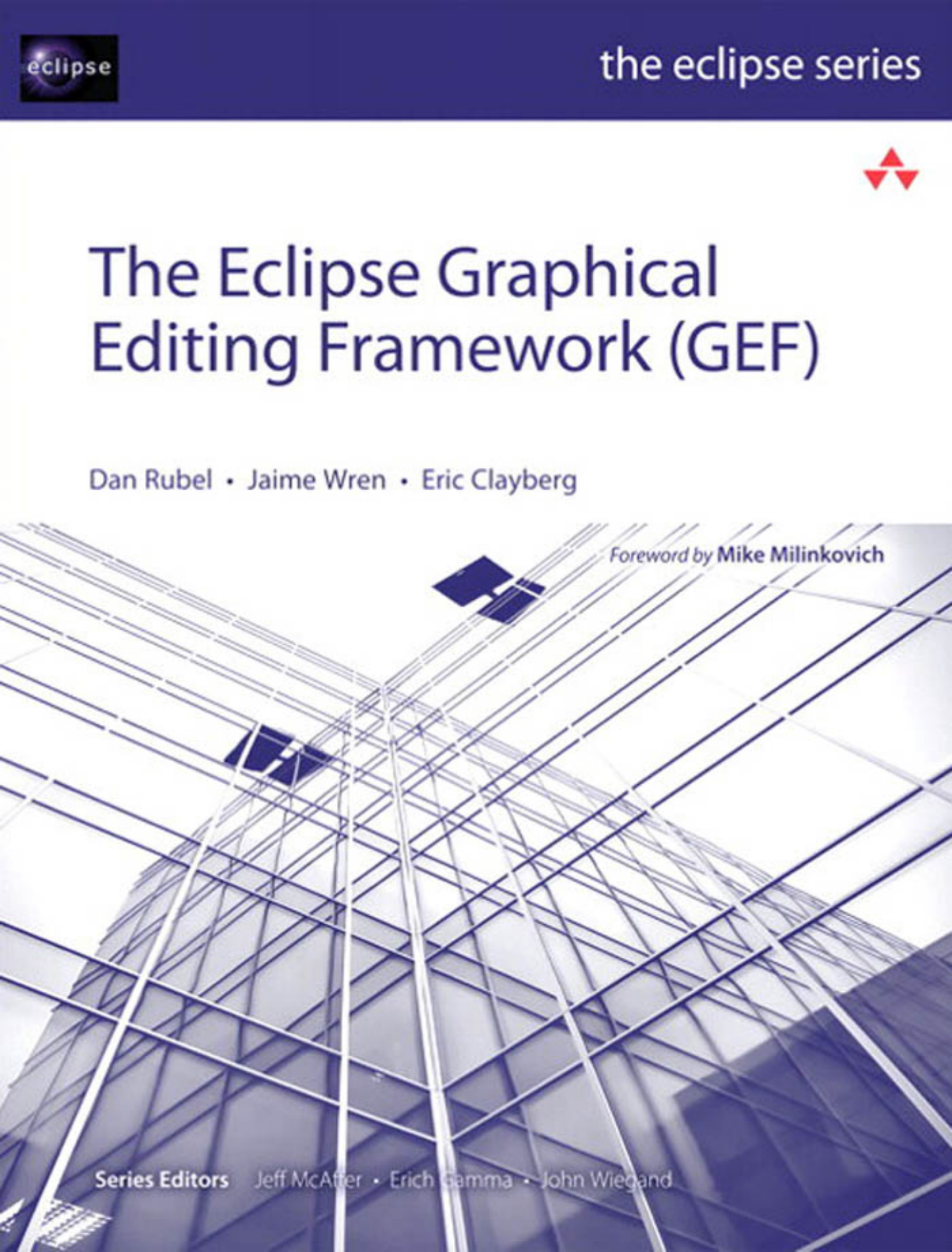

eclipse
the eclipse series
The Eclipse Graphical Editing Framework (GEF)
Dan Rubel • Jaime Wren • Eric Clayberg
Foreword by Mike Milinkovich
Series Editors Jeff McAffer • Erich Gamma • John Wiegand

Praise for Clayberg and Rubel's *Eclipse Plug-ins, Third Edition*

"Dan Rubel and Eric Clayberg are the authors of one of the most highly regarded books in the history of Eclipse. Their *Eclipse Plug-ins* is generally considered the seminal book on how to extend the Eclipse platform."

— Mike Milinkovich
Executive Director, Eclipse Foundation

"I'm often asked, 'What are the best books about Eclipse?' Number one on my list, every time, is *Eclipse Plug-ins*. I find it to be the clearest and most relevant book about Eclipse for the real-world software developer. Other Eclipse books focus on the internal Eclipse architecture or on repeating the Eclipse documentation, whereas this book is laser focused on the issues and concepts that matter when you're trying to build a product."

— Bjorn Freeman-Benson
Former Director, Open Source Process, Eclipse Foundation

"As the title suggests, this massive tome is intended as a guide to best practices for writing Eclipse plug-ins. I think in that respect it succeeds handily. Before you even think about distributing a plug-in you've written, read this book."

— Ernest Friedman-Hill
Marshall, JavaRanch.com

"If you're looking for just one Eclipse plug-in development book that will be your guide, this is the one. While there are other books available on Eclipse, few dive as deep as *Eclipse Plug-ins*."

— Simon Archer

"*Eclipse Plug-ins* was an invaluable training aid for all of our team members. In fact, training our team without the use of this book as a base would have been virtually impossible. It is now required reading for all our developers and helped us deliver a brand-new, very complex product on time and on budget thanks to the great job this book does of explaining the process of building plug-ins for Eclipse."

— Bruce Gruenbaum

"The authors of this seminal book have decades of proven experience with the most productive and robust software engineering technologies ever developed. Their experiences have now been well applied to the use of Eclipse for more effective Java development. A must-have for any serious software engineering professional!"

