# eclipse

# echips Modeling Project A Domain-Specific Language (DSL) Toolkit

Richard C. Gronback

SERIES EDITORS ► Erich Gamma • Lee Nackman • John Wiegand

Many of the designations used by manufacturers and sellers to distinguish their products are claimed as trademarks. Where those designations appear in this book, and the publisher was aware of a trademark claim, the designations have been printed with initial capital letters or in all capitals.

The author and publisher have taken care in the preparation of this book but make no expressed or implied warranty of any kind and assume no responsibility for errors or omissions. No liability is assumed for incidental or consequential damages in connection with or arising out of the use of the information or programs contained herein.

The publisher offers excellent discounts on this book when ordered in quantity for bulk purchases or special sales, which may include electronic versions and/or custom covers and content particular to your business, training goals, marketing focus, and branding interests. For more information, please contact:

U.S. Corporate and Government Sales (800) 382-3419 corpsales@pearsontechgroup.com

For sales outside the United States, please contact:

International Sales international@pearson.com

Visit us on the Web: informit.com/aw

Library of Congress Cataloging-in-Publication Data:

Gronback, Richard C. Eclipse modeling project : a domain-specific language (DSL) toolkit / Richard C. Gronback. p. cm.

ISBN 0-321-53407-7 (pbk. : alk. paper) 1. Computer software—Development. 2. Eclipse (Electronic resource) 3. Programming languages (Electronic computers) I. Title.

QA76.76.D47G785 2009 005.1—dc22

#### 2008050813

Copyright © 2009 Pearson Education, Inc.

All rights reserved. Printed in the United States of America. This publication is protected by copyright, and permission must be obtained from the publisher prior to any prohibited reproduction, storage in a retrieval system, or transmission in any form or by any means, electronic, mechanical, photocopying, recording, or likewise. For information regarding permissions, write to:

Pearson Education, Inc. Rights and Contracts Department 501 Boylston Street, Suite 900 Boston, MA 02116 Fax (617) 671 3447 ISBN-13: 978-0-321-53407-1 ISBN-10: 0-321-53407-7 Text printed in the United States on recycled paper at Courier in Stoughton, Massachusetts. First printing March 2009 Associate Publisher Mark Taub

Acquisitions Editor Greg Doench

Managing Editor Kristy Hart

Project Editor Jovana San Nicolas-Shirley

Copy Editor Krista Hansing Editorial Services, Inc.

Indexer Erika Millen

Technical Reviewer Simon Archer

David Orme Daniel Holt

Publishing Coordinator Michelle Housley

Cover Designer Sandra Schroeder

Compositor Nonie Ratcliff

# Foreword

Just like a pearl, the Eclipse Modeling Project has grown organically as layers around a central core. From the humble beginnings of the Eclipse Modeling Framework (EMF) (initially part of the Eclipse Tools Project) along with the Graphical Modeling Framework (GMF) and the Generative Modeling Tools (GMT) (both initially part of the Eclipse Technology Project), the Modeling Project coalesced to become one of Eclipse's most exciting and diverse projects. The depth and breadth of its technology is vast and even its rate of growth continues to increase. The Eclipse Modeling Project has truly become a Swiss Army knife for state-of-the-art model-driven software development.

The sheer volume of useful modeling technologies that the Eclipse Modeling Project includes makes mastering a significant portion of it a daunting task. Even determining which specific available technologies are useful for solving any particular problem is a challenge exacerbated by the fact that, as a rule, the documentation tends to lag far behind the development work. As such, this book fills a fundamentally important need in the modeling community: a coherent vision of how all this powerful technology can be best exploited to build domain specific languages (DSLs). In other words, the focus of this book is on pragmatic applications illustrated by way of concrete examples rather than on abstract modeling concepts and theories. This pragmatic focus reflects that of the Modeling Project overall—that is, a focus on building powerful frameworks that real programmers use every day. I'm sure this influential book—with its interesting examples and its excellent reference material—will become a key part of every toolsmith's technical arsenal.

EMF provides a sound basis for abstract syntax development and even includes a crude but effective XML-based concrete syntax. But, that is only the start of the journey, and the second edition of the *Eclipse Modeling Framework* book covers this basic material well. This book effectively picks up where EMF leaves off with an in-depth exploration of alternative forms of concrete syntax, particularly graphical syntax, model-to-model transformation (such as Query View Transformation Operational Mapping Language), and model-to-text transformation (such as Xpand). It rounds out the DSL picture with a holistic view of everything it takes to build a complete model-based product line.

It has been my great pleasure and honor to work closely with Richard C. Gronback as the Modeling Project coleader for the past few years. He has a keen mind and a sharp wit. This book reflects it well. I've learned a great deal from him, and I'm sure readers of this book will as well.

—Ed Merks, Ph.D. President, Macro Modeling

# Preface

## **About This Book**

This book covers a relatively new collection of technologies that focus on developing domain-specific languages (DSLs) using the Eclipse Modeling Project, offering a first look at a range of Eclipse projects that have not yet been covered in detail within this context. Although the core of these technologies has been available for several years in the Eclipse Modeling Framework (EMF), diagram definition and model transformation are emerging technologies at Eclipse. These are complemented by upcoming textual syntax development frameworks, which likely will be covered in detail in subsequent editions of this book.

This book delivers a pragmatic introduction to developing a product line using a collection of DSLs. A model-based, largely generative approach is designed to accommodate future adjustments to the source models, templates, and model transformation definitions, to provide customized solutions within the context of the product line. To illustrate this approach, this book presents a set of sample projects used to define a requirements product line.

## Audience

This book targets developers and architects who want to learn about developing DSLs using Eclipse Modeling Project technologies. It assumes a basic understanding of the Java programming language, Eclipse plug-in development, and familiarity with EMF. This book's target audience are those interested in learning about the Eclipse Graphical Modeling Framework (GMF), Model-to-Model Transformation (M2M) Query/View/Transformation Operational Mapping Language (QVT OML), and Model-to-Text Transformation (M2T) Xpand project components.

The book is divided into introductory, hands on, and reference sections. Readers who want an overview of the Eclipse Modeling Project and development of DSLs in the context of an Eclipse-based product line should read Part I, "Introduction." Readers who want to follow along in a tutorial fashion to learn how to use the projects listed earlier should read Part II, "Developing Domain-Specific Languages." Readers also can reference the sample project solutions in this section to get an overview of the techniques. Part III, "Reference," serves as a resource for readers who want a deeper understanding of Graphical Editing Framework (GEF), GMF, Xpand, and QVT OML while they are completing Part II or developing their own DSL-based projects.

Readers who want to experience the benefits of a commercial version of the technologies presented here can download the Borland Together product. There they will find enhanced domain modeling, refactoring, diagram development, transformation authoring and debugging, workflow, and generation capabilities in a well-integrated DSL Toolkit.

### Sample Code

The Modeling Amalgamation Project (Amalgam) at Eclipse holds the sample code from this book and is available as sample projects in the DSL Toolkit download. This package also includes all the prerequisites required for developing the sample applications.

Visit the Amalgam project Web site for more information on obtaining the DSL Toolkit: www.eclipse.org/modeling/amalgam.

### Feedback

The examples in this book are maintained within the Modeling Amalgamation Project at Eclipse. Feedback on their content—and, therefore, this book's content—is welcome on the project newsgroup, http://news.eclipse.modeling. amalgam. Alternatively, feel free to contact the author directly at richard. gronback@gmail.com.



# CHAPTER 3

# Developing a DSL Abstract Syntax

In this chapter, we walk through the development of a domain-specific language (DSL) using the Eclipse Modeling Framework (EMF) and supporting components. Specifically, we develop the DSL's abstract syntax using the Ecore metamodel. But first we cover some basics on what to consider when creating a DSL and the different implementation strategies you might want to employ when using EMF. Next, we provide an overview of EMF, leaving detailed information to the book [38] dedicated to this purpose. We cover some additional components of EMF and Model Development Tools (MDT) that enable you to further refine DSLs, and we develop a series of domain models for use in the sample projects.

#### DISCLAIMER

The domain models developed as samples are constructed to illustrate certain features of the associated tooling and, as such, should not necessarily be considered "best practices" in some cases.

## 3.1 DSL Considerations

Many considerations are involved in creating a DSL. Does a model already exist that is close enough? If so, can an existing model be extended, or is it fixed? Does the model need to be based on a standard? Does the DSL lend itself to graphical display and editing? Does the DSL require a textual syntax and editor? Will a

product line be built on the DSL, and perhaps others? Is the Ecore metamodel expressive enough to suit your needs for a DSL? How can you model dynamic behavior?

#### **BEST PRACTICE**

Leverage existing models, when appropriate. XML Schema Definition (XSD) and EMF are very popular technologies, and EMF can import just about any XSD, so search for existing domain models before you reinvent the wheel. Also consider publishing your domain model if you think that others might find it useful, if only as part of your application's API to aid in integration.

A key consideration is the amount of flexibility you need or will tolerate in the DSL. As you can see in the examples, sometimes a change in the domain model makes your transformation definitions much easier to write. Also, frameworks such as GMF have certain limitations—or, rather, were designed with particular use cases in mind. Your particular style of modeling might not lend itself well to graphical representation, but a few changes might allow mapping to diagram elements much easier. For example, certain mappings in Query/View/ Transformation (QVT) and template expressions can be facilitated by adding derived features or methods to the domain model. Complex queries using Object Constraint Language (OCL) (and, therefore, useful ones in QVT and Xtend) can be added to the domain model with code generated for their implementation at runtime. Having a feature available in the model will greatly simplify transformations and templates that access them.

#### TIP

Don't be afraid of modifying your domain model to make working with templates, transformations, and diagram definitions easier. Unless you're using a model that cannot be altered, the Toolsmith will appreciate being able to make certain design decisions in the domain model to suit the tooling, instead of having to create workarounds or write custom code to use the tooling with a domain model.

This is not to say that you should let the tooling influence your DSL to an extent you are not comfortable with. The question is, how do you maintain a satisfactory level of "purity" in your DSL when considering the additional complexity associated with developing and maintaining the other Model-Driven Software Development (MDSD) artifacts? In general, the more complex the metamodel (DSL) is, the more complex the transformation definitions, templates, and diagram definitions are.

A set of conventions and best practices for the definition of DSLs, transformations, and templates likely will arise, as it has for Java and other popular programming languages. With conventions and best practices comes tooling to support refactorings, static analysis, and cleanup. At this stage of the Modeling project's evolution, operations are still quite manual and even error prone. As an open source project that forms the basis for commercial products, Eclipse eventually will see more advanced features pushed down into it, thereby improving the Toolsmith experience.

### 3.2 Eclipse Modeling Framework

From the project description, EMF is "a modeling framework and code generation facility for building tools and other applications based on a structured data model." This pretty much sums it up, but there's a lot to know about EMF. I highly recommend that you first read, or at least have available, the book on EMF [38] to better understand its use in the context of this book. Alternatively, reading through the online documentation and tutorials on EMF should make its use in this book easy to follow. In other words, the examples in this book only scratch the surface of what is possible using EMF.

You can create models using EMF in many ways. You can use the provided editor (a tree with properties view) or import a set of annotated Java classes. An Ecore diagram is available from the EMFT project. If you have the XSD component installed, you can import an XSD file. If you have the Unified Modeling Language (UML) version 2 (UML2) component installed, you can import a UML2 model. If you have Graphical Modeling Framework (GMF) installed, you can use its example Ecore diagram editor. If you download and install Emfatic [42], you can work in a textual syntax and synchronize with your Ecore model. In the future, you will be able to design your own concrete textual syntax for Ecore, or any model, using the Textual Modeling Framework (TMF) project.

Regardless of the method you choose for importing or working with your domain model, you will find an .ecore file in your workspace—that is, unless you take a purely programmatic approach. If you open this file in a text editor, you will see that it is an XML Metadata Interchange (XMI) serialization of your Ecore-based model. By default, EMF enables you to edit models in a basic (generated) tree editor with a Properties view. You can easily generate a similar editor for your own model.

Before getting into more detail, let's take a look at the Ecore metamodel.

#### 3.2.1 Ecore Metamodel

The EMF book describes the Ecore metamodel in detail, but here you find a simplified diagram for reference (Figure 3-1), along with some discussion of the more relevant aspects used as we develop our DSL abstract syntax. It's a fairly simple model, which is part of its strength. In most cases, you can compensate for the lack of features in Ecore by using some of the more advanced modeling techniques, which are discussed in the EMF book. A longstanding topic of debate among EMF users is the lack of an EAssociation class, but we don't get into that here.



Figure 3-1 Ecore model

#### Annotations

Sometimes it's important to add information to a model element for documentation purposes, or to provide parameters to be considered during transformation or generation. EAnnotations provide these for all model elements in EMF. An EAnnotation has a **Source** field, which serves as a key, and a list of **References**. An EAnnotation may have zero or more **Details Entry** children, which have **Key** and **Value** properties. This simple capability of annotating models is quite flexible and turns out to be useful for many purposes, including XSD support.

Another particularly useful application of annotations is to declare OCL constraints, method bodies, and derived feature implementation, as discussed in Section 3.2.4, "Applying OCL."

#### 3.2.2 Runtime Features

The EMF runtime includes facilities for working with instances of your models. No strict dependencies exist on the Eclipse platform for the runtime and generated model and edit code, so these bundles can be used outside of the Eclipse workbench. As bundles, they can be deployed in any Equinox OSGi container, even within a server environment.

The generated code for your model has a dependency on the underlying EMF runtime components. A significant benefit is gained from the generated Application Programming Interface (API) and model implementation working with the provided runtime features. An efficient observer pattern implementation is provided to alert listeners to model change events. A generated reflective API provides an efficient means of working with models dynamically. In fact, EMF can be used in a purely dynamic fashion, requiring neither an .ecore model nor code generation. Finally, it's possible to have static registration of a dynamic package, but that's an advanced use case left to the EMF documentation.

When working with model instances, changes can be recorded in a change model that provides a reverse delta and allows for transaction support. A validation framework provides for invariant and constraint support with batch processing. The Model Transaction and Validation Framework components provide enhanced transaction and validation support, respectively.

For persistence of models, the EMF runtime provides a default XML serialization. The persistence layer is flexible, allowing for XMI, Universally Unique Identifiers (UUIDs), and even a zip option. A resource set consists of one or more resources, making it possible to persist objects in multiple files, including crosscontainment references. Proxy resolution and demand loading improve performance when working with large models across resources. Additionally, use of EMF Technology (EMFT) components Teneo and CDO allow for the persistence of models to a relational database. The generated editor for EMF models includes a multipage editor and properties view. Drag-and-drop support is provided, as is copy/paste support. A number of menu actions are available in the generated editor, including validation invocation and dynamic instance model creation. Each generated editor comes with a default creation wizard. Figure 3-2 shows an example of the editor, including a context menu showing options to create new elements, cut, copy, paste, delete, validate, and so on.



Figure 3-2 EMF-generated editor

#### 3.2.3 Code Generation

From an \*.ecore (Ecore) model, you need to produce a generator model and supply additional information required for code generation. An EMF generator model has a \*.genmodel file extension and is essentially a decorator model for a corresponding \*.ecore model. This generator model is fed to Java Emitter Templates (JETs) that are used to write Java and other files. JET is the Java Server Pages (JSP)-like technology used by default when generating text from Ecore models. This book does not cover it in detail, but a tutorial is available online [51] if you want to know more.

