated. The list of external dependencies is given below.

1. org.eclipse.birt.chart.piecewise constant.jar
2. org.eclipse.birt.chart.event.jar
3. Chartengineapi.jar
4. Com.ibm.icu.jar
5. org.eclipse.birt.core.jar
6. Js.jar
7. org.eclipse.core.runtime.jar
8. org.eclipse.emf.ecore.xmi.jar
9. org.eclipse.emf.ecore.jar
10. org.eclipse.emf.common.jar
11. org.eclipse.osgi.jar
12. org.eclipse.osgi.services.jar
13. org.eclipse.birt.chart.device.extension.jar
14. org.eclipse.birt.chart.engine.extension.jar
15. org.eclipse.equinox.preferences.jar
16. org.eclipse.equinox.registry.jar
17. org.eclipse.equinox.common.jar
18. org.eclipse.birt.chart.ui.jar
19. org.eclipse.birt.chart.ui.extension.jar

20. org.eclipse.birt.core.ui.jar
21. org.eclipse.jface.jar
22. org.eclipse.swt.win32.win32.x86_64.jar
23. org.eclipse.birt.chart.device.swt.jar
24. org.eclipse.ui.workbench.jar
25. org.eclipse.e4.core.di.jar
26. org.eclipse.e4.ui.model.workbench.jar
27. Org.eclipse.e4.ui.workbench3_0.12.0.jar
28. Org.eclipse.core.jobs.jar
29. Org.eclipse.core.commands.jar

Items 1, 2 are the extended BIRT plug-ins to draw the piecewise constant and event plots. The extended BIRT plug-ins (1, 2) are dependent on the plug-ins 3-17. The chart wizard integrated with DEVS-Suite depends on Eclipse workbench and BIRT chart UI plug-ins. 18-29 are JAR files that are related to the chart wizard. I have not included the versions of the JAR files. The Eclipse SWT JAR is OS dependent and varies accordingly.

2. EXECUTION ENVIRONMENT FOR DEVS-SUITE

As DEVS-Suite is a standalone Java project, it can be run independently of IDE. As the DEVS-Suite is dependent upon Eclipse framework, all the required external dependencies, including OSGI, equinox, and the other framework related plug-ins (see Section 1) have been added to its class path. Any dependency needed is to be added to its class path as mentioned above. This was tested both in Eclipse and NetBeans IDE.

Eclipse:

- Download Eclipse IDE (standard edition).
- Create a new Java project and point the location to the DEVS-Suite files.

- Add the above-mentioned JARs as external dependencies. The JAR files are located in the plug-in folder.

NetBeans:

- Download NetBeans IDE 8.0 or another version.
- Select File -> New Project from the main menu. Choose the Java category and the Java Project with the Existing Sources project type.
- Select Next. Enter the name for the project as DEVS-Suite.
- Select Next. Browse to the folder that contains the source files.
- Select Next. You will see the list of files that are to be added. Click Finish on this page.
- Add the JARs files mentioned in Section 1 as external dependencies.