— Ed Klimas

"This is easily one of the most useful books I own. If you are new to developing Eclipse plug-ins, it is a 'must-have' that will save you *lots* of time and effort. You will find lots of good advice in here, especially things that will help add a whole layer of professionalism and completeness to any plug-in. The book is very focused, well-structured, thorough, clearly written, and doesn't contain a single page of 'waffly page filler.' The diagrams explaining the relationships between the different components and manifest sections are excellent and aid in understanding how everything fits together. This book goes well beyond Actions, Views, and Editors, and I think everyone will benefit from the authors' experience. I certainly have."

— Tony Saveski

"Just wanted to also let you know this is an excellent book! Thanks for putting forth the effort to create a book that is easy to read *and* technical at the same time!"

— Brooke Hedrick

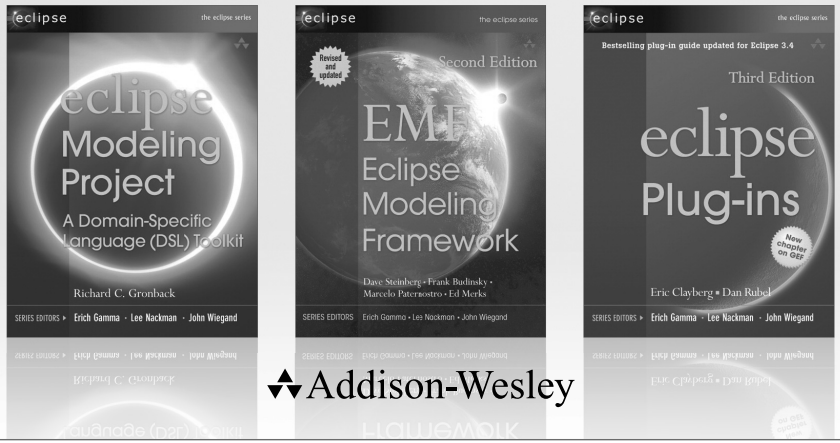
"The key to developing great plug-ins for Eclipse is understanding where and how to extend the IDE, and that's what this book gives you. It is a must for serious plug-in developers, especially those building commercial applications. I wouldn't be without it."

— Brian Wilkerson

The Eclipse Graphical Editing Framework (GEF)

The Eclipse Series

Eric McAffer, Erich Gamma, John Wiegand, Series Editors



Visit **informit.com/series/eclipse** for a complete list of available publications.

Eclipse is a universal, multilanguage software development environment—an open, extensible, integrated development environment (IDE)—that can be used for anything. Eclipse represents one of the most exciting initiatives to come from the world of application development, and it has the support of leading companies and organizations in the technology sector. Eclipse is gaining widespread acceptance in both commercial and academic arenas.

The Eclipse Series is the definitive collection of publications dedicated to the Eclipse platform. Books in this series bring you key technical information, critical insight, and the practical advice you need to build tools to support this revolutionary open-source platform.

PEARSON

Addison-Wesley Cisco Press EXAM/CRAM IBM Press. QUE PRENTICE HALL SAMS | Safari Books Online

The Eclipse Graphical Editing Framework (GEF)

Dan Rubel
Jaime Wren
Eric Clayberg

Addison-Wesley

Upper Saddle River, NJ • Boston • Indianapolis • San Francisco
New York • Toronto • Montreal • London • Munich • Paris • Madrid
Capetown • Sydney • Tokyo • Singapore • Mexico City

Library of Congress Cataloging-in-Publication Data

Rubel, Dan.
The Eclipse Graphical Editing Framework (GEF) / Dan Rubel, Jaime Wren, Eric Clayberg.
p. cm.
Includes bibliographical references and index.
ISBN 0-321-71838-0 (pbk. : alk. paper)
1. Graphical user interfaces (Computer systems) 2. Graphic methods—Computer programs. 3. Eclipse (Electronic resource) 4. Computer software—Development. I. Wren, Jaime. II. Clayberg, Eric. III. Title.

QA76.9.U83R814 2012
005.4'37--dc23 2011023246

ISBN-13: 978-0-321-71838-9
ISBN-10: 0-321-71838-0

Text printed in the United States on recycled paper at RR Donnelley in Crawfordsville, Indiana.

First printing, August 2011

To the women we love,
Kathy, Helene, and Karen

Contents

Foreword

The Eclipse Graphical Editor Framework (GEF) project supports the creation of rich graphical editors and views for Eclipse-based tools and Rich Client Platform (RCP) applications. GEF's three frameworks—Draw2D, Zest, and GEF—are amongst the most widely used within the Eclipse community and ecosystem.

"Mighty oaks from little acorns grow" is the story of the GEF project. In the context of the Eclipse community, GEF is a relatively small project. But the tools, applications, and products that have been enabled by GEF form a very long list indeed. Everything from mission planning for the Mars Rovers to most of the world's commercial modeling tools make use of GEF. GEF is also widely re-used within the Eclipse community itself, and is leveraged by Eclipse projects such as GMF, Graphiti, AMP, Sphinx and Papyrus. It is a testament to the idea that a small, powerful, and open source framework can make an enormous impact on the industry.

A big part of the success of the GEF project and its three frameworks has been its long-term focus on being a platform. Although there has been a steady flow of innovative new features, the quality, stability, and backwards compatibility of the GEF project APIs have been a big part of its success. That level of commitment to the "platformness" (to coin a phrase) of a framework is the hallmark of a great project at Eclipse. It requires a great deal of commitment and discipline by the project team to accomplish.

Eclipse projects are powered by people, so I would like to recognize the contributions of the present GEF project leader Anthony Hunter, and the past leaders Randy Hudson and Steven Shaw, all of IBM. I would also like to recognize the many contributions of the projects committers past and present: Nick Boldt, Alex Boyko, Ian Bull, Marc Gobeil, Alexander Nyssen, Cherie

Revells, Pratik Shah, and Fabian Steeg. I would also like to recognize the contributions and investments of IBM, itemis AG, EclipseSource, and Tasktop in supporting the team working on GEF.