You can customize the existing generation output using custom templates. Furthermore, you can extract constraint, pre-/post-condition, and body implementations from OCL annotations for use in generation and invocation at runtime. This is not a native EMF capability, but you can add it using the MDT OCL component. You will use this technique in the context of the sample projects. When regenerating code, the JMerge component is used to prevent overwriting user modifications. Generated Java code is annotated with @generated javadoc style tags to identify it and distinguish it from user code. Removing the tag or adding NOT after the tag ensures that JMerge will not overwrite the modified code. Typically, using @generated NOT is preferred because it allows the Toolsmith to identify code that was generated and modified, as opposed to newly added code. Note that not all code benefits from merging. Specifically, plugin.xml and MANIFEST.MF files need to be deleted before an update can occur.

#### 3.2.4 Applying OCL

Many opportunities arise for using OCL in EMF models. Class constraints, method bodies, and derived feature implementations can all be provided using MDT OCL and EMF dynamic templates. The approach of using OCL and custom templates in this book comes from an Eclipse Corner article [44] and has been modified only slightly to conform to a similar approach taken to leverage OCL added to models in QVT, as discussed in Section 6.5.6, "Leveraging OCL in EMF Models." The templates are generic and can easily be added to any project that needs to provide OCL-based implementations in its generated model code. It is also worth noting that GMF uses OCL extensively in its models, employing an EMF Validator to maintain the integrity of its models.

To add an OCL expression to a model element, we begin by adding a normal EAnnotation. For the Source property, enter http://www.eclipse.org/ 2007/OCL. This URI allows our custom templates and QVT engine to recognize this annotation as OCL, where it can expect to find Details Entry children of type constraint, derive, or body. Note that the original article [44] used http://www.eclipse.org/ocl/examples/OCL as the URI.

Depending on the context, add the appropriate Key (EMF constraint key, derive, or body) to a child Details Entry of the EAnnotation and specify the OCL in the Value property. For invariant constraints, the OCL annotations complement the normal EMF constraint annotations by providing implementation for the validation framework to enforce constraints.

#### TIP

To test your OCL, it's helpful to use the Interactive OCL Console with a dynamic instance of your model, as discussed in Section 1.5.4, "Object Constraint Language." Be sure to select the proper model element for the expression, as well as the proper metalevel in the console.

To invoke the provided OCL at runtime, you must use custom JET templates for your domain model. The generated code retrieves the OCL statement from the model element and invokes it, evaluating the result. An alternative to this is to generate a Java implementation of the OCL during generation and avoid invoking the OCL interpreter at runtime.

The referenced article covers the details of the custom templates, so they are not covered here. Also, the templates are included in the book's sample projects and are touched upon during the development of the sample projects. For now, we take a look at just the derived feature implementation, both before and after using the OCL with a custom template approach. First, consider the default generated code for a derived reference—in this case, the rootTopics reference from the MapImpl class in our mindmap example.

```
/**
 * <!-- begin-user-doc -->
 * <!-- end-user-doc -->
 * @generated
 */
public EList<Topic> getRootTopics() {
  // TODO: implement this method to return the 'Root Topics'
  // reference list
  // Ensure that you remove @generated or mark it @generated NOT
  // The list is expected to implement
  // org.eclipse.emf.ecore.util.InternalEList and
  // org.eclipse.emf.ecore.EStructuralFeature.Setting
  // so it's likely that an appropriate subclass of
  // org.eclipse.emf.ecore.util.EcoreEList should be used.
      throw new UnsupportedOperationException();
}
```

Let's add the following OCL statement to the derived feature using the previous convention. Here we see the annotation within the mindmap.ecore model in its native XMI serialization. Note that this OCL statement could be simplified by using the parent eOpposite relationship on our Topic's subtopics reference, which was added to facilitate the diagram definition of Section 4.3.5, "Subtopic Figure."

```
<eStructuralFeatures xsi:type="ecore:EReference"
name="rootTopics" upperBound="-1" eType="#//Topic" volatile="true"
transient="true" derived="true">
<eAnnotations source="http://www.eclipse.org/2007/OCL">
<details key="derive"
value="let topics : Set(mindmap::Topic) = self.elements->
select(oclIsKindOf(mindmap::Topic))->
collect(oclAsType(mindmap::Topic))->asSet() in
```

Before regeneration, we need to make some changes in the genmodel. To allow the OCL plug-in to be added to our generated manifest dependencies, we need to add OCL\_ECORE=org.eclipse.ocl.ecore to the Model Plug-in Variables property of the genmodel root. Also, we need to set the Dynamic Templates property to true and enter the templates path (such as /org. eclipse.dsl.mindmap/templates/domain) to the Template Directory property. After we generate, we can see the following implementation in our MapImpl class.

```
private static OCLExpression<EClassifier> rootTopicsDeriveOCL;
private static final String OCL_ANNOTATION_SOURCE =
  "http://www.eclipse.org/2007/OCL";
/**
 * <!-- begin-user-doc -->
 * <!-- end-user-doc -->
 * @generated
 */
public EList<Topic> getRootTopics() {
  EStructuralFeature eFeature =
    MindmapPackage.Literals.MAP__ROOT_TOPICS;
  if (rootTopicsDeriveOCL == null) {
    Helper helper = OCL ENV.createOCLHelper();
    helper.setAttributeContext(MindmapPackage.Literals.MAP, eFeature);
    EAnnotation ocl = eFeature.getEAnnotation(OCL_ANNOTATION_SOURCE);
    String derive = (String) ocl.getDetails().get("derive");
    try {
      rootTopicsDeriveOCL = helper.createQuery(derive);
    } catch (ParserException e) {
      throw new UnsupportedOperationException(e.getLocalizedMessage());
    }
  }
  Query<EClassifier, ?, ?> query =
    OCL_ENV.createQuery(rootTopicsDeriveOCL);
  @SuppressWarnings("unchecked")
  Collection<Topic> result = (Collection<Topic>) query.evaluate(this);
  return new EcoreEList.UnmodifiableEList<Topic>(this, eFeature,
    result.size(), result.toArray());
}
```

The generated code checks to see if the OCLExpression for this derivation has been created already; if not, it initializes it by retrieving the statement from the EAnnotation and its detail with key derive. Then the expression is evaluated and the list of Topic elements is returned.

As mentioned in the article, some improvements could be made to this approach, but it illustrates the usefulness of adding OCL statements to your EMF models. It's not hard to imagine how a significant portion of an application could be generated from a model adorned with OCL for invariant constraints, method bodies, and derived features. In GMF, we can see how OCL is used to augment the diagram-mapping model to provide for constraints, feature initialization, audit definition, and model metric definition.

#### **BEST PRACTICE**

Adding constraints and validation is essential in model-driven software development. Although you can place validation code within QVT, Xpand templates, and so on, it's most useful to ensure that your model instance is well formed when created, or before moving to a model transformation.

#### 3.2.5 Dynamic Instances

A powerful feature of EMF, and one that is useful to a Toolsmith developing a new DSL, is the capability to create dynamic instances of a model. The reflective framework of EMF is leveraged to allow instantiation of a model element without generating code beforehand. This can be done from within the default Ecore editor by selecting an element and choosing the Create Dynamic Instance context menu item. The instance is stored in an XMI file within the development workspace, so the generation or launch of plug-ins is not required to test a model or, more importantly, to test Xpand templates and QVT transformations under development. This is one important distinction when comparing JET to Xpand. Dynamic instances are used in the context of our sample projects.

#### **BEST PRACTICE**

Use dynamic instance models for development as much as possible. Xpand templates, QVT transformations, and the OCL console can all work with dynamic instance models and avoid making Toolsmiths generate code and

invoke a runtime instance to test their work. GMF diagrams still require code generation to develop effectively, although generated diagrams are capable of working with dynamic instances.

Figure 3-3 is an example of a dynamic instance model for our mindmap domain model, along with the Properties view. It's similar in functionality to the generated EMF editor, although it requires the metamodel to be loaded and reflected upon, as you can see from the loaded mindmap.ecore resource file.

Mindmap.xmi 🕱 📃	Properties 🕅	
🔻 🕘 platform:/resource/org.eclipse.dsl.mindmap/model/Mindmap.xmi	Property	Value
🔻 💠 Map Test Mindmap	Description	E Another test topic
Topic A Topic	End	I ≥ 2007-06-13T00:00:00.000-0400
Topic Another Topic	Name	📧 Another Topic
Topic Yet another topic	Parent	🖅 Topic A Topic
Topic A sub-subtopic	Priority	E HICH
Relationship depend	Start	I = 2007-06-12T00:00:00.000-0400
Relationship include	Subtopics	12
Topic test		
platform:/resource/org.eclipse.dsl.mindmap/model/mindmap.ecore		
( ) ) + +	$\bigcirc$	)∢►(

Figure 3-3 Mindmap dynamic instance model

#### TIP

You can view any Ecore model using the **Sample Reflective Ecore Model Editor**, so there's little need to generate the EMF .editor plug-in. This applies to XMI dynamic instances, such as GMF-based diagrams files where both the domain and notation models are persisted in a single file. Simply right-click the file and select **Open With**  $\rightarrow$  **Other**  $\rightarrow$  **Internal Editors**  $\rightarrow$  **Sample Reflective Ecore Model Editor**.

### 3.3 Developing the Mindmap Domain Model

We develop a simple mindmap DSL and use it throughout the book to provide an example of how to use components of the Modeling project as a DSL Toolkit. This model forms the base of our fictitious Requirements Elicitation Project (REP). This is the beginning of those sections in the book that you can follow in a tutorial fashion. Solution projects are available to save time, although you should be able to begin from scratch and produce these solutions on your own. It's up to you to follow along or simply review the solutions as we proceed.

Figure 3-4 is a diagram of the basic mindmap DSL we create in this section. Not much explanation should be required here because you can easily see that a Map class serves as the container for Topics and Relationships, which both extend from MapElement. The following sections provide details on setting up a DSL project and creating this model, along with the other DSL artifacts associated with the project.



Figure 3-4 Mindmap domain model

#### 3.3.1 Project Setup

Before getting started defining our mindmap domain model, we need a new project. Although EMF and GMF provide their own project wizards, we use the DSL Project Wizard provided by the Amalgam project to hold our DSL artifacts. You can create an equivalent project by starting with a plug-in project and adding the required dependencies, natures, and builders. The DSL project is also a plug-in project, as we'll eventually want to deploy the project to facilitate revisioning and extension. Furthermore, Xpand and workflow files currently need to be located in source paths to be developed, so we need a Java project anyway. In the future, this should not be required. For our mindmap project, switch to the DSL perspective and use File  $\rightarrow$  New  $\rightarrow$  DSL Project to create a new project named org.eclipse.dsl.mindmap. The wizard creates a set of folders: /model, /diagrams, /templates, /transformations, and /workflows. Not all of these folders are required for each DSL project, but we use them for our mindmap. The wizard also adds natures and builders for QVT and Xpand/Model Workflow Engine (MWE).

#### 3.3.2 Creating the Mindmap Domain Model

As mentioned earlier, creating an Ecore model involves many starting points. If we had an existing XML Schema for our domain, we could import it and EMF would take care of serializing documents conforming to the schema. If we used the UML2 project and associated the UML2 Tools class diagram to model our domain, we could import it to create an EMF model. We begin using "classic" EMF to create our mindmap DSL from scratch.

Typically, we'd begin with File  $\rightarrow$  New  $\rightarrow$  Other  $\rightarrow$  Eclipse Modeling Framework  $\rightarrow$  Ecore Model (Ctrl+3  $\rightarrow$  Ecore Model). However, the DSL Toolkit from Amalgam provides some wizard redefinitions to facilitate DSL development and defines capability definitions to hide the original UI contributions from various Modeling projects. To create our model, we select the /model folder and use the File  $\rightarrow$  New  $\rightarrow$  Domain Model (Ctrl+3  $\rightarrow$  Domain Model) wizard, which is really just the GMF Ecore diagram wizard. Name the model and diagram files mindmap.ecore and mindmap.ecore\_diagram, respectively. Optionally, you can use the Ecore Tools component, available from the EMFT project. It provides some capabilities beyond those that the GMF example editor provides.

Before we begin to model the domain, we need to set some defaults in our mindmap Ecore model. First, right-click on the blank diagram surface and select **Show Properties View**. This shows the properties for the root package in our new Ecore model. Each Ecore model has a single root package, under which we can create a number of additional subpackages. In our case, we set the properties accordingly: name and Ns Prefix should be set to mindmap; Ns URI should be set to http://www.eclipse.org/2008/mindmap.

Using Figure 3-4, model the mindmap domain using the palette and Properties view. It's a straightforward model to implement, with only a couple noteworthy items: First, the MapElement class is abstract; second, the rootTopics relationship is derived, transient, and volatile. We implement this derived reference using OCL in Section 3.3.5, "Adding OCL."

The diagram surface has many features to explore, as discussed in Section 10.1, "Overview." You should note a few things, however, when using the Ecore diagram to create the mindmap domain model:

Aggregation links create a reference with the Containment property set to true, in contrast with Association links, which are noncontainment references.

Setting the upper bound property of a link to -1 creates a many relationship and causes the link to be displayed with the familiar 0..\* notation.

References with eOpposites are shown in Figure 3-4 as a single connection, whereas the Ecore diagram shows two separate links.

#### 3.3.3 Creating the Mindmap Generator Model

With our mindmap.ecore model complete, we can validate it and create a generator model. To validate it, open the model in the **Sample Ecore Model Editor** and right-click on the root package. Select **Validate** and confirm that no errors exist. If there are errors, correct them and continue. We look into adding validation for our mindmap model later, which leverages a similar validation framework provided for all Ecore models.

To create mindmap.genmodel, right-click the mindmap.ecore file in Explorer view and select New  $\rightarrow$  Other  $\rightarrow$  Domain-Specific Language  $\rightarrow$  Domain Generator Model (Ctrl+3  $\rightarrow$  Domain Gen). Note that the original EMF wizard is found in New  $\rightarrow$  Other  $\rightarrow$  Eclipse Modeling Framework  $\rightarrow$  EMF Model (Ctrl+3  $\rightarrow$  EMF Model). We started by selecting our mindmap.ecore model, so the wizard defaults to the same folder and provides the name we want. It also recognizes that we are importing an Ecore model, but we have to load it ourselves, curiously. We have only one root package, so we can finish the wizard and take a look at the generator model properties.

EMF generator models include several properties to consider. For our mindmap, we need to change only a couple from their default settings. In the root, change the Compliance Level from 1.4 to 5.0 (if it's not already set to 5.0) and change the Model Directory to be /org.eclipse.mindmap/src. (Note that this changes the edit, editor, and tests properties as well.) We need to manually change the Model Plug-in ID property to org.eclipse.mindmap, however. In the properties for the Mindmap root package, we need to set the Base Package property to org.eclipse to match our new plug-in namespace.

This gets us started, so we can move on to code generation. Later, we return to our mindmap model and add constraints, validation, and other enhancements.