Dan Rubel and Eric Clayberg are the authors of one of the most highly regarded books in the history of Eclipse. Their *Eclipse Plug-ins* is generally considered the seminal book on how to extend the Eclipse platform. Dan and Eric, this time joined by their colleague Jaime Wren, have brought their clear prose, deep knowledge, and focus on the issues that matter to developers using the Eclipse GEF framework to this new book. I know that you will find it a useful addition to your Eclipse library.

—Mike Milinkovich
Executive Director
Eclipse Foundation, Inc.

Preface

When we were first exposed to Eclipse back in late 1999, we were struck by the magnitude of the problem IBM was trying to solve. IBM wanted to unify all its development environments on a single code base. At the time, the company was using a mix of technology composed of a hodgepodge of C/C++, Java, and Smalltalk.

Many of IBM's most important tools, including the award-winning VisualAge for Java IDE, were actually written in Smalltalk—a wonderful language for building sophisticated tools, but one that was rapidly losing market share to languages like Java. While IBM had one of the world's largest collections of Smalltalk developers, there wasn't a great deal of industry support for it outside of IBM, and very few independent software vendors (ISVs) were qualified to create Smalltalk-based add-ons.

Meanwhile, Java was winning the hearts and minds of developers worldwide with its promise of easy portability across a wide range of platforms, while providing the rich application programming interface (API) needed to build the latest generation of Web-based business applications. More important, Java was an object-oriented (OO) language, which meant that IBM could leverage the large body of highly skilled object-oriented developers it had built up over the years of creating Smalltalk-based tools. In fact, IBM took its premier Object Technology International (OTI) group, which had been responsible for creating IBM's VisualAge Smalltalk and VisualAge Java environments (VisualAge Smalltalk was the first of the VisualAge brand family, and VisualAge Java was built using it), and tasked the group with creating

a highly extensible integrated development environment (IDE) construction set based in Java. Eclipse was the happy result.

OTI was able to apply its highly evolved OO skills to produce an IDE unmatched in power, flexibility, and extensibility. The group was able to replicate most of the features that had made Smalltalk-based IDEs so popular the decade before, while simultaneously pushing the state of the art in IDE development ahead by an order of magnitude.

The Java world had never seen anything as powerful or as compelling as Eclipse, and it now stands, with Microsoft's .NET, as one of the world's premier development environments. That alone makes Eclipse a perfect platform for developers wishing to get their tools out to as wide an audience as possible. The fact that Eclipse is completely free and open source is icing on the cake. An open, extensible IDE base that is available for free to anyone with a computer is a powerful motivator to the prospective tool developer.

It certainly was to us. At Instantiations and earlier at ObjectShare, we had spent the better part of a decade as entrepreneurs focused on building add-on tools for various IDEs. We had started with building add-ons for Digitalk's Smalltalk/V, migrated to developing tools for IBM's VisualAge Smalltalk, and eventually ended up creating tools for IBM's VisualAge Java (including our award-winning VA Assist product and our jFactor product, one of the world's first Java refactoring tools). Every one of these environments provided a means to extend the IDE, but they were generally not well documented and certainly not standardized in any way. Small market shares (relative to tools such as VisualBasic) and an eclectic user base also afflicted these environments and, by extension, us.

As an Advanced IBM Business Partner, we were fortunate to have built a long and trusting relationship with the folks at IBM responsible for the creation of Eclipse. That relationship meant that we were in a unique position to be briefed on the technology and to start using it on a daily basis nearly a year-and-a-half before the rest of the world even heard about it. When IBM finally announced Eclipse to the world in mid-2001, our team at Instantiations had built some of the first demo applications IBM had to show. Later that year, when IBM released its first Eclipse-based commercial tool, WebSphere Studio Application Developer v4.0 (v4.0 so that it synchronized with its then-current VisualAge for Java v4.0), our CodePro product became the very first commercial add-on available for it (and for Eclipse in general) on the same day. Two years later, we introduced our first GEF-based tool, WindowBuilder Pro, a powerful graphical user interface (GUI) development tool.

Developing WindowBuilder over the last several years has provided us with an opportunity to learn the details of Eclipse GEF development at a level matched by very few others. WindowBuilder has also served as a testbed for

many of the ideas and techniques presented in this book, providing us with a unique perspective from which to write.

WindowBuilder's product suite (especially GWT Designer) caught the attention of Google, which acquired Instantiations in August of 2010. Since the acquisition Google has donated the WindowBuilder architecture and the two projects, SWT Designer and Swing Designer, to the Eclipse Foundation. The GWT Designer product has been folded into the Google Plug-in for Eclipse (GPE).

Goals of the Book

This book provides an in-depth description of the process involved in building Eclipse GEF-based tools and editors. This book has several complementary goals:

- To provide a quick introduction to GEF for new users
- To provide a reference for experienced Eclipse GEF users wishing to expand their knowledge and improve the quality of their GEF-based products
- To provide a detailed tutorial on creating sophisticated GEF tools suitable for new and experienced users

The first chapter introduces GEF, Draw2D, and Zest and includes examples of what has been built using GEF. The next two chapters outline the process of building a simple Draw2D example. The intention of these chapters is to help developers new to GEF quickly understand and pull together an example they can use to experiment with.

The next five chapters progressively introduce the reader to more and more of the Draw2D framework that forms the foundation of GEF. The fourth chapter introduces figures, which are the building blocks for the rest of the book. Chapters 5 through 8 bring the user through the complete Draw2D Genealogy example, introducing concepts such as layout managers, connections, layers, and viewports.

The ninth chapter presents Zest, a graph visualization project part of GEF.

The remaining chapters present the non-Draw2D portions of the GEF project, including `EditParts`, `EditPolicies`, tools, commands, and actions. These chapters walk the user through the development of a GEF Editor for a genealogy model.

Each chapter focuses on a different aspect of the topic and includes an overview, a detailed description, a discussion of challenges and solutions, diagrams, screenshots, cookbook-style code examples, relevant API listings, and a summary.

Sometimes a developer needs a quick solution, while at other times that same developer needs to gain in-depth knowledge about a particular aspect of development. The intent is to provide several different ways for the reader to absorb and use the information so that both needs can be addressed. Relevant APIs are included in several of the chapters so that the book can be used as a standalone reference during development without requiring the reader to look up those APIs in the IDE. Most API descriptions are copied or paraphrased from the Eclipse platform Javadoc.

The examples provided in the chapters describe building various aspects of a concrete Eclipse GEF-based plug-in that will evolve over the course of the book. When you use the book as a reference rather than read it cover to cover, you will typically start to look in one chapter for issues that are covered in another. To facilitate this type of searching, every chapter contains numerous cross-references to related material that appears in other chapters.

Intended Audience

The audience for this book includes Java tool developers wishing to build graphical editing products that integrate with Eclipse and other Eclipse-based products, relatively advanced Eclipse users wishing to build their own graphical tools, or anyone who is curious about what makes Eclipse GEF tick. You should be a moderately experienced Eclipse developer to take full advantage of this book. If you are new to Eclipse or Eclipse plug-in development, we recommend starting with our companion book, *Eclipse Plug-ins*. We also anticipate that the reader is a fairly seasoned developer with a good grasp of Java and at least a cursory knowledge of extensible markup language (XML).

Conventions Used in This Book

The following formatting conventions are used throughout the book.

Bold—the names of UI elements such as menus, buttons, field labels, tabs, and window titles

Italic—emphasize new terms

`Courier`—code examples, references to class and method names, and filenames

`Courier Bold`—emphasize code fragments

"Quoted text"—indicates words to be entered by the user

Acknowledgments

The authors would like to thank all those who have had a hand in putting this book together or who gave us their support and encouragement throughout the many months it took to create.

To our comrades at Instantiations and Google, who gave us the time and encouragement to work on this book: Rick Abbott, Brad Abrams, Brent Caldwell, Devon Carew, David Carlson, David Chandler, Jim Christensen, Taylor Corey, Rajeev Dayal, Dianne Engles, Marta George, Nick Gilman, Seth Hollyman, Alex Humesky, Bruce Johnson, Mark Johnson, Ed Klimas, Tina Kvavle, Florin Malita, Warren Martin, Miguel Méndez, Steve Messick, Alexander Mitin, Gina Nebling, John O'Keefe, Keerti Parthasarathy, Phil Quitslund, Chris Ramsdale, Mark Russell, Rob Ryan, Andrey Sablin, Konstantin Scheglov, Chuck Shawan, Bryan Shepherd, Julie Taylor, Mike Taylor, Solveig Viste, Andrew Wegley, and Brian Wilkerson.

To our editor, Greg Doench, our production editor, Elizabeth Ryan, our copy editor, Barbara Wood, our editorial assistant, Michelle Housley, our marketing manager, Stephane Nakib, and the staff at Pearson, for their encouragement and tremendous efforts in preparing this book for production.

To the series editors, Erich Gamma, Lee Nackman, and John Wiegand, for their thoughtful comments and for their ongoing efforts to make Eclipse the best development environment in the world.

We would also like to thank our wives, Kathy, Helene, and Karen, for their endless patience, and our children, Beth, Lauren, Lee, and David, for their endless inspiration.

About the Authors

Dan Rubel is Senior Software Engineer for Google. He is an entrepreneur and an expert in the design and application of OO technologies with more than 17 years of commercial software development experience, including 15 years of experience with Java and 11 years with Eclipse. He is the architect and product manager for several successful commercial products, including RCP Developer, WindowTester, jFactor, and jKit, and has played key design and leadership roles in other commercial products such as VA Assist, and CodePro. He has a B.S. from Bucknell and was a cofounder of Instantiations.

Jaime Wren is Software Engineer for Google. He has worked with object-oriented technologies for the last nine years, and Eclipse tools for the past six years, gaining extensive expertise in developing commercial Eclipse-based tools. At Instantiations, Jaime made significant contributions as a developer on the CodePro and WindowBuilder product lines. After the acquisition of Instantiations by Google, he continues to work on the WindowBuilder product on the Google Web Toolkit (GWT) team. Jaime holds a double B.S. in Mathematics and Computer Science from the University of Oregon.

Eric Clayberg is Software Engineering Manager for Google. Eric is a seasoned software technologist, product developer, entrepreneur, and manager with more than 19 years of commercial software development experience, including 14 years of experience with Java and 11 years with Eclipse. He is the primary author and architect of more than a dozen commercial Java and Smalltalk add-on products, including the popular WindowBuilder, CodePro, and the award-winning VA Assist product lines. He has a B.S. from MIT, and an M.B.A. from Harvard, and has cofounded two successful software companies—ObjectShare and Instantiations.

Google is a multinational public corporation invested in Internet search, cloud computing, and advertising technologies. Google hosts and develops a number of Internet-based services and products, and its mission statement from the beguinning has been "to organize the world's information and make it universally accessible and useful."

How to Contact Us

While we have made every effort to make sure that the material in this book is timely and accurate, Eclipse is a rapidly moving target, and it is quite possible that you may encounter differences between what we present here and what you experience using Eclipse. The Eclipse UI has evolved considerably over the years, and the latest 3.7 release is no exception. While we have targeted it at Eclipse 3.7 and used it for all of our examples, this book was completed after Eclipse 3.6 was finished and during the final phases of development of Eclipse 3.7. If you are using an older or newer version of Eclipse, this means that you may encounter various views, dialogs, and wizards that are subtly different from the screenshots herein.