#### 3.3.4 Generate and Run

The last thing to do is generate our mindmap plug-ins and code. Technically, we don't need to generate code at this time because we plan to leverage dynamic instances as long as we can in the development of our DSLs. However, for those new to EMF, it's worthwhile to continue with generation at this point to see how things work. This is accomplished by right-clicking the root of the mindmap. genmodel in the editor tree and selecting Generate All. This generates our model code, edit code, editor code, and test skeletons, each in their own plug-in projects. We don't need the generated editor code because a diagram provides our primary means of working with mindmap instance models. For now, we can continue by running the generated editor to verify our model and configuration.

To run our plug-ins and test the functionality of our editor, we need to be in the Plug-in Development Environment perspective to gain access to the appropriate Run option. Select  $\operatorname{Run} \to \operatorname{Open} \operatorname{Run} \operatorname{Dialog} (\operatorname{Ctrl}_{+3} \to \operatorname{Run} C)$  and create a new Eclipse Application run configuration named requirements in a directory named runtime-requirements. Figure 3-5 is an image of this dialog. Figure 3-6 shows the Arguments page with some arguments for launching on Mac OS X. We use this launch configuration throughout our development of the sample projects, hence the general requirements name.

#### TIP

If you get tired of adding arguments to your launch configurations each time you create one, navigate in the Preferences dialog to **Plug-In Development**  $\rightarrow$  **Target Platform**  $\rightarrow$  **Launching Arguments** and enter them in the field provided. These values will be copied into any new launch configuration you create.

Run this configuration to launch a new instance of Eclipse with the new plug-ins installed. We could trim the plug-in list to launch only those plug-ins we need for our application. This makes launching faster and keeps us aware of our underlying plug-in dependencies. In Chapter 8, "DSL Packaging and Deployment," we fine-tune our launch settings before creating our product configuration. In the runtime workbench, create a new project and follow New  $\rightarrow$  Example EMF Model Creation Wizards  $\rightarrow$  Mindmap Model, giving it whatever name you want and selecting Map as the Model Object. The default EMF-generated editor appears upon finish, ready for you to create new Topic and Relationship instances within the map.

You again need to open the Properties view to set model element properties and establish subtopics and relationship links. Notice that validation is available for our model instances and enforces the basic structural features defined in our model. For example, we declared 1 for the upper and lower bounds on source and target references of our Relationship class. Creating a new Relationship instance in our model and invoking the Validate menu option brings up a dialog that points out that these required features were not set. As we enhance our model further, EMF and the Validation Framework will provide additional validation, as used by GMF for diagram validation.

000	Run			
Create, manage, and run configurations Create a configuration to launch an Eclipse application.				
Image: Second state of the second	Name: requirements	g-ins Configuration Tracing Convironment Common untime-requirements only Workspace File System Variables learing lipse.platform.ide ipse.ui.ide.workbench ips		
Filter matched 26 of 26 items		Apply Revert		
0		Close Run		

Figure 3-5 Requirements launch configuration



Figure 3-6 Requirements launch configuration arguments

#### 3.3.5 Adding OCL

As you should recall, we added a derived, transient, volatile rootTopics reference in our Map class. Section 3.2.4, "Applying OCL," described the basics of adding OCL and using dynamic templates to generate implementations for invariant constraints, method bodies, and derived features. The example in that section covered the rootTopics implementation using OCL and used a set of dynamic templates that we use in this context as well. At this time, rename the default templates folder to be a templates-domain source folder in the mindmap project, and copy the templates provided in the solution into this folder. We'll have additional templates later for deployment, so we can separate them into different root folders. Each DSL project that uses OCL to refine its domain model will reuse this set of templates. Then return to Section 3.2.4 and configure the mindmap.ecore model to use OCL to implement the rootTopics feature.

We can leverage OCL in our model in additional places to provide an implementation and avoid having to modify our generated code. Let's begin by adding a method that returns the full set of subtopics for a given Topic.

#### **Finding All Subtopics**

Currently, our model has a subtopics reference on each Topic, along with a method, allSubtopics(), that is intended to return a list of all of a Topic's subtopics—that is, its subtopics, all of their subtopics, and so on. All methods declared in an Ecore model require an implementation to be provided, so we turn to OCL, where the implementation of this method is trivial, thanks to the non-standard closure iterator in MDT OCL:

```
self->closure(subtopics)
```

We need to add an EAnnotation to the method in our model with Source equal to http://www.eclipse.org/2007/OCL. A child Details Entry is added to the annotation with the previous expression as its Value property and with a Key value of body. When we regenerate our model code, we can see that our implementation is provided:

```
/**
 * The parsed OCL expression for the body of the
 * '{@link #allSubtopics <em>All Subtopics</em>}' operation.
 * <!-- begin-user-doc -->
 * <!-- end-user-doc -->
 * @see #allSubtopics
 * @generated
 */
private static OCLExpression<EClassifier> allSubtopicsBodyOCL;
private static final String OCL_ANNOTATION_SOURCE =
  "http://www.eclipse.org/2007/OCL";
/**
 * <!-- begin-user-doc -->
 * <!-- end-user-doc -->
 * @generated
 * /
public EList<Topic> allSubtopics() {
  if (allSubtopicsBodyOCL == null) {
    EOperation eOperation =
      MindmapPackage.Literals.TOPIC.getEOperations().get(0);
    OCL.Helper helper = OCL_ENV.createOCLHelper();
```

}

```
helper.setOperationContext (MindmapPackage.Literals.TOPIC,
    eOperation);
  EAnnotation ocl = eOperation.getEAnnotation(OCL ANNOTATION SOURCE);
  String body = ocl.getDetails().get("body");
  trv {
    allSubtopicsBodyOCL = helper.createQuery(body);
  } catch (ParserException e) {
    throw new UnsupportedOperationException(e.getLocalizedMessage());
  }
}
Ouery<EClassifier, ?, ?> query =
  OCL_ENV.createQuery(allSubtopicsBodyOCL);
@SuppressWarnings("unchecked")
Collection<Topic> result = (Collection<Topic>) query.evaluate(this);
return new BasicEList.UnmodifiableEList<Topic>(result.size(),
  result.toArray());
```

Again, here we see the boilerplate OCL code that configures an OCLExpression if it's the first invocation, and then it invokes the expression obtained from the annotation. We leave the mindmap model at this point and move on to develop the second domain model in our product line.

## 3.4 Developing the Requirements Domain Model

In a similar fashion to our mindmap model, we create a new org.eclipse. dsl.requirements DSL project here to hold our requirements model. This forms the base of our fictitious Requirements Management Project (RMP). We create the new requirements.ecore model using the Domain Model Wizard and GMF Ecore diagram, and we complete it to match the diagram and description of Figure 3-7.

Basically, a model contains a collection of RequirementGroups, which contain a number of children groups and Requirements. Requirements have child references and contain Version and optional Comment elements. A number of enumerations for Priority, State, Type, and Resolution are also in the model. A Requirement can also have a number of dependent requirements, which become the basis for our dependency diagram. Note that the author attributes are simple strings. We could create a Team model and reference these elements to assign to our requirements and comments. We also could have a separate Discussion model to use here and in our mindmap, as a topic might have an associated discussion thread. Many possibilities exist, but for the purposes of our sample application, we keep it simple.



Figure 3-7 Requirements domain model

#### 3.4.1 Requirements Generator Model

We create a requirements.genmodel in the usual manner, using the new **Domain Generator Model** (Ctrl+3  $\rightarrow$  Domain Gen) wizard and selecting our requirements.ecore model as the input. We'll make some adjustments to this genmodel and to the generated Edit code because we intend to use the generated EMF editor as part of our solution.

For the display string of a requirement in the editor selection tree, we want to have it be id (major.minor.service):title, where major, minor, and service are from the contained Version element. We'll be using the Properties view to edit the details of the requirement, so we'll have plenty of horizontal space to use in the editor, allowing even longer requirement titles to fit. Another option is to navigate using the Project Explorer view, but this is narrow and does not allow for much information display. Furthermore, we'll have a second tab in the editor to display a requirements dependency diagram, which will also require a bit of editor space. To accomplish the task, we'll select the requirement element in the genmodel and change its Edit Label Feature to be our id:EString attribute. Unfortunately, we cannot set two attributes to display for the label, as we can for GMF diagrams. This means we have to modify the generated code. Before generation, we need to check the other properties and make changes accordingly. As with the mindmap and other models, we want to generate our model, edit, and editor code to their own projects, so we can change the **Model Plug-in ID** and **Model Directory** properties to be org.eclipse.requirements.model. We generate the three plug-ins and open the org.eclipse. requirements.provider.RequirementItemProvider class from our Edit plug-in. Modify the getText() method as seen next. Note that if we wanted to preserve the generated method to allow the label feature of the generator model to have an effect, we could use the getTextGen() method approach, as described in the EMF documentation.

```
/**
 * This returns the label text for the adapted class.
 * Modified to show id (major.minor.service) : title
 * @generated NOT
 */
@Override
public String getText(Object object) {
  StringBuilder sb = new StringBuilder();
  sb.append(((Requirement)object).getId());
  sb.append(" (");
  Version version = ((Requirement)object).getVersion();
    if (version != null) {
      sb.append(((Requirement)object).getVersion().getMajor());
      sb.append(".");
      sb.append(((Requirement)object).getVersion().getMinor());
      sb.append(".");
      sb.append(((Requirement)object).getVersion().getService());
    } else {
      sb.append("0.0.0");
    }
    sb.append(") : ");
    sb.append(((Requirement)object).getTitle());
    String label = sb.toString();
    return label == null || label.length() == 0 ?
      getString("_UI_Requirement_type") : label;
}
```

We've eliminated the redundant Requirement prefix from our label because we're using a custom icon to distinguish Requirements from Requirement Groups, Comments, and so on. For our RequirementGroup element, we can similarly modify the getText() method to display only the name attribute; we can modify the Comment element to display created, author, and subject.

### 3.5 Developing the Scenario Domain Model

Because we're basing the notation for our Scenario diagram on the Business Process Modeling Notation (BPMN) standard, we could simply use its description of the underlying domain model and semantics to develop our DSL. A better approach would have been to find an XSD for BPMN and simply import it into EMF. Unfortunately, no such schema is published with the specification even worse, a new specification from the OMG, the Business Process Definition Metamodel (BPDM), is slated to provide a domain model for BPMN2. Also unfortunate is that this specification has no serialized format that we can use and is overly complicated for our Scenario DSL. This leaves us to create our own scenario model.

In a new org.eclipse.dsl.scenario project, we can create our scenario.ecore model as the base of our fictitious Requirements Scenario Project (RSP) project. Figure 3-8 is the model to create using our Ecore diagram.



Figure 3-8 Scenario domain model

Elements of a scenario model are maintained in the Process class, which itself extends Element. A Connection maintains target and source references for Elements that are connected in Sequence or Message flows. An Association also connects elements. Elements come in a variety of types, including Tasks, Events, DataObjects, and Gateways. These elements all map in a straightforward manner to notation elements because the model is inherently graphical in nature. The model is actually similar to the description provided in the BPMN specification, although it is a subset.

## 3.6 Developing the Business Domain Model

Plenty of options exist for developing the domain model that will form the base of our fictitious Business Domain Modeling (BDM) project. We want something less complicated than the Unified Modeling Language (UML) model for structural class modeling, but something expressive enough to generate code either directly or through an intermediate model. Also, the model should constrain the user to the supported methodology and domain of business models. For the purposes of this book, the four archetypes described for business domain modeling in *Java Modeling in Color with UML* [46] seem like a good option. Figure 3-9 is a partial image of the Domain-Neutral Component (DNC) model, created with the editor we develop in Section 4.6, "Developing the Color Modeling Diagram." Of course, a black-and-white rendering of a color modeling diagram in the printed form of this book is not very compelling.



Figure 3-9 Color archetypes

Basically, a set of archetypes is used to model moment-interval, role, party, place, thing, and description elements and their relationships. The relationships and several described patterns of interaction are provided in the book, which we want to facilitate in our modeling environment. First, however, we need an underlying model (a DSL).

This DSL is strongly rooted in a general object-oriented DSL, so we begin with just that. Figure 3-10 is an occore.ecore model that we extend to add our

archetypes and other DNC elements. Why begin with a general object-oriented DSL? Well, we might decide to use this model as the basis for another DSL in the future. Why not simply extend Ecore itself, you might ask? Well, it contains elements that we really don't need or want, leaving us with all those *E*-prefixed elements and their properties. Besides, it's straightforward to develop our own object-oriented DSL. We can use the Java EMF Model (JEM) as a transformation target, giving us a chance to see what a model that extends Ecore is like.

Adventurous types can create a new org.eclipse.dsl.oocore DSL project and create the oocore.ecore model, as we have done previously. Complete the model using Figure 3-10 as a reference. Otherwise, simply copy the oocore.ecore domain model from the solutions into your project. Finally, create a new org.eclipse.dsl.dnc DSL project to hold our dnc.ecore model that will extend our core model.



Figure 3-10 Object-oriented core domain model

With our base model complete, we can create our dnc.ecore model. To reference our oocore.ecore model in our newly created dnc.ecore model, we need to use the EMF Load Resource context menu in the default EMF editor. Fortunately, the dialog that appears now contains options to Browse Registered Packages, Browse File System, and Browse Workspace. At one time, you needed to enter platform: / URIs into the field to load a registered package. In our case, the oocore.ecore model is easily found in our workspace.

In creating our DNC model (Figure 3-11), several options exist, as always. You've seen that using an enumeration to define the type is one solution, as was used in the Mindmap domain model's Relationship class and Type enum. This approach has some drawbacks, including the loss of polymorphic behavior in our templates and transformation scripts. To illustrate the differences, let's go ahead and create an Archetype abstract class that extends our oocore::Class class. Each of our archetypes will extend the Archetype class. We also add an Association class that extends oocore::Reference and add a property to signify aggregation. Although it is not a true Association class in the UML sense, it aids us in developing our diagram and hiding some complexities of the underlying model to the Practitioner. As we develop the diagram and other DSL artifacts, we'll revisit this model and refine it as necessary, potentially pulling up some functionality into the domain model to aid in our color modeling and model transformations.



Figure 3-11 Domain-neutral component domain model

## 3.7 Summary

This chapter explored the capabilities of EMF as the means of describing our DSL abstract syntax. Although we leave the details of EMF to its own book, we covered enough to get started developing our sample projects. The benefits of leveraging a common underlying metamodel and generation capabilities should become clear as we continue to develop the DSL projects.

At this point, we have starter domain models for our fictitious ERP, plus a reference to a fifth oocore DSL. We now move on to describing how to create graphical concrete syntax for those we want to provide diagrams for, understanding that we will most likely revisit and enhance the EMF models we created in this chapter.