- Questions about the book's technical content should be addressed to info@qualityeclipse.com
- Sales questions should be addressed to Addison-Wesley at www.informit.com/store/sales.aspx
- Source code for the projects presented can be found at www.qualityeclipse.com/projects
- Errata can be found at www.qualityeclipse.com/errata
- Tools used and described can be found at www.qualityeclipse.com/tools

CHAPTER 2

A Simple Draw2D Example

Before covering the Draw2D infrastructure (see Chapter 3 on page 21) and each area of building a Draw2D diagram in depth, it is useful to create a simple example on which discussion can be based. This chapter takes a step-by-step approach to creating a simple Draw2D diagram representing the relationship between two people and their offspring. To start, we take an unsophisticated "brute force" approach, which we will refactor and refine in later chapters as we introduce more concepts. This process provides valuable first-hand experience using the Draw2D API.

2.1 Draw2D Installation

Select the **Help > Install New Software...** menu to install the GEF framework into Eclipse. When the **Install** wizard opens, select the Eclipse release update site (e.g., Indigo - http://download.eclipse.org/releases/indigo). Once the wizard refreshes, expand the **Modeling** category and select **Graphical Editing Framework GEF SDK** (see Figure 2–1). Alternatively, if you would like to install a different version of GEF, enter the GEF specific update site (http://download.eclipse.org/tools/gef/updates/releases) in the **Install** wizard and select the GEF features you wish to install. After you click **Finish** and restart Eclipse, the GEF framework installation is complete.

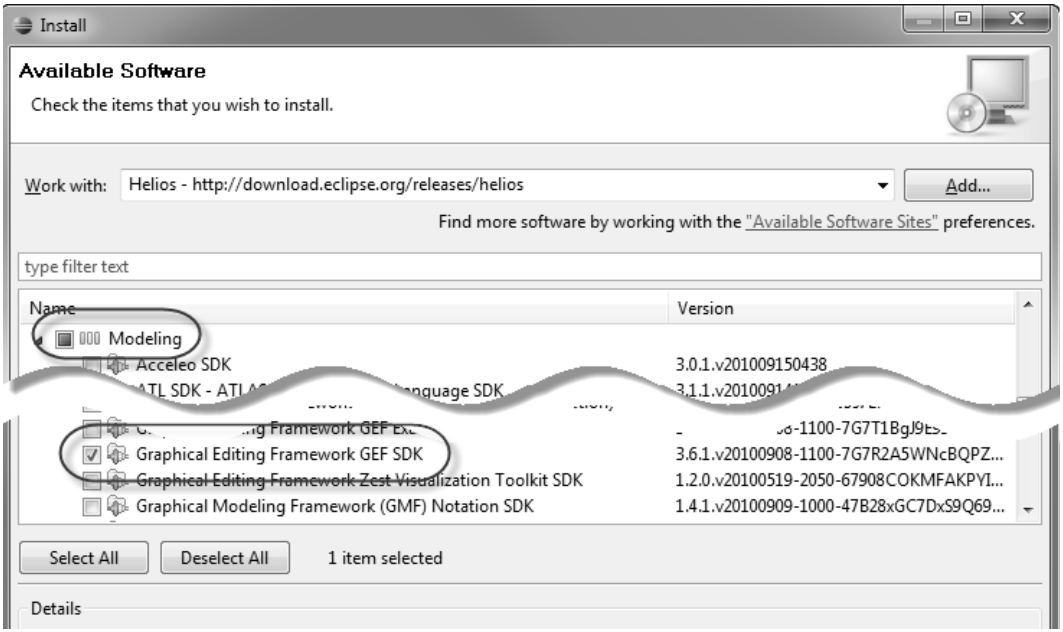


Figure 2–1 Install wizard.

2.2 Draw2D Project

The full Eclipse RCP framework is not needed to use the Draw2D framework, so if you are creating a simple Java application, you can create a simple Java project in Eclipse and modify its build path to include the following Eclipse JAR files:

- `ECLIPSE_HOME/plugins/org.eclipse.swt_3.7.X.vXXXX.jar`
- `ECLIPSE_HOME/plugins/org.eclipse.swt.win32.win32.x86_3.7.X.vXXXX.jar`
- `ECLIPSE_HOME/plugins/org.eclipse.draw2d_3.7.X.vXXXX.jar`

Alternatively, if you are creating a diagram as part of a larger Eclipse RCP application, then create a Plug-in project with the following plug-in dependencies (see Chapter 2 in the *Eclipse Plug-ins* book for more about Plug-in projects):

- `org.eclipse.ui`
- `org.eclipse.core.runtime`
- `org.eclipse.draw2d`

Since the second half of this book describes techniques that require the Eclipse RCP framework, we use a Plug-in project rather than a simple Java project for all of the samples in this book.

2.3 Draw2D Application

Since the full Eclipse RCP framework is not needed to use the Draw2D framework, we create a simple Java class containing a `main(…)` method.

```
package com.qualityeclipse.genealogy.view;

import org.eclipse.draw2d.*;
import org.eclipse.draw2d.geometry.*;
import org.eclipse.swt.SWT;
import org.eclipse.swt.layout.GridData;
import org.eclipse.swt.layout.GridLayout;
import org.eclipse.swt.widgets.Canvas;
import org.eclipse.swt.widgets.Composite;
import org.eclipse.swt.widgets.Display;
import org.eclipse.swt.widgets.Shell;

public class GenealogyView
{
   public static void main(String[] args) {
      new GenealogyView().run();
   }
}
```

Tip: All of this source can be downloaded from www.qualityeclipse.com.