# Index

# **Symbols**

:= (assignment operator), 568 « » (guillemets), 606 += (assignment operator), 568

# Α

absolute coordinates (Draw2d), 323-324 abstract syntax definition of, 4 developing with EMF (Eclipse Modeling Framework) Model Compare, 11 Model Query, 11 Model Search, 11 Model Transaction, 10 Model Validation, 10 persistence alternatives, 11 AbstractCommand class, 463 AbstractEditPart (GEF), 325 AbstractProviderConfiguration class, 378 AbstractTransactionalCommand, 464 access keyword, 553 accessorCall() function, 219 ActionFilterProvider extension-point, 424-426 ActionFilterService, 423-426 actions. See also specific actions selection actions, 342 subtopic actions, 90-93 for user interface plug-ins, 307-310 activate() method, 336 add() operation, 582

addChild() method, 334-335 addChildVisual() method, 335 addFields() method, 206 addFixedChild() method, 166 addNotationalListeners() method, 211, 546 addNotify() method, 335 addSuffixNumber() operation, 593 AdviceContent operations, 627 AdviceContent properties, 626 Aggregation Link Mapping (color modeling diagram), 198-200 aliases. See shortcuts Align Bottom toolbar element, 356 Align Center toolbar element, 355 Align Left toolbar element, 355 Align Middle toolbar element, 356 Align Right toolbar element, 355 Align Top toolbar element, 355 AlignmentAction, 349 All Compartments toolbar element, 356 allSubobjects() operation, 579 allSubobjectsOfKind() operation, 579 allSubobjectsOfType() operation, 579 Amalgam, 15 Amalgamation project, 20 anchors, 94-96, 323 annotations annotation figure definition (color modeling diagram), 188-189 Annotation Mapping (color modeling diagram), 195 in Ecore metamodel, 33

Ant. 649 Ant tasks, OML and, 243-244 Appearance properties (diagrams), 358-359 AppearancePreferencePage class, 206 AppearancePreferencePage.xpt template, 216-217 Apply Appearance Property toolbar element, 355 archetype figure definition (color modeling diagram), 182-184 AROUND statement (Xpand), 617-619 Arrange All toolbar element, 355 Arrange Selected toolbar element, 355 artifacts (DSL Toolkit) abstract elements, 17 Practitioner, 18-19 Toolsmith artifacts, 17-18 artwork, adding to products, 304-305 asBoolean() operation, 591 asFloat() operation, 592 asInteger() operation, 592 asList() operation, 583 asOrderedTuple() operation, 578 aspects (Xpand), 647-648 assert expression, 576-577 assignment operator (:=), 568 assignment operator (+=), 568 asTransformation() operation, 582 ATL (Atlas Transformation Language), 231 AttributeParserProvider class, 221-224 audits audits and metrics model, 532-533 for mindmap diagram, 97-103 Auto Size toolbar element, 356

## B

Background Color property (figures), 508 BDM (Business Domain Modeling), 22 bending connections (GEF), 351-352 Boolean operations, 622 Border Layout, 513 Border property (figures), 508 borders Border Layout, 513 Compound Border, 510 Custom Border, 510 Line Border, 510 Margin Border, 510 Both command (Diagram menu), 361 BPDM (Business Process Definition Metamodel), 50, 665 BPEL, mapping with BPMN, 231 BPMN (Business Process Modeling Notation), 50, 665 BPMN2, 14 mapping with BPEL, 231 Bring Forward command (Order menu), 361 Bring to Front command (Order menu), 361 business domain model, 51-53 Business Domain Modeling (BDM), 22 Business Process Modeling Notation. See BPMN

# С

cached extensions (Xtend), 639 canExecute() method, 483 canonical containers, 547 Canvas, 505-506 Canvas Mapping, 520 mindmap diagram, 71 requirements diagram, 108 casting, 635 chain expression, 634-635 Checks language, 228 ChopboxAnchor (Draw2d), 323 class2columns() guery, 600 classes. See specific classes clipboardSupport extension-point, 457-458 clipping in Draw2d, 321 clone() operation, 579 Coad, Peter, 57 code generation in EMF, 34-35 for mindmap domain model, 43-45 code modification, GMF customization options, 65 collapsible compartments, 518 collect() operation, 564, 631-632

collections, 630 operations, 623-624 properties, 623 collectselect imperative iterate expression, 567 color in graphical notations, 57 color modeling diagram, 22, 181-182 annotation figure definition, 188-189 archetype figure definition, 182-184 color preferences, 205-220 addNotationalListeners() method, 211 DiagramAppearancePreferencePage class, 207-208 getPreferenceColor() method, 211 initializeDefaultPreferences() method, 209 initPreferenceStoreListener() method, 211 Messages class, 209 messages.properties file, 210 NodeEditPart.xpt template file, 212, 215-216 preference page templates, 216-220 PreferencePropertyChangeListener class, 211, 214 setForegroundColor() method, 211 setForegroundColor() method, overriding, 205 toPreferenceConstant() function, 213 updateArchetypeColor() method, 211 custom parsers, 220 AttributeParserProvider class, 221-224 OperationParserProvider class, 224-225 generalization link definition, 186-187 generation, 201 gradient figures, 201-205 mapping definition Aggregation Link Mapping, 198-200 Annotation Mapping, 195 DNC Association Mapping, 195-197 DNC Canvas Mapping, 190 Generalization Mapping, 200-201 Moment-Interval Archetype Mapping, 191-192 Package Mapping, 194 Role Archetype Mapping, 193

package figure definition, 185-186 tooling definition, 189 CombinedTemplateCreationEntry, 344 combining layouts, 515 comma-separated values (CSV) files, generating, 281-282 command infrastructure (GMF runtime), 462-464 commands. See specific commands comments OML, 593 Xtend, 637 Compartments, 518 compliance, specification compliance, 668 COMPONENT\_ROLE key, 343 Compound Border, 510 concrete syntax development, 4, 12 GMF (Graphical Modeling Framework). See GMF runtime; GMF Tooling TMF (Textual Modeling Framework), 12 configuration properties (transformations), 555 configureGraphicalViewer() method, 327 connection bend interactions (GEF), 351-352 connection creation interactions (GEF), 349-350 connection edit interactions (GEF), 350-351 connection EditParts, 330-331 Connection figures, creating for scenario diagram, 155-158 connection handles (diagrams), 364-365 connection navigation, 657-658 ConnectionBendpointTracker, 352 ConnectionCreationTool (GEF), 349 ConnectionEndpointEditPolicy, 351 ConnectionRouter (Draw2d), 322 Connections, 517 bending, 351-352 creating, 349-350, 479-484 in Draw2d, 322-323 editing, 350-351 constraints, node constraints, 526 ContainerEditPolicy, 348 containers, canonical, 547 Contains Shortcuts To property (Gen Diagram), 536 context menus, 363-364

contributing to GMF (Graphical Modeling Framework), 546 contributionItemProvider extension-point, 409-418 ContributionItemService contributionItemProvider extension-point, 409-418 overview, 408 coordinate systems in Draw2d, 323-324 copy() operation, 582 Copyright Text property (Gen Editor Generator), 534 Core language (QVT), 231, 549 Core properties (diagrams), 358 create expression, 635 createDecorators() method, 498 createDefaultEditPolicies() method, 385, 474 createEditPart() method, 334 createEditPolicies() method, 333-335, 385, 481 createEmptyDocument() method, 126 createEmptyModel() operation, 582 createFigure() method, 328 createGraphicEditPart() method, 382 createMainFigure() method, 114 createNodePlate() method, 516 createPages() method, 129-130 createPartControl() method, 139 creation interactions (GEF) CreateRequests consuming, 345 producing, 344 creation sequence, 345-347 Creation Wizard, 137-138 Creation Wizard Category ID property (Gen Diagram), 538 CSV (comma-separated values) files, generating, 281-282 Custom Behavior element, 539 Custom Border, 510 custom EditPolicy, 493-497 custom figures, 89-90 custom layout, 82-84 custom parsers (color modeling diagram), 220 AttributeParserProvider class, 221-224 OperationParserProvider class, 224-225

Custom Property Tab (requirements diagram), 140 custom style, 490-493 customization options (GMF) code modification, 65 decorators, 66, 497-500 dynamic templates, 65-66 extension-points, 65 model extensions, 66

## D

dashboard view (GMF), 66-67 Data Object figure, creating for scenario diagram, 158-159 Data Object Mapping (scenario diagram), 165 database persistence, 177-181 dataType2columns() query, 600 dataType2primaryKeyColumns() mapping, 603-604 deactivate() method, 336-337 deactivating EditParts, 336-337 declaring extension-points, 373-375 decorateView() method, 488 DecorationService decoratorProvider extension-point, 392-396 overview, 392 decoratorProvider extension-point, 392-396 decorators, custom, 66, 497-500 deepclone() operation, 580 Default Size (diagram elements), 516 DEFINE statement (Xpand), 607-608 definition operations, 627 definition properties, 627 DelegatingLayout (Draw2d), 323 delete interactions (GEF), 342-343 DeleteAction, 342 dependencies (GEF), 318 deploying DSLs, 303 artwork, 304-305 generation models, 310-311 requirements.product, 312 source deployment, 312-313 user interface plug-ins, 305-310

designing graphical notations, 56-57 developing DSLs, 6 development workflow (DSL Toolkit), 18-20 DI (Diagram Interchange), 664 diagram editor, 467-468 cycling between diagram editor and palette, 653-654 Diagram File Extension property (Gen Editor Generator), 534 Diagram Interchange (DI), 664 Diagram Label, 517 Diagram menu commands, 361 DiagramAppearancePreferencePage class, 207-208 DiagramDecoratorProvider, 396 diagramEventBrokerProvider extension-point, 441-443 DiagramEventBrokerService extension-point, 441-443 DiagramRootEditPart (GEF), 326 diagrams. See also specific diagrams adding EditParts to, 334-336 connection handles, 364-365 constructing from scratch compared to generated diagrams, 485-486 connections, 479-484 diagram editor, 467-468 domain model, 468-469 EditPart definition, 473-475 New Diagram Wizard, 475-476 palette definition, 470-471 properties view, configuring, 476-479 view definition, 471-473 context menus, 363-364 diagram elements Compartment, 518 Connection, 517 Diagram Label, 517 Node, 516-517 overview, 516 Diagram menu, 361 extending custom decorator, 497-500 custom EditPolicy, 493-497 custom style, 490-493

scenario diagram custom view and edit providers, 486-489 figures changing appearance based on preference changes, 546 changing dynamically, 546 changing properties of, 547 making nonresizable, 547 viewing elements on diagrams, 547 including in multipage editors, 548 navigation, 655 Order menu, 361 Outline view, 366 Palette view, 360 partitioning, 171, 522 Requirements diagram partition, 172-177 subprocess partitions, 171-172 pop-up bars, 365 preferences, 367-368 properties Appearance properties, 358-359 Core properties, 358 Rulers & Grid properties, 356-358 toolbar elements, 354-356 View menu, 361-363 dialogs. See specific dialogs disjunction in mapping operations, 562 disjuncts keyword, 562 DNC Association Mapping (color modeling diagram), 195-197 DNC Canvas Mapping (color modeling diagram), 190 documentProviders extension-point, 453-455 doExecuteWithResult() method, 483 Domain File Extension property (Gen Editor Generator), 535 Domain Gen Model property (Gen Editor Generator), 535 domain models, 108, 468-469. See also metamodels best practices, 30 business domain models, 264-276 business DSL, 51-53 creating DSLs, 29-31 dynamic instance models, 262

dynamic instances, 38-39 Ecore models, 39 importing, 31 mindmap DSL, 39-40 code generation, 43-45 creating, 41-42 generator model, creating, 42 OCL, adding, 45-47 project setup, 40-41 running, 43-45 transformation to requirements domain model, 244-251 transformation to XHTML, 251-258 OCL in. 35-38 requirements DSL, 47-49 scenario DSL, 50-51 synchronizing, 59, 76 doRun() method, 495 downloading DSL Toolkit, 20 DragDropListenerProvider extension-point, 436-438 DragDropListenerService, 436-438 DragTracker, 341 Draw2d, 318 connections and routing, 322-323 coordinate systems, 323-324 figures, 318-319 interaction sequence, 320 layout, 322 LightweightSystem, 318-319 painting, 320-321 text. 319 UpdateManager, 318 **DSL** Instances Java DSL Instances, 24-25 mindmap and requirements DSL Instances, 23-24 scenario DSL Instances, 24-25 DSL Project Wizard, 40 DSL Toolkit abstract elements, 17 BDM (Business Domain Modeling), 22 development workflow, 18-20 downloading, 20 Java DSL Instances, 24-25

mindmap and requirements DSL Instances, 23-24 Practitioner, 18-19 Practitioner's view of sample projects, 25 **REP** (Requirements Elicitation Project), 21 **RMP** (Requirements Management Project), 21 RSP (Requirement Scenario Project), 22 scenario DSL Instances, 24-25 Toolsmith artifacts, 17-18 **DSLs** business DSL, 51-53 creating, 29-31 definition of, 5 developing, 6 graphical notation. See graphical notation mindmap DSL, 39-40 code generation, 43-45 creating, 41-42 generator model, creating, 42 OCL, adding, 45-47 project setup, 40-41 running, 43-45 transformation to requirements domain model, 244-251 transformation to XHTML, 251-258 object-oriented, 51 packaging and deployment, 303 artwork, 304-305 generation models, 310-311 requirements.product, 312 source deployment, 312-313 user interface plug-ins, 305-310 requirements DSL, 47-49 scenario DSL, 50-51 textual syntaxes overview, 227-228 TCS (Textual Concrete Syntax), 229 Xtext, 228-229 dynamic instance models, 38-39, 262 dynamic templates, 65-66 Dynamic Templates property (Gen Editor Generator), 535

## Е

EAnnotations, 33 Eclipse Packaging Project (EPP), 20 Eclipse Requirements Project (ERP) BDM (Business Domain Modeling), 22 **REP** (Requirements Elicitation Project), 21 **RMP** (Requirements Management Project), 21 RSP (Requirement Scenario Project), 22 Ecore metamodel, 32-33, 39 edit providers, 486-489 Edit Source Folder dialog, 298 editing. See GEF (Graphical Editing Framework) Editing Domain ID property (Gen Diagram), 538 EditingDomain, sharing, 123-126 EditorProvider extension-point, 427-428 editors, OVT OML editor, 240 EditorService, 427-428 EditPartFactory, 326-327, 372 editpartProvider extension-point, 382-385 EditParts (GEF), 325-326 AbstractEditPart, 325 adding to diagram, 334-336 connection EditParts, 330-331 creating, 326-329 deactivating, 336-337 definition, 473-475 DiagramRootEditPart, 326 EditPartFactory, 326-327, 372 GraphicalEditParts, 325 lifecycle, 334-337 ScalableFreeformRootEditPart, 326 TreeEditParts, 325 EditPartService, 382-385 EditPolicies, 333-334, 493-497 editpolicyProvider extension-point, 386-387 EditPolicyService, 385-387 element creation (GMF runtime), 459-461 element operations (OML) allSubobjects(), 579 allSubobjectsOfKind(), 579 allSubobjectsOfType(), 579 clone(), 579

deepclone(), 580 \_globalId(), 578 localId(), 578markedAs(), 580 markValue(), 580 metaClassName(), 578 stereotypedBy(), 580 stereotypedStrictlyBy(), 580 subobjects(), 578 subobjectsOfKind(), 579 subobjectsOfType(), 579 elements. See specific elements ElementSelectionProvider extension-point, 429-430 ElementSelectionService, 428-430 elementTypeBindings extension-point, 449-451 elementTypes extension-point, 443-449 Ellipse, 509 EMF (Eclipse Modeling Framework) code generation, 34-35 domain models, importing, 31 dynamic instances, 38-39 Ecore metamodel, 32-33 editor, integrating with requirements diagram refactoring editor, 127-130 sharing editing domain, 123-126 sharing file extension, 122-123 Model Compare, 11 Model Query, 11 Model Search, 11 Model Transaction, 10 Model Validation, 10 OCL in, 35-38 overview, 10 persistence alternatives, 11 runtime features, 33-34 EMOF (Essential MOF), 663 End Events, 152 endsWith() operation, 587 entry operations (mapping operations), 559-560 enumerationAttributes2columns() query, 602 EPP (Eclipse Packaging Project), 20 equalsIgnoreCase() operation, 589

ERP (Eclipse Requirements Project) BDM (Business Domain Modeling), 22 REP (Requirements Elicitation Project), 21 **RMP** (Requirements Management Project), 21 RSP (Requirement Scenario Project), 22 ERROR statement (Xpand), 616 escape sequences, 594 Essential MOF (EMOF), 663 Event figures, creating for scenario diagram, 152-154 Event Mapping (scenario diagram), 164 execute() method, 378 executing transformations assert expression, 576-577 log expression, 576 trace model, 575 transform() operation, 577 transformation composition, 577 execution semantics (mapping operations), 558-559 ExecutionStrategy, 376 exists() operation, 633 expand element, 646 EXPAND statement (Xpand), 608-612 exporting figures, 62 expression language (Xpand) casting, 635 chain expression, 634-635 collect() operation, 631-632 collections, 630 create expression, 635 exists() operation, 633 forAll() operation, 632 let expression, 635 literals, 628-630 reject() operation, 631 select() operation, 630 sortBy() operation, 633 switch expression, 634 ternary expression, 634 typeSelect() operation, 631 expressions. See specific expressions extending diagrams custom decorator, 497-00500 custom EditPolicy, 493-497

custom style, 490-493 scenario diagram custom view and edit providers, 486-489 extends keyword, 553 extensibility mechanisms (GMF), 372-373. See also extension-points Extensible Markup Language (XML), 664 extension keyword, 636 EXTENSION statement (Xpand), 613-614 extension-points ActionFilterProvider, 424-426 clipboardSupport, 457-458 contributionItemProvider, 409-418 declaring, 373-375 decoratorProvider, 392-396 diagramEventBrokerProvider, 441-443 documentProviders, 453-455 DragDropListenerProvider, 436-438 EditorProvider, 427-428 editpartProvider, 382-385 editpolicyProvider, 386-387 ElementSelectionProvider, 429-430 elementTypeBindings, 449-451 elementTypes, 443-449 GlobalActionHandlerProvider, 420-423 GMF customization options, 65 IconProvider, 397-399 layoutProvider, 406-408 logListeners, 452-453 MarkerNavigationProvider, 400-401 modelingAssistantProvider, 404-406 overview, 373-375 paletteProvider, 388-392 ParserProvider, 402-403 Pathmaps, 459 perspectiveExtensions, 306 perspectives, 306 preferencePages, 306 propertiesConfiguration, 453 PropertiesProvider, 431-433 PropertyModifier, 434-435 renderedImageFactory, 456-457 TransferAdapterProvider extension-point, 439-441 viewProvider, 379-382 ExtensionMap element, 643

extensions (Xtend) cached extensions, 639 extension invocation, 637-638 Java extensions, 639 private extensions, 639 Externalizer.xpt template, 216-218 extractor (TCS), 229

## F

factories, 7-8 feature initialization, 527-528 feature properties, 625 Figure Gallery, 61, 504 Border Layout, 513 Compound Border, 510 Custom Border, 510 custom figures, 89-90 Ellipse, 509 figure properties, 507-508 Flow Layout, 511-512 Grid Layout, 513-515 Label, 510 Line Border, 510 Margin Border, 510 overview, 506-507 Polygon, 509 Polyline, 509 Polyline Connection, 509 Rectangle, 509 Rounded Rectangle, 509 Stack Layout, 512 subtopic figures, 84-89 TemplatePoint, 509 XY Layout, 512 figure plug-ins, 62 FigureDescriptor, 503 FigureRef, 504 figures, 318-319 Border Layout, 513 changing appearance based on preference changes, 546 changing dynamically, 546 changing properties of, 547

in color modeling diagram annotation figure definition, 188-189 archetype figure definition, 182, 184 generalization link definition, 186-187 package figure definition, 185-186 Compound Border, 510 Custom Border, 510 Ellipse, 509 exporting, 62 Flow Layout, 511-512 Grid Layout, 513-515 Label, 510 Line Border, 510 making nonresizable, 547 Margin Border, 510 Polygon, 509 Polyline, 509 Polyline Connection, 509 properties common properties, 507-508 Flow Layout figure properties, 511 shape figure properties, 508 Rectangle, 509 Rounded Rectangle, 509 in scenario diagram, 145-146 Connection figures, 155-158 Data Object figure, 158-159 Event figures, 152-154 Gateway figure, 149-152 Subprocess figure, 147-149 Task figure, 146-147 Stack Layout, 512 TemplatePoint, 509 viewing elements on diagrams, 547 XY Layout, 512 FigureUtilities class, 501 FILE statement (Xpand), 612 Fill property (shape figures), 508 fillGradient() method, 202 fillShape() method, 202 filters in graphical notations, 57-58 find() operation, 589 findFeature() method, 474 FIRST service execution strategy, 377 firstElementOnly element, 644 firstToUpper() operation, 586

fixed anchor locations, 94-96
Flow Layout, 511-512
Font property (figures), 508
Fontoura, Marcus, 56
forAll() operation, 632
Force Single Line property (Flow Layout figure), 511
forEach loop expression, 565
FOREACH statement (Xpand), 612-613
Foreground Color property (figures), 508
format() operation, 584
forOne loop expression, 565
FORWARD service execution strategy, 377
Frankel, David, 4

# G

Gateway figure, creating for scenario diagram, 149-152 Gateway Mapping (scenario diagram), 163-164 GEF (Graphical Editing Framework) Commands, 332 connection interactions connection bend interactions, 351-352 connection creation interactions, 349-350 connection edit interactions, 350-351 creation interactions connection creation interactions, 349-350 consuming CreateRequests, 345 creation sequence, 345-347 producing CreateRequests, 344 delete interactions, 342-343 dependencies, 318 editing support Commands, 332 EditPolicies, 333-334 Requests, 331-332 EditParts, 325-326 AbstractEditPart, 325 adding to diagrams, 334-336 connection EditParts, 330-331 creating, 326-329 deactivating, 336-337

DiagramRootEditPart, 326 EditPartFactory, 326-327 GraphicalEditParts, 325 lifecycle, 334-337 ScalableFreeformRootEditPart, 326 TreeEditParts, 325 EditPolicies, 333-334 graphical view ScrollingGraphicalViewer, 326 setting up, 327-328 Model-View-Controller (MVC) architecture, 324-325 moving and resizing interactions, 347-349 overview, 317 palette, 337 Requests, 331-332 selection interactions keyboard selection, 342 overview, 338-340 selection actions, 342 selection handles, 340-341 selection requests, 342 selection targeting and feedback, 341 Selection Tool, 339 selection with DragTracker, 341 tools, 337 Gen Application element, 543 Gen Diagram Contains Shortcuts To property, 536 Creation Wizard Category ID property, 538 Editing Domain ID property, 538 Icon Path ID property, 538 Shortcuts Provided For property, 536 Synchronized property, 537 Units property, 537 Updater element, 542 Validation Decorators property, 537 Validation Enabled property, 537 Visual ID property, 538 Without Domain property, 538 Gen Editor Generator Copyright Text property, 534 Diagram File Extension property, 534 Domain File Extension property, 535 Domain Gen Model property, 535

Dynamic Templates property, 535 Model ID property, 535 Package Name Prefix property, 535 Same File for Diagram and Model property, 535-536 Template Directory property, 535 Gen Editor View element, 540-541 Gen Link, 539 Gen Navigator element, 541-542 Gen Plugin element, 540 generalization link definition (color modeling diagram), 186-187 Generalization Mapping (color modeling diagram), 200-201 generalizations2columns() query, 600 Generate as Eclipse Editor property (Gen Editor View), 541 Generate Domain Model Navigator property (Gen Navigator), 541 generating color modeling diagram, 201 CSV files, 281-282 HTML with templates (M2T project), 297-301 Iava files with DNC model with templates, 291-297 with Java model and dedicated template, 283-291 requirements diagram, 113 scenario diagram border item adjustment, 166-167 figures plug-in generation, 168-171 GMF generator model, 166 intermediate event outline, 167-168 generation models, 310-311 Generative Modeling Technologies (GMT), 15 generator model, 64 Custom Behavior, 539 Gen Application, 543 Gen Diagram Contains Shortcuts To property, 536 Creation Wizard Category ID property, 538 Editing Domain ID property, 538 Icon Path property, 538

Shortcuts Provided For property, 536 Synchronized property, 537 Units property, 537 Validation Decorators property, 537 Visual ID property, 538 Without Domain property, 538 Gen Diagram Updater, 542 Gen Editor Generator Copyright Text property, 534 Diagram File Extension property, 534 Domain File Extension property, 535 Domain Gen Model property, 535 Dynamic Templates property, 535 Model ID property, 535 Package Name Prefix property, 535 Same File for Diagram and Model property, 535-536 Template Directory property, 535 Gen Editor View, 540-541 Gen Link, 539 Gen Navigator, 541-542 Gen Plugin, 540 mindmap diagram example, 74-76 mindmap generator model, 42 Open Diagram Behavior, 539 overview, 533-534 Property Sheet, 542-543 requirements generator model, 48-49 getActionBarContributor() method, 136 getCommand() method, 332 getContributorId() method, 134 getDecoratorTargetNode() method, 498-500 getDragSourceListener() method, 435 getDropTargetListener() method, 435 getModelChildren() method, 326 getModelSourceConnections() method, 330 getModelTargetConnections() method, 330 getNodeEditPartClass() method, 475, 488 getNotationView() method, 382 getParseCommand() method, 224 getParser() method, 224 getPreferenceColor() method, 211 getPropertySheetPage() method, 134 getReferencedElementEClass() method, 475 getSemanticCommand() method, 482 getSemanticEClass() method, 472

getSourceConnectionAnchor() method, 330 getStrCounter() operation, 592 getTargetConnectionAnchor() method, 330 getTargetEditPart() method, 332 GlobalActionHandlerProvider extension-point, 420-423 GlobalActionHandlerService, 419-423 \_globalId() operation, 578 GMF (Graphical Modeling Framework) runtime, 12, 55, 59. See also GMF Tooling advantages of, 545 canonical containers, 547 color modeling diagram, 181-182 annotation figure definition, 188-189 archetype figure definition, 182-184 color preferences, 205-220 custom parsers, 220-225 generalization link definition, 186-187 generation, 201 gradient figures, 201-205 mapping definition, 190-201 package figure definition, 185-186 tooling definition, 189 command infrastructure, 462-464 contributing to, 546 customization options code modification, 65 decorator models, 66 dynamic templates, 65-66 extension-points, 65 model extensions, 66 dashboard view, 66-67 diagram creation compared to generated diagrams, 485-486 connections, 479-484 diagram editor, 467-468 domain model, 468-469 EditPart definition, 473-475 New Diagram Wizard, 475-476 palette definition, 470-471 properties view, configuring, 476-479 view definition, 471-473 diagram extension custom decorator, 497-500 custom EditPolicy, 493-497

custom style, 490-493 scenario diagram custom view and edit providers, 486-489 diagram structure connection handles, 364-365 context menus, 363-364 Diagram menu, 361 Order menu, 361 Outline view, 366 Palette view, 360 pop-up bars, 365 preferences, 367-368 properties, 356-359 toolbar elements, 354-356 View menu, 361-363 editor, integrating with requirements diagram refactoring editor, 127-130 sharing editing domain, 123-126 sharing file extension, 122-123 element creation, 459-461 extension-points ActionFilterProvider, 424-426 clipboardSupport, 457-458 contributionItemProvider, 409-418 declaring, 373-375 decoratorProvider, 392, 394-396 diagramEventBrokerProvider, 441-443 documentProviders, 453-455 DragDropListenerProvider, 436-438 EditorProvider, 427-428 editpartProvider, 382-385 editpolicyProvider, 386-387 ElementSelectionProvider, 429-430 elementTypeBindings, 449-451 elementTypes, 443-447, 449 GlobalActionHandlerProvider, 420-423 IconProvider, 397-399 layoutProvider, 406-408 logListeners, 452-453 MarkerNavigationProvider, 400-401 modelingAssistantProvider, 404-406 overview, 373-375 paletteProvider, 388-392 ParserProvider, 402-403