The `main(…)` method calls a `run()` method to initialize the shell, create the diagram, and show the shell. The `run()` method is not interesting with respect to Draw2D and is included here only for completeness. For more information on SWT and shells, please see the *Eclipse Plug-ins* book.

```
private void run() {
   Shell shell = new Shell(new Display());
   shell.setSize(365, 280);
   shell.setText("Genealogy");
   shell.setLayout(new GridLayout());

   Canvas canvas = createDiagram(shell);
   canvas.setLayoutData(new GridData(GridData.FILL_BOTH));

   Display display = shell.getDisplay();
   shell.open();
   while (!shell.isDisposed()) {
      while (!display.readAndDispatch()) {
         display.sleep();
      }
   }
}
```

The `run()` method calls the `createDiagram(...)` method to create and populate the diagram. This method creates a root figure to contain all of the other figures in the diagram (see Chapter 4 on page 27 for more about figures). A simple layout manager (see Chapter 5 on page 55 for more about layout managers) is used to statically lay out the figures that are added later in this section. Finally, the last bit of code creates a `Canvas` on which the diagram is displayed and a `LightweightSystem` used to display the diagram (see Section 3.1 on page 21 for more about `LightweightSystem`).

```
private Canvas createDiagram(Composite parent) {

   // Create a root figure and simple layout to contain
   // all other figures
   Figure root = new Figure();
   root.setFont(parent.getFont());
   XYLayout layout = new XYLayout();
   root.setLayoutManager(layout);

   // Create a canvas to display the root figure
   Canvas canvas = new Canvas(parent, SWT.DOUBLE_BUFFERED);
   canvas.setBackground(ColorConstants.white);
   LightweightSystem lws = new LightweightSystem(canvas);
   lws.setContents(root);
   return canvas;
}
```

Tip: Always set the font for the root figure

```
root.setFont(parent.getFont());
```

so that each `Label`'s preferred size will be correctly calculated.

If you run the `main(...)` method, an empty window will appear (see Figure 2–2).

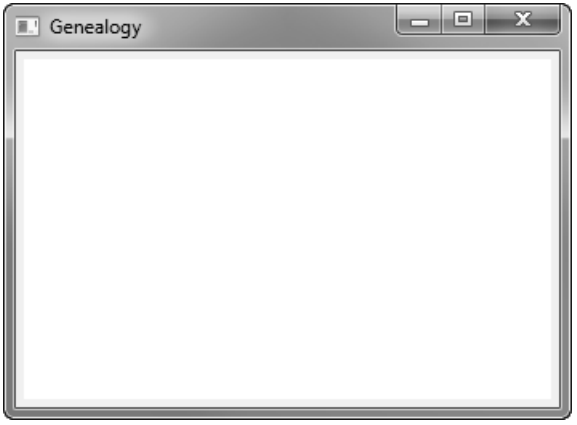


Figure 2–2 Empty Genealogy window.

Next, we want to add figures to the diagram representing a man, a woman, and their one child. Add the following to the `createDiagram(...)` method so that these figures are created and displayed.

```
private Canvas createDiagram(Composite parent) {
   ... existing code here ...

   // Add the father "Andy"
   IFigure andy = createPersonFigure("Andy");
   root.add(andy);
   layout.setConstraint(andy,
      new Rectangle(new Point(10, 10), andy.getPreferredSize()));

   // Add the mother "Betty"
   IFigure betty = createPersonFigure("Betty");
   root.add(betty);
   layout.setConstraint(betty,
      new Rectangle(new Point(230, 10), betty.getPreferredSize()));

   // Add the son "Carl"
   IFigure carl = createPersonFigure("Carl");
   root.add(carl);
   layout.setConstraint(carl,
      new Rectangle(new Point(120, 120), carl.getPreferredSize()));

   ... existing code here ...
}
```

Tip: Rather than adding the figure and then separately setting the layout constraint:

```
root.add(andy);
layout.setConstraint(andy,
   new Rectangle(new Point(10, 10),
   andy.getPreferredSize()));
```

combine this into a single statement using the `IFigure.add(child, constraint)` method:

```
root.add(andy,
   new Rectangle(new Point(10, 10),
   andy.getPreferredSize()));
```

Refactor the `createDiagram(...)` method above to use this more compact form, and inline the layout as we do not need to refer to it.

```
root.setLayoutManager(new XYLayout());
```

The `createDiagram(...)` method now calls a new `createPersonFigure(...)` method to do the work of instantiating and initializing the figure representing a person. This person figure contains a nested `Label` figure to display the person's name (see Section 4.3.2 on page 35 for more on nested figures).

```
private IFigure createPersonFigure(String name) {
   RectangleFigure rectangleFigure = new RectangleFigure();
   rectangleFigure.setBackgroundColor(ColorConstants.lightGray);
   rectangleFigure.setLayoutManager(new ToolbarLayout());
   rectangleFigure.setPreferredSize(100, 100);
   rectangleFigure.add(new Label(name));
   return rectangleFigure;
}
```

Now, when the `main(...)` method is run, the following window appears (see Figure 2–3).

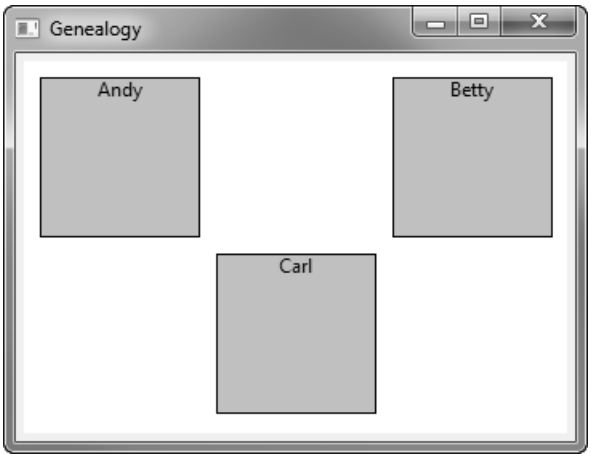


Figure 2–3 Genealogy window with three people.

Next, add more code to the `createDiagram(...)` method to create a "marriage" figure representing the relationship among the three people. This additional code calls a new `createMarriageFigure(...)` method to instantiate and initialize the marriage figure. This marriage figure is displayed using a `PolygonShape` (see Section 4.2 on page 29 for more about shapes) in the form of a diamond.

```
private Canvas createDiagram(Composite parent) {

   ... existing figure creation for people here ...

   IFigure marriage = createMarriageFigure();
   root.add(marriage,
      new Rectangle(new Point(145, 35),
      marriage.getPreferredSize()));

   ... prior code here ...
}
```

```
private IFigure createMarriageFigure() {
   Rectangle r = new Rectangle(0, 0, 50, 50);
   PolygonShape polygonShape = new PolygonShape();
   polygonShape.setStart(r.getTop());
   polygonShape.addPoint(r.getTop());
   polygonShape.addPoint(r.getLeft());
   polygonShape.addPoint(r.getBottom());
   polygonShape.addPoint(r.getRight());
   polygonShape.addPoint(r.getTop());
   polygonShape.setEnd(r.getTop());
   polygonShape.setFill(true);
   polygonShape.setBackgroundColor(ColorConstants.lightGray);
   polygonShape.setPreferredSize(r.getSize());
   return polygonShape;
}
```

Now the marriage figure is displayed when the `main(...)` method is run (see Figure 2–4).

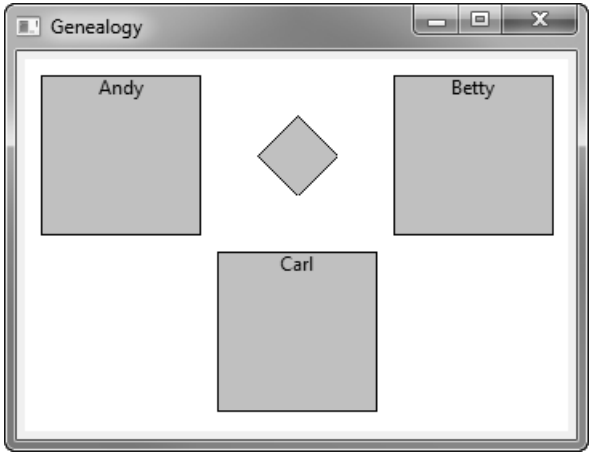


Figure 2–4 Genealogy window showing marriage figure.

Finally, connect each of the people to the marriage (see Figure 2–5), showing their relationship to one another by modifying the `createDiagram(...)` method again. This is accomplished by calling a `connect(...)` method to create the line connecting the center of one figure to the center of another (see Chapter 6 on page 69 for more about connections).

```
private Canvas createDiagram(Composite parent) {

   ... existing figure creation for marriage here ...

   root.add(connect(andy, marriage));
   root.add(connect(betty, marriage));
   root.add(connect(carl, marriage));

   ... prior code here ...
}

private Connection connect(IFigure figure1, IFigure figure2) {
   PolylineConnection connection = new PolylineConnection();
   connection.setSourceAnchor(new ChopboxAnchor(figure1));
   connection.setTargetAnchor(new ChopboxAnchor(figure2));
   return connection;
}
```

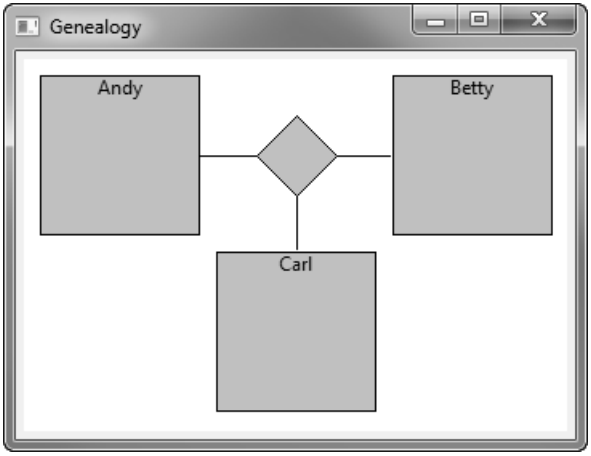


Figure 2–5 Genealogy window showing connections.

2.4 Draw2D View

The above example diagram can also be displayed in a view that is part of an Eclipse RCP application (see Chapter 7 in the *Eclipse Plug-ins* book for more about views). Start by adding the following extension to the plugin.xml:

```
<extension point="org.eclipse.ui.views">
  <category
    id="com.qualityeclipse.gef"
    name="GEF Book">
  </category>
  <view
    category="com.qualityeclipse.gef"
    class="com.qualityeclipse.genealogy.view.GenealogyView"
    id="com.qualityeclipse.genealogy.view"
    name="Genealogy"
    restorable="true">
  </view>
</extension>
```

Now modify the `GenealogyView` class to be a subclass of `org.eclipse.ui.part.ViewPart`, and add the following methods:

```
package com.qualityeclipse.genealogy.view;

... existing imports ...
import org.eclipse.ui.part.ViewPart;

public class GenealogyView extends ViewPart
{
   public void createPartControl(Composite parent) {
      createDiagram(parent);
   }
   public void setFocus() {
   }

   ... existing methods ...
}
```

When you launch the runtime workbench and open the **Genealogy** view, you'll see something like this (see Figure 2–6).

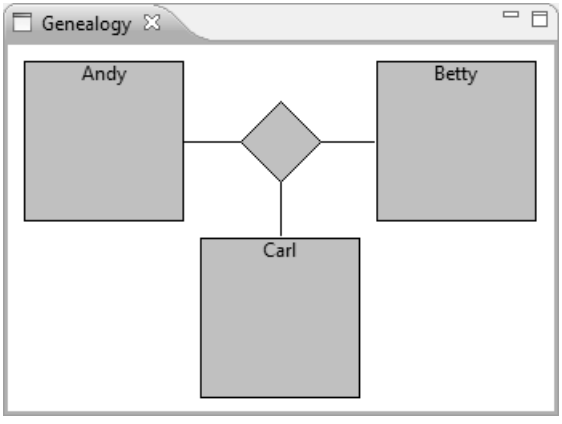


Figure 2–6 The Genealogy view.

Tip: When developing a Draw2D view with figures that don't have dependencies on the Eclipse RCP framework, add a `main(...)` method to your `ViewPart` as shown in Section 2.3 on page 9 so that you can quickly test your diagram in a shell rather than launching the entire Eclipse RCP application.