Pathmaps, 459 propertiesConfiguration, 453 PropertiesProvider, 431-433 PropertyModifier, 434-435 renderedImageFactory, 456-457 TransferAdapterProvider, 439, 441 viewProvider, 379-382 extensibility mechanisms, 372-373 figures changing appearance based on preference changes, 546 changing dynamically, 546 changing properties of, 547 making nonresizable, 547 viewing elements on diagrams, 547 key bindings connection navigation, 657-658 cycling between diagram editor and palette, 653-654 diagram navigation, 655 palette item navigation, 654 Properties View navigation, 658-659 shape navigation, 655-656 mindmap diagram improvements audits and metrics, 97-103 custom figures, 89-90 custom layout, 82-84 fixed anchor locations, 94-96 graphical definition model, 76-79 mapping definition model, 80-81 preferences settings, 97 subtopic actions, 90-93 subtopic figures, 84-89 tooling definition model, 79 topic figure layout, 81-82 nodes, sticking to border of parent, 548 notation model, 369-372 online resources, 545 overview, 353-354 prerequisites, 546 requirements diagram Creation Wizard, 137-138 diagram definition, 104-107 domain model, 108 EMF and GMF editor integration, 122-130

generation, 113 mapping definition, 107-113 menus and toolbar, 135-137 navigator extension, 139 Outline view, 139-140 properties, 140-145 Properties view, 134-135 selection handling, 130-134 tooling definition, 107 ToolTips, 114-121 runtime component, 60-61 scenario diagram Data Object Mapping, 165 database persistence, 177-181 diagram partitioning, 171-177 Event Mapping, 164 figures plug-in generation, 168-171 Gateway Mapping, 163-164 generation, 166-168 graphical definition, 145-159 Link Mapping, 165 mapping definition, 161 Task Mapping, 161-163 tooling definition, 159-161 services ActionFilterService, 423-426 ContributionItemService, 408-418 DecorationService, 392, 394-396 DiagramEventBrokerService, 441-443 DragDropListenerService, 435-438 EditorService, 427-428 EditPartService, 382-385 EditPolicyService, 385-387 ElementSelectionService, 428-430 GlobalActionHandlerService, 419-423 IconService, 396-399 LayoutService, 406-408 MarkerNavigationService, 399-401 ModelingAssistantService, 403-406 overview, 375-376 PaletteService, 387-392 ParserService, 401-403 PropertiesModifierService, 433-435 PropertiesService, 430-433 Service class, 376

service execution strategies, 376-377 TransferAdapterService, 438-441 ViewService, 378-382 templates, modifying output of, 547 tooling component, 61-62 FAQs, 547-548 generator model, 64, 74-76 graphical definition model, 61-63, 68-69, 76-79 mapping model, 63-64, 70-74, 80-81 tooling definition model, 63, 69, 79 Xpand, 548 GMF (Graphical Modeling Framework) Tooling. See also GMF runtime generator model Custom Behavior, 539 Gen Application, 543 Gen Diagram, 536-538 Gen Diagram Updater, 542 Gen Editor Generator, 534-536 Gen Editor View, 540-541 Gen Link, 539 Gen Navigator, 541-542 Gen Plugin, 540 Open Diagram Behavior, 539 overview, 533-534 Property Sheet, 542-543 graphical definition model Canvas, 505-506 diagram elements, 516-518 Figure Gallery. See Figure Gallery figures, 504 overview, 503 mapping model audits and metrics, 532-533 Canvas Mapping, 520 feature initialization, 527-528 Link Mapping, 529-532 Node Mapping, 522-526 overview, 519-520 side-affixed nodes (pins and ports), 528-529 Top Node Reference, 521-522 tooling definition model, 518-519 GMT (Generative Modeling Technologies), 15

gotoMarker() method, 399 Grab Excess Horizontal Space property (Grid Layout figure), 514 Grab Excess Vertical Space property (Grid Lavout figure), 514 Gradient (diagram elements), 516 gradient figures, 201-205 graphical concrete syntax, 4 graphical definition model, 61-63 Canvas, 505-506 diagram elements Compartment, 518 Connection, 517 Diagram Label, 517 Node, 516-517 overview, 516 Figure Gallery, 504 Border Layout, 513 Compound Border, 510 Custom Border, 510 Ellipse, 509 figure properties, 507-508 Flow Layout, 511-512 Grid Layout, 513-515 Label, 510 Line Border, 510 Margin Border, 510 overview, 506-507 Polygon, 509 Polyline, 509 Polyline Connection, 509 Rectangle, 509 Rounded Rectangle, 509 Stack Layout, 512 TemplatePoint, 509 XY Layout, 512 figures, 504 mindmap diagram example, 68-69, 76-79 overview, 503 of scenario diagram, 145-146 Connection figure, 155-158 Data Object figure, 158-159 Event figures, 152-154 Gateway figure, 149-152 Subprocess figure, 147-149 Task figure, 146-147

HTML, generating with templates (M2T

Modeling Framework) design recommendations, 56-57 filters/layers, 57-58 layout, 58 links in, 58 selecting, 55 shortcuts in, 59 synchronization with domain models, 59 graphical view (GEF) ScrollingGraphicalViewer, 326 setting up, 327-328 GraphicalEditParts (GEF), 325 GraphicalNodeEditPolicy, 350-351 GraphicalViewerKeyHandler, 342 gratuitous graphics, avoiding, 56 Grid Layout, 513-515 Group Icon property (Gen Navigator Child Reference), 542 Group Name property (Gen Navigator Child Reference), 542 guillemets (« »), 606

Graphical Editing Framework. See GEF

runtime; GMF Tooling

Graphical Modeling Framework. See GMF

graphical notations. See also GMF (Graphical

## н

handleActivate() method, 131 handleContentOutlineSelection() method, 132-133 handleNotificationEvent() method, 547 hasPersistentClasses() query, 599 Height command (Diagram menu), 361 helper operations (OML), 562-563 Hide Connector Labels toolbar element, 356 Hide If Empty property (Gen Navigator Child Reference), 542 Horizontal Alignment property (Grid Layout figure), 514 Horizontal Indent property (Grid Layout figure), 514 Horizontal Span property (Grid Layout figure), 514

project), 297-301 HUNT (Human-Usable Textual Notation), 665 Icon Path property Gen Diagram, 538 Gen Editor View, 540

IconProvider extension-point, 397-399 IconService, 396-399 ID property Gen Editor View, 541 Gen Plugin, 540 IDiagramWorkbenchPart interface, 137 IDragDropListenerProvider interface, 435 IF statement (Xpand), 614-615 IGlobalActionHandler interface, 419 IGlobalActionHandlerContext interface, 419 IGraphicEditPart interface, 382 IIconProvider interface, 396 IMM (Information Management Metamodel), 14 imperative iterate expressions (OML), 566-567 imperative operations (OML), 565-566 forEach loop expression, 565 forOne loop expression, 565 switch expression, 566 while loop expression, 565-566 import keyword, 606, 636 importing domain models, 31 XSD, 252-253 in-place transformations, 553 incrStrCounter() operation, 593 indexOf() operation, 586 Information Management Metamodel (IMM), 14 inheritance in mapping operations, 560-561 inherits keyword, 560-561 initArchetypeDefaults() method, 209, 219 initializeDataStore() method, 179 initializeDefaultPreferences() method, 209 initializeEditingDomain() method, 123 initPreferenceStoreListener() method, 211

injector (TCS), 229 inline graphics, 57 insertAt() operation, 583 Insets property (figures), 508 installEditPolicv() method, 335 instance models, 4 instances, 4 integer operations, 621-622 interactions (GEF) connection bend interactions, 351-352 connection creation interactions, 349-350 connection edit interactions, 350-351 creation interactions, 344 consuming CreateRequests, 345 creation sequence, 345-347 producing CreateRequests, 344 delete interactions, 342-343 moving and resizing interactions, 347-349 overview, 338 selection interactions keyboard selection, 342 overview, 338-340 selection actions, 342 selection handles, 340-341 selection requests, 342 selection targeting and feedback, 341 Selection Tool, 339 selection with DragTracker, 341 interfaces IDiagramWorkbenchPart, 137 IDragDropListenerProvider, 435 IGlobalActionHandler, 419 IGlobalActionHandlerContext, 419 IGraphicEditPart, 382 IIconProvider, 396 IOperation, 378 IParser, 401 IPropertiesProvider, 430 IProvider, 376 ISemanticParser, 224 intermediate elements, 554 Intermediate Events, 152 inv prefix, 573 invoking extensions, 637-638 Java, 648 mappings, 570-571

IOperation interface, 378 IParser interface, 401 IPropertiesProvider interface, 430 IProvider interface, 376 ISemanticParser interface, 224 isPrimitive() query, 601 isQuoted() operation, 590 iterator properties, 627

#### **J** Java

business domain model transformations to, 264-276 DSL Instances, 24-25 extensions (Xtend), 639 files, generating with DNC model with templates, 291-297 with Java model and dedicated template, 283-291 IEM (Java EMF Model), 265 JETs (Java Emitter Templates), 34, 665 invocation, 648 *Java Modeling in Color with UML* (Coad, et al), 22, 57 JavaBeautifier, 296 JEM (Java EMF Model), 265 jemUtil.ext file, 283-286 JETs (Java Emitter Templates), 34, 665 joinfields() operation, 583

## Κ

key bindings (GMF) connection navigation, 657-658 cycling between diagram editor and palette, 653-654 diagram navigation, 655 palette item navigation, 654 Properties View navigation, 658-659 shape navigation, 655-656 keyboard selection (GEF), 342 keyboard shortcuts, adding to elements, 90-93 keywords. *See specific keywords* 

## L

Label Offset (diagram elements), 516 labels, 510 Diagram Label, 517 Label Offset (diagram elements), 516 Lanza, Michele, 57 LAST service execution strategy, 377 lastToUpper() operation, 586 late operator, 574-575 launch configurations arguments, 43 in OML, 240-242 layers in graphical notations, 57-58 LayoutEditPolicy, 345 layoutProvider extension-point, 406-408 lavouts Border Layout, 513 combining, 515 custom layout, 82-84 with Draw2d, 322 Flow Layout, 511-512 of graphical notations, 58 Grid Layout, 513-515 Stack Layout, 512 of topic figures, 81-82 XY Layout, 512 LayoutService, 406-408 LET statement, 615-616, 635 libraries. See Stdlib lifecycle of EditParts, 334-337 LightweightSystem (Draw2d), 318-319 Line Border, 510 Line Kind property (shape figures), 508 Line Width property (shape figures), 508 Link Mapping, 529-532 Link Mapping (scenario diagram), 165 links in graphical notations, 58 lists, 624 lite runtime (GMF), 60-61 Literal Strings, 594 literals (Xpand), 628-630 Load Resource dialog, 188 \_localId() operation, 578 Location property (figures), 508 log expression, 576 logListeners extension-point, 452-453

loops

forEach loop expression, 565 forOne loop expression, 565 while loop expression, 565-566

## Μ

M2M (Model-to-Model Transformation), 12-13 M2T (Model-to-Text Transformation) project CSV files, generating, 281-282 HTML, generating, 297-301 Java files, generating with DNC model with templates, 291-297 dncUtil.ext file, 295 Entity bean template, 291-294 workflow file, 296 Java files, generating with Java model and dedicated template, 283-291 Address type output, 287-288 jemUtil.ext file, 283-286 Person class, 289-290 Phone class, 290-291 workflow, 286-287 overview, 277 Xpand, 278-281 Xtend, 278-281 main keyword, 559 Major Alignment property (Flow Layout figure), 511 Major Spacing property (Flow Layout figure), 511 map keyword, 570 mapping definition Aggregation Link Mapping, 198-200 Annotation Mapping, 195 DNC Association Mapping, 195-197 DNC Canvas Mapping, 190 Generalization Mapping, 200-201 Moment-Interval Archetype Mapping, 191-192 Package Mapping, 194 requirements diagram mapping definition, 107-113 Canvas mapping, 108 Requirement link mappings, 112

Requirement node mapping, 111-112 RequirementGroup link mapping, 110 RequirementGroup node mapping, 109-110 Requirements Dependency link mappings, 113 Role Archetype Mapping, 193 of scenario diagram Data Object Mapping, 165 Event Mapping, 164 Gateway Mapping, 163-164 Link Mapping, 165 Task Mapping, 161-163 mapping model, 63-64 audits and metrics, 532-533 Canvas Mapping, 520 feature initialization, 527-528 Link Mapping, 529-532 mindmap diagram example, 70-74, 80-81 Node Mapping, 522-524 containment, 524 node constraints, 526 phantom nodes, 524-526 references, 524 overview, 519-520 side-affixed nodes (pins and ports), 528-529 Top Node Reference, 521-522 mapping operations (OML). See also transformations BPMN and BPEL, 231 disjunction, 562 entry operations, 559-560 execution semantics, 558-559 inheritance, 560-561 invoking mappings, 570-571 mapping body, 558 merger, 561 return statement, 559 syntax, 556-558 Margin Border, 510 Marinescu, Radu, 57 markedAs() operation, 580 MarkerNavigationProvider extension-point, 400-401 MarkerNavigationService, 399-401

markValue() operation, 580 Match Minor Size property (Flow Layout figure), 511 match() operation, 589 matchBoolean() operation, 590 matchFloat() operation, 591 matchIdentifier() operation, 591 matchInteger() operation, 591 MatchSizeAction, 349 Maximum Size property (figures), 508 MDA (Model-Driven Architecture) implemented standards **Business Process Modeling** Notation, 665 Diagram Interchange, 664 Extensible Markup Language, 664 Human-Usable Textual Notation, 665 Meta-Object Facility, 662-663 MOF Models to Text Transformation Language, 664 Object Constraint Language, 663 overview, 662 Query/View/Transformation, 664 Software Process Engineering Metamodel, 666 Unified Modeling Language, 663 overview, 661-662 working relationship implementations influencing specifications, 668 membership, 666-667 open and transparent nature, 668-669 specification compliance, 668 specification delivery, 667-668 MDSD (Model-Driven Software Development) metamodels, 3-4 overview, 7 MDTs. See Model Development Tools menuAboutToShow() method, 136 menus adding to elements, 90-93 requirements diagram, 135-137 merges keyword, 561 merging in mapping operations, 561 models, 237

Messages class, 209 messages.properties file, 210 Meta-Object Facility (MOF), 662-663 metaClassName() operation, 578 metaModel element, 646 Metamodel Explorer view, 240-241 metamodels, 3-4, 32-33. See also domain models methods. See specific methods metrics for mindmap diagram, 97-103 migration of models, 233-237 mindmap and requirements DSL Instances, 23-24 mindmap domain model, 39-40 code generation, 43-45 creating, 41-42 generator model for, 74-76 generator model, creating, 42 graphical definition model for, 68-69, 76-79 improvements to audits and metrics, 97-103 custom figures, 89-90 custom layout, 82-84 fixed anchor locations, 94-96 graphical definition model, 76-79 mapping definition model, 80-81 preferences settings, 97 subtopic actions, 90-93 subtopic figures, 84-89 tooling definition model, 79 topic figure layout, 81-82 mapping model for, 70-74, 80-81 OCL, adding, 45-47 project setup, 40-41 running, 43-45 tooling definition model for, 69, 79 transformations to requirements domain model, 244-251 to XHTML, 251-258 mindmap2csv.mwe file, 282 mindmap2csv.xpt template, 281 mindmap2requirements transformation, 308-309 Minimum Size property (figures), 508

Minor Alignment property (Flow Layout figure), 511 Minor Spacing property (Flow Layout figure), 511 Model Compare (EMF), 11 Model Development Tools (MDT), 662 **BPMN2**, 14 IMM (Information Management Metamodel), 14 OCL (Object Constraint Language), 14 UML2, 13 UML2 Tools, 14 XSD (XML Schema), 13 Model Driven Architecture: Applying MDA to Enterprise Computing (Frankel), 4 Model-Driven Architecture. See MDA Model-Driven Software Development. See MDSD model extensions, 66 Model ID property (Gen Editor Generator), 535 model merge, 237 model migration, 233-237 model operations (OML) add(), 582 asList(), 583 asTransformation(), 582 copy(), 582 createEmptyModel(), 582 insertAt(), 583 joinfields(), 583 objects(), 581 objectsOfType(), 581 prepend(), 582 removeElement(), 581 rootObjects(), 581 Model Query (EMF), 11 model refactoring, 232-233 Model Search (EMF), 11 Model Transaction (EMF), 10 model transformations. See transformations Model Validation (EMF), 10 Model-to-Model Transformation (M2M), 12-13 Model-to-Text Transformation project. See M2T project Model-View-Controller (MVC) architecture, 324-325

model2RDBModel() mapping, 598 Modeling Amalgamation Project, 15 modeling, overview of, 3-5 modelingAssistantProvider extension-point, 404-406 ModelingAssistantService, 403-406 models. See specific models modelSlot element, 644 modeltype declaration, 550-553 MOF (Meta-Object Facility), 662-663 MOF2Text, 664 Moment-Interval Archetype Mapping (color modeling diagram), 191-192 monitor size for graphical notations, 56 moving and resizing interactions (GEF), 347-349 MultiPageSelectionProvider, 130-131 MVC (Model-View-Controller) architecture, 324-325

# Ν

Name Compartment Onl toolbar element, 356 Name property (figures), 508 navigation connection navigation, 657-658 diagram navigation, 655 palette item navigation, 654 Properties View navigation, 658-659 shape navigation, 655-656 Navigator Child Reference property (Gen Navigator), 541-542 navigator extension (requirements diagram), 139 New Diagram Wizard, 475-476 NodeEditPart.xpt template file, 212, 215-216 Nodes, 516-517 node constraints, 526 Node Mapping, 86, 522-524 containment, 524 node constraints, 526 phantom nodes, 524-526 references, 524 sticking to border of parent, 548 nonresizable figures, creating, 547 NonResizeableEditPolicy, 547 normalizeSpace() operation, 588

notation model, 369-372 notations. *See* graphical notations null in transformation declarations, 556 number type operations (OML), 583

# 0

Object Constraint Language (OCL), 14, 663 object keyword, 568 object operations (OML), 578 object-oriented DSLs, 51 Object-Oriented Metrics in Practice (Lanza and Marinescu), 57 objects. See also specific objects creating, 568-570 operations, 620 populating, 568-570 objects() operation, 581 objectsOfType() operation, 581 Oblique Style Routing toolbar element, 355 OCL (Object Constraint Language), 14, 663 adding to mindmap domain model, 45-47 in EMF, 35-38 OML and, 243 testing, 35 OMG MDA (Model-Driven Architecture). See MDA working relationship implementations influencing specifications, 668 membership, 666-667 open and transparent nature, 668-669 specification compliance, 668 specification delivery, 667-668 OML (Operational Mapping Language) Ant tasks and, 243-244 comments, 593 described, 238 element operations, 580 helper operations, 562-563 imperative iterate expressions, 566-567 imperative operations, 565-566 forEach loop expression, 565 forOne loop expression, 565

switch expression, 566 while loop expression, 565-566 launch configurations, 240-242 libraries, 556 mapping invocation, 570-571 mapping operations disjunction, 562 entry operations, 559-560 execution semantics, 558-559 inheritance, 560-561 mapping body, 558 merger, 561 return statement, 559 syntax, 556-558 Metamodel Explorer view, 240-241 model operations, 580 object creation and population, 568-570 OCL statements and, 243 OCL synonyms, 596 operations and iterators collect() operation, 564 select() operation, 563 overview, 550 QVT OML editor, 240 QVT Operational Project wizard, 238-239 resolution operators inv prefix, 573 late, 574-575 resolve, 571-572 resolveIn, 573 resolveone, 573 resolveoneIn, 573 shorthand, 594-595 Std library add() operation, 582 addSuffixNumber() operation, 593 allSubobjects() operation, 579 allSubobjectsOfKind() operation, 579 allSubobjectsOfType() operation, 579 asBoolean() operation, 591 asFloat() operation, 592 asInteger() operation, 592 asList() operation, 583 asOrderedTuple() operation, 578 asTransformation() operation, 582 clone() operation, 579

copy() operation, 582 createEmptyModel() operation, 582 deepclone() operation, 580 endsWith() operation, 587 equalsIgnoreCase() operation, 589 find() operation, 589 firstToUpper() operation, 586 format() operation, 584 getStrCounter() operation, 592 \_globalId() operation, 578 incrStrCounter() operation, 593 indexOf() operation, 586 insertAt() operation, 583 isQuoted() operation, 590 joinfields() operation, 583 lastToUpper() operation, 586 \_localId() operation, 578 markedAs() operation, 580 markValue() operation, 580 match() operation, 589 matchBoolean() operation, 590 matchFloat() operation, 591 matchIdentifier() operation, 591 matchInteger() operation, 591 metaClassName() operation, 578 normalizeSpace() operation, 588 objects() operation, 581 objectsOfType() operation, 581 prepend() operation, 582 quotify() operation, 590 range() operation, 583 removeElement() operation, 581 replace() operation, 588 repr() operation, 578 restartAllStrCounter() operation, 593 rfind() operation, 589 rootObjects() operation, 581 size() operation, 584 startStrCounter() operation, 592 startsWith() operation, 587 stereotypedBy() operation, 580 stereotypedStrictlyBy() operation, 580 subobjects() operation, 578 subobjectsOfKind() operation, 579 subobjectsOfType() operation, 579 substringAfter() operation, 585

substringBefore() operation, 584 toLower() operation, 585 toString() operation, 583 toUpper() operation, 585 trim() operation, 587 unquotify() operation, 590 strings, 594 trace model, 242-243 transformation declarations access keyword, 553 configuration properties, 555 extends keyword, 553 in-place transformations, 553 intermediate elements, 554 modeltype declaration, 550-553 null, 556 predefined variables, 555 renaming elements, 555 transformation execution assert expression, 576-577 log expression, 576 trace model, 575 transform() operation, 577 transformation composition, 577 UML to RDB Transformation Project, 596-603 associationAttributes2columns() mapping, 604 class2columns() query, 600 dataType2columns() query, 600 dataType2primaryKeyColumns() mapping, 603 enumerationAttributes2columns() query, 602 generalizations2columns() query, 600 hasPersistentClasses() query, 599 isPrimitive() query, 601 model2RDBModel() mapping, 598 package2schema() query, 599 package2schemas() mapping, 598 persistentClass2table() query, 599 primitiveAttributes2columns() query, 601 relationshipAttribute2foreignKey() mapping, 603

relationshipAttributes2columns() auery, 602 transformation declaration, 597 open and transparent nature, 668-669 Open Diagram Behavior element, 539 Operational Mapping Language. See OML OperationParserProvider class, 224-225 operations (OML). See specific operations operators inv prefix, 573 late, 574-575 resolve, 571-572 resolveIn, 573 resolveone, 573 resolveoneIn, 573 Order menu commands, 361 org.eclipse.gmf.runtime.draw2d.render plug-ins, 501 org.eclipse.gmf.runtime.gef.ui plug-in, 502 outlet element, 646 Outline property (shape figures), 508 Outline view (diagrams), 139-140, 366 outlineShape() method, 167 overwriting user code modifications, avoiding, 35

## Ρ

package figure definition (color modeling diagram), 185-186 Package Mapping (color modeling diagram), 194 Package Name Prefix property (Gen Editor Generator), 535 Package Selection dialog, 116 package2schema() query, 599 package2schemas() query, 598 packaging DSLs, 303 artwork, 304-305 generation models, 310-311 requirements.product, 312 source deployment, 312-313 user interface plug-ins, 305 actions, 307-310 preferences, 306-307 wizard and perspective, 305-306

paint() method, 320 paintBorder() method, 321 paintChildren() method, 321 paintClientArea() method, 320 paintFigure() method, 320 painting with Draw2d, 320-321 palette cycling between diagram editor and palette, 653-654 definition, 470-471 GEF (Graphical Editing Framework) palette, 337 palette item navigation, 654 Palette view (diagrams), 360 paletteProvider extension-point, 388-392 PaletteService, 387-392 ParserProvider extension-point, 402-403 ParserService, 401-403 partitioning. See diagram partitioning PasteTemplateAction, 345 Pathmaps extension-point, 459 performFinish() method, 137-138 performRequest() method, 333 persistence database persistence, 177-181 persistence alternatives (EMF), 11 persistentClass2table() query, 599 Person class, 289-290 perspectiveExtensions extension-point, 306 perspectives extension-point, 306 phantom nodes, 524-526 Phone class, 290-291 platformUri element, 642 Plug-in Project wizard, 305-306 plug-ins, developing actions, 307-310 preferences, 306-307 wizard and perspective, 305-306 policies. See specific policies Polygon, 509 Polyline, 509 Polyline Connection, 322, 509 polymorphism, Xpand support for, 610-612 pop-up bars (diagrams), 365 populating objects, 568-570

postprocessor element, 646-647 Practitioner (DSL Toolkit), 18-19 predefined variables, 555 preference page templates (color modeling diagram), 216-220 PreferenceInitializer.xpt template, 219 preferencePages extension-point, 306 PreferencePropertyChangeListener class, 211, 214 preferences diagrams, 367-368 mindmap diagram improvements, 97 user interface plug-ins, 306-307 Preferred Size property (figures), 508 prepend() operation, 582 primitiveAttributes2columns() query, 601 Printing Enabled property (Gen Plugin), 540 private extensions (Xtend), 639 product line engineering, 7-8 project interaction, 15-16 project setup, mindmap DSL, 40-41 properties of diagrams Appearance properties, 358-359 Core properties, 358 Rulers & Grid properties, 356-358 of figures common properties, 507-508 Flow Layout figure properties, 511 shape figure properties, 508 of Gen Diagram Contains Shortcuts To property, 536 Creation Wizard Category ID property, 538 Editing Domain ID property, 538 Icon Path property, 538 Shortcuts Provided For property, 536 Synchronized property, 537 Units property, 537 Validation Decorators property, 537 Validation Enabled property, 537 Visual ID property, 538 Without Domain property, 538 of Gen Editor Generator Copyright Text, 534 Diagram File Extension, 534

Domain File Extension, 535 Domain Gen Model, 535 Dynamic Templates, 535 Model ID, 535 Package Name Prefix, 535 Same File for Diagram and Model, 535-536 Template Directory, 535 of Gen Editor View Generate as Eclipse Editor, 541 Icon Path, 540 ID, 541 of Gen Navigator Generate Domain Model Navigator, 541 Navigator Child Reference, 541-542 of Gen Plugin ID, 540 Printing Enabled, 540 Provider, 540 Required Plugin Identifiers, 540 Version, 540 of requirements diagram, 140-145 Custom Property Tab, 140 Property tab, 144 RequirementDescriptionProperty Section class, 142-143 of workflow configuration files, 641 transformation configuration properties, 555 Properties View, 134-135, 476-479, 658-659 propertiesConfiguration extension-point, 453 PropertiesModifierService, 433-435 PropertiesProvider extension-point, 431-433 PropertiesService, 430-433 property operations, 626 Property Sheet element, 542-543 Property tab (requirements diagram), 144 PropertyModifier extension-point, 434-435 PROTECT statement (Xpand), 615 Provider property (Gen Plugin), 540 ProviderPriority, 376 provides() method, 224, 376, 498

# Q

Query/View/Transformation. See QVT quotify() operation, 590 QVT (Query/View/Transformation), 664. See also transformations Core language, 549 OML (Operational Mapping Language). See OML Operational Project wizard, 238-239 Relations language, 549 QVTO scripts, 307

# R

range() operation, 583 RDBMS, UML to RDB Transformation Project, 596-603 associationAttributes2columns() mapping, 604 class2columns() query, 600 dataType2columns() query, 600 dataType2primaryKeyColumns() mapping, 603 enumerationAttributes2columns() query, 602 generalizations2columns() query, 600 hasPersistentClasses() query, 599 isPrimitive() query, 601 model2RDBModel() mapping, 598 package2schema() query, 599 package2schemas() mapping, 598 persistentClass2table() query, 599 primitiveAttributes2columns() query, 601 relationshipAttribute2foreignKey() mapping, 603 relationshipAttributes2columns() query, 602 transformation declaration, 597 Reader (Xpand), 644 real operations, 622 Rectangle, 509 Rectilinear Style Routing toolbar element, 355 recursion (Xtend), 638-639 refactoring editors, 127-130 refactoring models, 232-233

Reference Type property (Gen Navigator Child Reference), 542 refines keyword, 554 refresh() method, 335, 500 refreshChildren() method, 335 refreshVisuals() method, 329-331, 335 RegisterEcoreFile element, 643 RegisterGeneratedEPackage element, 642-643 reject() operation, 631 Relations language (QVT), 231, 549 relationship link mapping (mindmap diagram), 73-74, 77-80 relationshipAttribute2foreignKey() mapping, 603 relationshipAttributes2columns() query, 602 relative coordinates (Draw2d), 323-324 REM statement (Xpand), 616 removeChild() method, 335-336 removeElement() operation, 581 renaming elements, 555 \*.render.awt plug-ins, 501 renderedImageFactory extension-point, 456-457 **REP** (Requirements Elicitation Project), 21 replace() operation, 588 repr() operation, 578 Requests (GEF), 331-332 CreateRequests consuming, 345 producing, 344 selection requests, 342 Required Plugin Identifiers property (Gen Plugin), 540 Requirement link mappings (requirements diagram), 112 Requirement node mapping (requirements diagram), 111-112 Requirement Scenario Project (RSP), 22 RequirementDescriptionPropertySection class, 142-143 RequirementGroup link mapping (requirements diagram), 110 RequirementGroup node mapping (requirements diagram), 109-110 Requirements Dependency link mappings (requirements diagram), 113

requirements diagram Creation Wizard, 137-138 diagram definition, 104-107 domain model, 108 EMF and GMF editor integration refactoring editor, 127-130 sharing editing domain, 123-126 sharing file extension, 122-123 generation, 113 mapping definition, 107-113 Canvas mapping, 108 Requirement link mappings, 112 Requirement node mapping, 111-112 RequirementGroup link mapping, 110 RequirementGroup node mapping, 109-110 Requirements Dependency link mappings, 113 menus and toolbar, 135-137 navigator extension, 139 Outline view, 139-140 partition, 172-177 properties, 140-145 Custom Property Tab, 140 Property tab, 144 RequirementDescriptionProperty Section class, 142-143 Properties view, 134-135 selection handling, 130-134 tooling definition, 107 ToolTips, 114-121 requirements domain model, 47-48 generator model, creating, 48-49 transformations from mindmap domain model, 244-251 Requirements Elicitation Project (REP), 21 Requirements Management Project (RMP), 21 requirements.product, configuring, 312 RequirementsEditorPart class, 127-128 RequirementsGeneralPreferencePage class, 307 ResizableEditPolicy, 348 resizing interactions (GEF), 347-349 resolution operators (OML) inv prefix, 573 late, 574-575

resolve, 571-572 resolveIn. 573 resolveone, 573 resolveoneIn, 573 resolve operator, 571-572 resolveIn operator, 573 resolveone operator, 573 resolveoneIn operator, 573 resolveSemanticElement() method, 382 resources, GMF (Graphical Modeling Framework) resources, 545 restartAllStrCounter() operation, 593 result keyword, 569 return statement, 559 revalidate() method, 322 **REVERSE** service execution strategy, 377 rfind() operation, 589 RMP (Requirements Management Project), 21 Role Archetype Mapping (color modeling diagram), 193 roles of EditPolicies, 333 rootObjects() method, 560, 581 Rounded Rectangle, 509 routing in Draw2d, 322-323 RSP (Requirement Scenario Project), 22 Rulers & Grid properties (diagrams), 356-358 run() method, 309-310, 648 running mindmap domain model, 43-45 runtime, EMF, 33-34 runtime, GMF. See GMF runtime

# S

Same File for Diagram and Model property (Gen Editor Generator), 535-536 Sample Reflective Ecore Model Editor, 39 ScalableFreeformRootEditPart (GEF), 326 scenario diagram, 145 custom view, 486-489 Data Object Mapping, 165 database persistence, 177-181 diagram partitioning Requirements diagram partition, 172-177 subprocess partitions, 171-172