2.5 Draw2D Events

We would like the user to be able to drag the figures around the diagram. To accomplish this, we create a new Draw2D event listener to process mouse events, move figures, and update the diagram. Start by creating a new `FigureMover` class that implements the Draw2D `MouseListener` and `MouseMotionListener` interfaces. Add a constructor that hooks the listener to the specified figure and a concrete method that does nothing for each method specified in the interfaces.

```
package com.qualityeclipse.genealogy.listener;

import org.eclipse.draw2d.*;
import org.eclipse.draw2d.geometry.*;

public class FigureMover
   implements MouseListener, MouseMotionListener
{
   private final IFigure figure;

   public FigureMover(IFigure figure) {
      this.figure = figure;
      figure.addMouseListener(this);
      figure.addMouseMotionListener(this);
   }

   ... stub methods here ...
}
```

When the user presses the mouse button, we need to record the location where the mouse down occurred by adding a field and implementing the `mousePressed(...)` method. In addition, this method must mark the event as "consumed" so that the Draw2D event dispatcher will send all mouse events to this listener's figure until the mouse button is released.

```
private Point location;
public void mousePressed(MouseEvent event) {
   location = event.getLocation();
   event.consume();
}
```

As the user moves the mouse around with the mouse button held down, we need to move the figure in the same direction and distance. The `mouseDragged(...)` method calculates the distance moved, moves the figure, and marks the event as consumed. To move the figure, we must update both the figure's bounding box and the layout information. Both the figure's original location and new location must be marked as "dirty" so that the update manager will redraw the diagram appropriately. The `getBounds()` method returns the actual rectangle object used by the figure to remember its bounds, so we cannot modify that object. Instead, we call `getCopy()` before calling `translate(...)` to prevent any undesired side effects.

```
public void mouseDragged(MouseEvent event) {
   if (location == null)
      return;
   Point newLocation = event.getLocation();
   if (newLocation == null)
      return;
   Dimension offset = newLocation.getDifference(location);
   if (offset.width == 0 && offset.height == 0)
      return;
   location = newLocation;

   UpdateManager updateMgr = figure.getUpdateManager();
   LayoutManager layoutMgr = figure.getParent().getLayoutManager();
   Rectangle bounds = figure.getBounds();
   updateMgr.addDirtyRegion(figure.getParent(), bounds);
   bounds = bounds.getCopy().translate(offset.width, offset.height);
   layoutMgr.setConstraint(figure, bounds);
   figure.translate(offset.width, offset.height);
   updateMgr.addDirtyRegion(figure.getParent(), bounds);
   event.consume();
}
```

Tip: To prevent undesired side effects, call `getCopy()`, then modify the copy rather than modifying the original rectangle.

When the mouse button is released, we clear the cached location and mark the event as consumed.

```
public void mouseReleased(MouseEvent event) {
   if (location == null)
      return;
   location = null;
   event.consume();
}
```

Finally, modify the `GenealogyView` class to import the `FigureMover` class and hook the listeners to each person figure and the marriage figure by modifying the `createPerson(...)` and `createMarriage()` methods. Once these steps are complete, the figures can be dragged around the window (see Figure 2–7). For more information on how the framework determines where figures are for the mouse events, see Chapter 7 on page 91.

```
private RectangleFigure createPersonFigure(String name) {

   ... existing code here ...

   new FigureMover(rectangleFigure);
   return rectangleFigure;
}

private PolygonShape createMarriageFigure() {

   ... existing code here ...

   new FigureMover(polygonShape);
   return polygonShape;
}
```

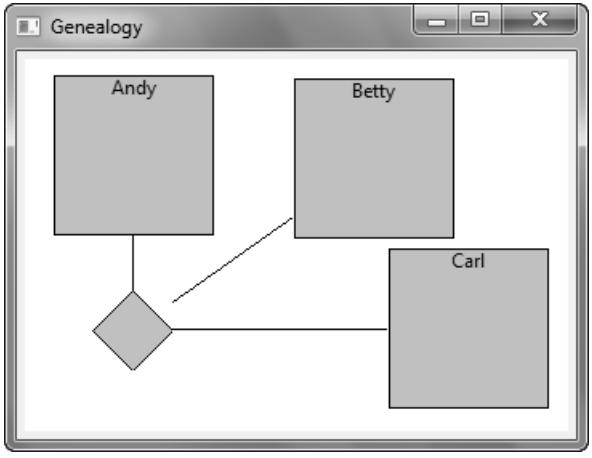


Figure 2–7 Genealogy view showing dragging of figures.

Implementing listeners such as these is useful when providing user interaction with pure Draw2D diagrams, but much of this functionality, such as dragging figures around a diagram, is already provided by the higher-level GEF framework.

2.6 Book Samples

Source code for each chapter can be downloaded and compiled into Eclipse for the reader to review, run, and modify. Go to www.qualityeclipse.com and click **Book Samples**, or go directly to the Quality Eclipse update site www.qualityeclipse.com/update to download the **QualityEclipse Book Samples** view into Eclipse. Once the samples are installed, open the view by selecting **Eclipse > QualityEclipse Book Samples**.

The view can be used to download the content for each chapter and compare the workspace content to the content in each chapter.

2.7 Summary

This chapter quickly brought the reader through a simple Draw2D example which includes a few figures that can be dragged and dropped on the canvas. The following chapters will walk through Draw2D content in more detail. All source code covered in this book can be downloaded from www.qualityeclipse.com.

References

Chapter source (see Section 2.6 on page 20).

Clayberg, Eric, and Dan Rubel, *Eclipse Plug-ins, Third Edition*. Addison-Wesley, Boston, 2009.

GEF and Draw2D Plug-in Developer Guide, Eclipse Documentation (see http://help.eclipse.org/).

Lee, Daniel, *Display a UML Diagram using Draw2D*, August 2005 (see www.eclipse.org/articles).

INDEX

F

G

M

N

Z