Event Mapping, 164 Gateway Mapping, 163-164 generation border item adjustment, 166-167 figures plug-in generation, 168-171 GMF generator model, 166 intermediate event outline, 167-168 graphical definition, 145-146 Connection figures, 155-158 Data Object figure, 158-159 Event figures, 152-154 Gateway figure, 149-152 Subprocess figure, 147-149 Task figure, 146-147 Link Mapping, 165 mapping definition, 161 Task Mapping, 161-163 tooling definition, 159-161 transformations to test cases, 258-264 scenario domain model, 50-51 scenario DSL Instances, 24-25 ScenarioEditPartProvider class, 488 scripts, QVTO scripts, 307 ScrollingGraphicalViewer (GEF), 326 Select All Links toolbar element, 355 Select All Shapes toolbar element, 355 Select All toolbar element, 355 select() operation, 563, 630 SelectAllAction, 342 selecting graphical notations, 55 selection handling (requirements diagram), 130-134 selection interactions (GEF) keyboard selection, 342 overview, 338-340 selection actions, 342 selection handles, 340-341 selection requests, 342 selection targeting and feedback, 341 Selection Tool, 339 selection with DragTracker, 341 Selection Tool (GEF), 339 selectionProvider attribute (MultiPageSelectionProvider), 130-131 SelectionTreeEditorPart class, 128 Send Backward command (Order menu), 361

Send to Back command (Order menu), 361 serialization syntax, 4 Service class, 376 services ActionFilterService, 423-426 ContributionItemService, 408-418 DecorationService, 392-396 DiagramEventBrokerService, 441-443 DragDropListenerService, 435-438 EditorService, 427-428 EditPartService, 382-385 EditPolicyService, 385-387 ElementSelectionService, 428-430 GlobalActionHandlerService, 419-423 IconService, 396-399 LayoutService, 406-408 MarkerNavigationService, 399-401 ModelingAssistantService, 403-406 overview, 375-376 PaletteService, 387-392 ParserService, 401-403 PropertiesModifierService, 433-435 PropertiesService, 430-433 Service class, 376 service execution strategies, 376-377 TransferAdapterService, 438-441 ViewService, 378-382 setContents() method, 327, 334 setCurrentViewer() method, 131 setForegroundColor() method, 205, 211 setModel() method, 334 setParent() method, 334 setupContentPane() method, 517 shape figures Ellipse, 509 Polygon, 509 Polyline, 509 Polyline Connection, 509 properties, 508 Rectangle, 509 Rounded Rectangle, 509 TemplatePoint, 509 shape navigation, 655-656 SharedImages class, 396 sharing EditingDomain, 123-126 shortcuts in graphical notations, 59

Shortcuts Provided For property (Gen Diagram), 536 shorthand (QVT), 594-595 Show Connector Labels toolbar element, 356 side-affixed nodes (pins and ports), 528-529 simplicity in graphical notations, 56 Size property (figures), 508 size() operation, 584 software factories, 7-8 Software Process Engineering Metamodel (SPEM), 666 software product lines, 7-8 sortBy() operation, 633 source, deploying, 312-313 specifications compliance, 668 implementations influencing specifications, 668 specification delivery, 667-668 SPEM (Software Process Engineering Metamodel), 666 Stack Layout, 512 StandaloneSetup class ExtensionMap element, 643 platformUri element, 642 RegisterEcoreFile element, 643 RegisterGeneratedEPackage element, 642-643 uriMap element, 642 Start Events, 152 startStrCounter() operation, 592 startsWith() operation, 587 startup() method, 378 statements. See specific statements StaticProperty operations, 626 StdLib library, 556 add() operation, 582 addSuffixNumber() operation, 593 allSubobjects() operation, 579 allSubobjectsOfKind() operation, 579 allSubobjectsOfType() operation, 579 asBoolean() operation, 591 asFloat() operation, 592 asInteger() operation, 592 asList() operation, 583 asOrderedTuple() operation, 578

asTransformation() operation, 582 clone() operation, 579 copy() operation, 582 createEmptyModel() operation, 582 deepclone() operation, 580 element operations, 580 endsWith() operation, 587 equalsIgnoreCase() operation, 589 find() operation, 589 firstToUpper() operation, 586 format() operation, 584 getStrCounter() operation, 592 \_globalId() operation, 578 incrStrCounter() operation, 593 indexOf() operation, 586 insertAt() operation, 583 isQuoted() operation, 590 joinfields() operation, 583 lastToUpper() operation, 586 \_localId() operation, 578 markedAs() operation, 580 markValue() operation, 580 match() operation, 589 matchBoolean() operation, 590 matchFloat() operation, 591 matchIdentifier() operation, 591 matchInteger() operation, 591 metaClassName() operation, 578 normalizeSpace() operation, 588 objects() operation, 581 objectsOfType() operation, 581 prepend() operation, 582 quotify() operation, 590 range() operation, 583 removeElement() operation, 581 replace() operation, 588 repr() operation, 578 restartAllStrCounter() operation, 593 rfind() operation, 589 rootObjects() operation, 581 size() operation, 584 startStrCounter() operation, 592 startsWith() operation, 587 stereotypedBy() operation, 580 stereotypedStrictlyBy() operation, 580 subobjects() operation, 578

subobjectsOfKind() operation, 579 subobjectsOfType() operation, 579 substringAfter() operation, 585 substringBefore() operation, 584 toLower() operation, 585 toString() operation, 583 toUpper() operation, 585 trim() operation, 587 unquotify() operation, 590 stereotypedBy() operation, 580 stereotypedStrictlyBy() operation, 580 StoreController class, 179 string operations (OML), 594, 620 addSuffixNumber(), 593 asBoolean(), 591 asFloat(), 592 asInteger(), 592 endsWith(), 587 equalsIgnoreCase(), 589 find(), 589 firstToUpper(), 586 format(), 584 getStrCounter(), 592 incrStrCounter(), 593 indexOf(), 586 isQuoted(), 590 lastToUpper(), 586 match(), 589 matchBoolean(), 590 matchFloat(), 591 matchIdentifier(), 591 matchInteger(), 591 normalizeSpace(), 588 quotify(), 590 replace(), 588 restartAllStrCounter(), 593 rfind(), 589 size(), 584 startStrCounter(), 592 startsWith(), 587 substringAfter(), 585 substringBefore(), 584 toLower(), 585 toUpper(), 585 trim(), 587 unquotify(), 590

styles, custom, 490-493 subobjects() operation, 578 subobjectsOfKind() operation, 579 subobjectsOfType() operation, 579 Subprocess figure, creating for scenario diagram, 147-149 subprocess partitions, 171-172 substringAfter() operation, 585 subtopic actions, mindmap diagram improvements, 90-93 subtopic figures, mindmap diagram improvements, 84-89 subtopic link mapping (mindmap diagram), 73 switch expression, 566, 634 Synchronized property (Gen Diagram), 537 synchronizing domain models, 59, 76 synonyms (OCL), 596

## Т

Task figure, creating for scenario diagram, 146-147 Task Mapping (scenario diagram), 161-163 TCS (Textual Concrete Syntax), 4, 229 Template Directory property (Gen Editor Generator), 535 TemplatePoint, 509 templates dynamic templates, 65-66 mindmap2csv.xpt, 281 modifying output of, 547 TemplateTransferDragSourceListener, 344 Teneo, 177-180 ternary expression, 634 Test and Performance Tools Project (TPTP), 258 - 264test cases, transforming scenarios to, 258-264 testing OCL, 35 text Draw2d support for, 319 with inline graphics, 57 Textual Concrete Syntax (TCS), 4, 229 Textual Generic Editor (TGE), 229 Textual Modeling Framework (TMF), 12, 665 textual syntaxes for DSLs overview, 227-228 TCS (Textual Concrete Syntax), 4, 229 Xtext, 228-229 TGE (Textual Generic Editor), 229 Tiger project, 233 TMF (Textual Modeling Framework), 12, 665 toLower() operation, 585 toolbars elements, 354-356 requirements diagram, 135-137 tooling component (GMF), 61-62 FAOs. 547-548 generator model, 64, 74-76 graphical definition model, 61-63, 68-69, 76-79 mapping model, 63-64, 70-74, 80-81 tooling definition model, 63, 69, 79 tooling definition, 63, 518-519 of color modeling diagram, 189 of mindmap diagram example, 69, 79 of scenario diagram, 159-161 of requirements diagram, 107 tools, GEF (Graphical Editing Framework) tools, 337 Toolsmith artifacts (DSL Toolkit), 17-18 ToolTips, 114-121 Top Node Reference, 521-522 topic figure layout, 81-82 topic mapping (mindmap diagram), 72 TopicEditPart class., 475 toPreferenceConstant() function, 213-214 toRequirementsModel() method, 559 toStateful() method, 561 toString() operation, 583 toUpper() operation, 585 TPTP (Test and Performance Tools Project), 258-264 trace model, 242-243, 575 TransactionalEditingDomain, 123-130 TransferAdapterProvider extension-point, 439-441 TransferAdapterService, 438-441 transform() operation, 577

transformations

business domain models to Java, 264-276 executing assert expression, 576-577 log expression, 576 trace model, 575 transform() operation, 577 transformation composition, 577 implementation techniques, 231-232 M2M (Model-to-Model Transformation), 12-13 M2T (Model-to-Text Transformation) project CSV fies, generating, 281-282 HTML, generating, 297-301 Java files, generating with DNC model with templates, 291-297 Java files, generating with Java model and dedicated template, 283-291 overview, 277 Xpand, 278-281 Xtend, 278-281 mindmap domain model to requirements domain model, 244-251 to XHTML, 251-258 mindmap2requirements, 308-309 for model merge, 237 for model migration, 233-237 for model refactoring, 232-233 OML transformation declarations access keyword, 553 configuration properties, 555 extends keyword, 553 in-place transformations, 553 intermediate elements, 554 modeltype declaration, 550-553 null, 556 predefined variables, 555 renaming elements, 555 scenario diagrams to test cases, 258-264 transformation composition, 577 transparent nature, 668-669 Tree Style Routing toolbar element, 355 TreeContainerEditPolicy, 345 TreeEditParts (GEF), 325

trim() operation, 587 Tufte, Edward, 56 types API documentation AdviceContent operations, 627 AdviceContent properties, 626 Boolean operations, 622 collection operations, 623-624 collection properties, 623 definition operations, 627 definition properties, 627 feature properties, 625 integer operations, 621-622 iterator properties, 627 list operations, 624 object operations, 620 object properties, 619 operation operations, 626 property operations, 626 real operations, 622 StaticProperty operations, 626 string operations, 620 string properties, 620 type operations, 625 type properties, 624 type inference (Xtend), 638 type operations, 625 type properties, 624 typeSelect() operation, 631

## U

UML (Unified Modeling Language), 663 UML to RDB Transformation Project, 596-603 associationAttributes2columns() mapping, 604 class2columns() query, 600 dataType2columns() query, 600 dataType2primaryKeyColumns() mapping, 603 enumerationAttributes2columns() query, 602 generalizations2columns() query, 600 hasPersistentClasses() query, 599

isPrimitive() query, 601 model2RDBModel() mapping, 598 package2schema() query, 599 package2schemas() mapping, 598 persistentClass2table() query, 599 primitiveAttributes2columns() query, 601 relationshipAttribute2foreignKey() mapping, 603 relationshipAttributes2columns() query, 602 transformation declaration, 597 UML2, 13 UML2 Tools, 14 The UML Profile for Framework Architectures (Fontoura, et al), 56 umlPrimitive2rdbPrimitive() method, 568 UmlRdbUtil library, 556 UmlUtil library, 556 Unified Modeling Language (UML), 663 Units property (Gen Diagram), 537 unquotify() operation, 590 updateArchetypeColor() method, 211 UpdateManager (Draw2d), 318 updating model versions, 233-237 uri element, 644 uriMap element, 642 user code modifications, avoiding overwriting, 35 user interaction sequence (Draw2d), 320 user interface plug-ins, developing actions, 307-310 preferences, 306-307 wizard and perspective, 305-306

## V

validate() method, 322 validation, 38 Validation Decorators property (Gen Diagram), 537 Validation Enabled property (Gen Diagram), 537 variables, predefined, 555 Version property (Gen Plugin), 540 versions of models, updating, 233-237 Vertical Alignment property (Grid Layout figure), 514 Vertical property (Flow Layout figure), 511 Vertical Span property (Grid Layout figure), 514 View menu commands, 361-363 viewing Ecore models, 39 viewProvider extension-point, 379-382 views definition, 471-473 scenario diagram custom view, 486-489 ViewService, 378-382 Visual Facets (diagram elements), 516 Visual ID property (Gen Diagram), 538

## W

while loop expression, 565-566 Width command (Diagram menu), 361 Without Domain property (Gen Diagram), 538 wizards Creation Wizard, 137-138 New Diagram Wizard, 475-476 Plug-in Project wizard, 305-306 workflow engine (Xpand), 641 Ant, 649 aspects, 647-648 EMF setup ExtensionMap element, 643 platformUri element, 642 RegisterEcoreFile element, 643 RegisterGeneratedEPackage element, 642-643 uriMap element, 642 Java invocation, 648 properties, 641 Reader, 644 Xpand component, 644-647 expand element, 646 metaModel element, 646 outlet element, 646 postprocessor element, 646-647

## X-Y-Z

xcollect imperative iterate expression, 566 XHTML, transforming mindmap domain model to, 251-258 xmap keyword, 570 XML (Extensible Markup Language), 664 XML Schema (XSD), 13 XOR Fill property (shape figures), 509 XOR Outline property (shape figures), 509 Xpand, 665 AROUND statement, 617-619 DEFINE statement, 607-608 ERROR statement, 616 EXPAND statement, 608-612 expression language, 628 casting, 635 chain expression, 634-635 collect() operation, 631-632 collections, 630 create expression, 635 exists() operation, 633 forAll() operation, 632 let expression, 635 literals, 628-630 reject() operation, 631 select() operation, 630 sortBy() operation, 633 switch expression, 634 ternary expression, 634 typeSelect() operation, 631 EXTENSION statement, 613-614 FILE statement, 612 FOREACH statement, 612-613 in GMF (Graphical Modeling Framework), 548 IF statement, 614-615 IMPORT statement, 606 LET statement, 615-616 M2T (Model-to-Text Transformation) project, 278-281 overview, 605-606 polymorphism support, 610-612 PROTECT statement, 615 REM statement, 616 type system, 619-627

workflow engine Ant. 649 aspects, 647-648 EMF setup, 642-643 Iava invocation, 648 properties, 641 Reader, 644 Xpand component, 644-647 Xtend cached extensions, 639 comments, 637 examples, 640-641 extension invocation, 637-638 extension keyword, 636 extensions, 637 import keyword, 636 Iava extensions, 639 overview, 636 private extensions, 639 recursion, 638-639 type inference, 638 Xpand component, 644-647 expand element, 646 metaModel element, 646 outlet element, 646 postprocessor element, 646-647 XSD (XML Schema), 13, 252-253 Xtend cached extensions, 639 comments, 637 examples, 640-641 extension invocation, 637-638 extension keyword, 636 extensions, 637 import keyword, 636 Java extensions, 639 M2T (Model-to-Text Transformation) project, 278-281 overview, 636 private extensions, 639 recursion, 638-639 type inference, 638 Xtext, 228-229 XY Layout, 512

Zoom toolbar element, 356