RCP Viewer Developers' Guide

Developing Naked Objects business applications on the Eclipse Rich Client Platform.

RCP Viewer Developers' Guide: Developing Naked Objects business applications on the Eclipse Rich Client Platform.

Table of Contents

Chapter 1. Introduction

Background

Note to readers: I've adopted current rather than future tense, eg "RCP Viewer supports" rather than "RCP Viewer will support...". That"s basically so I don't have to rewrite whole tracts of this documentation when we actually implement this stuff (ie we haven't written it yet). My intent is to make the documentation consistent with the software as and when we eventually put out a release.

Naked Objects is a radical approach to business systems design and development. Core business objects such as Customer, Product and Order are exposed directly and to the user instead of being masked behinds the constructs of a conventional user interface.

What this means in practice is that it is not necessary for application developers to have to write any user interface code for domain objects. While it's clear to see that writing less code means that it should be possible to develop applications more quickly, the consequences are much more profound than that:

- developers and end-users communicate only in terms of domain concepts; because domain objects are represented so directly in the user interface, then misunderstandings quickly come to light
- because all functionality must reside on domain objects, it forces the responsibilities of those objects to be assigned appropriately. In short, a *naked* domain object model tends to be a very *good* domain object model.
- the approach lends itself well to using agile (iterative and incremental) development processes. Indeed, it is somewhat difficult to develop with Naked Objects without using such an approach.

The ideas and implementation in Naked Objects have been developed by Richard Pawson and Robert Matthews. Richard is the originator of the ideas that make up Naked Objects; the best exposition of the concept is in his PhD thesis [*** ref]. Robert meanwhile has (almost single-handedly) developed an open source framework to implement these ideas. This framework - the *Naked Objects framework* - is dual-licensed under GPL or can be licensed commercially through their company. Richard and Robert also co-authored a book to express their ideas, again called *Naked Objects*.

Latterly Dan Haywood (project lead for *RCP Viewer*) has been working closely with Richard and Robert on a major project to re-implement the Pensions benefit for the Irish Government's Department of Social & Family Affairs (DSFA). The project has successfully coupled TogetherJ (as championed by Dan) with Naked Objects: TogetherJ synchronizes the code with UML representation at development-time; the Naked Objects framework synchronizes the code with the user-interface at run-time. Dan has also contributed to the overall architecture of the NO framework.

Audience

The RCP Viewer project aims to develop a generic viewing mechanism for Naked Objects applications using the Eclipse Rich Client Platform. In addition, the project aims to develop Eclipse tooling to assist in the development of Naked Objects applications. As such, it has a number of audiences.

- End-users

  As a generic viewing mechanism, RCPViewer offers end-users a rich viewing and editing experience of domain objects. By leveraging Eclipse RCP, the application offers native look
- Domain Programmers

  Domain programmers is the term we use for developers who have domain expertise and/or good analytical skills, who use RCP tooling to develop business applications. In a "traditional" development such developers might be employed as business analysts: however, in Naked Objects (and very much supporting agile development methodologies) such developers represent their analysis by building (ie programming) a domain model.

- View Programmers

  Naked Objects automatically provides a default (and reasonably sophisticated) user interfrace without requring the domain programmer to write any code. However, sometimes it makes sense to provide additional ways of interacting with a domain object.

  For example, a domain object that represents a meeting might want to be shown as a shaded region on a "day-at-a-time" calendar view. Or, a domain object representing a physical location, for example a traffic light in a city), might be rendered on an overlay of the city's grid map.

  RCP Viewer therefore provides a simplified API that allows programmers with (realistically) at least some experience of writing Eclipse plugins to be able to develop additional views. Our expectation is that a large system development project might have one or two developers who staff this role.

- Component Programmers

  RCP Viewer uses Spring Framework to stitch together a number of well-defined components. These components - for such things as persistence, security and distribution - are well documented so that other implementations can be developed as required.

Chapter 2. Naked Objects Concepts & Consequences

In our experience it's rare for those new to the naked objects pattern to appreciate its power. So this chapter aims to provide just a little bit of an insight into what the developing using naked objects means.

Behaviourally Complete Objects

Layers

*** Domain object, state management, authorization

Subtractive Programming

*** NO for identifying constraints.

Domain Driven Design vs Use Case Driven Design

*** Use cases are useful for driving out scenarios.

FAQs

The Naked Objects concept is controversial because it challenges the current dominant design of building business systems today. If you have time, we refer you to Richard and Robert's book, or to Richard's PhD thesis. But let's tackle a couple of points here anyway.

Doesn't the Naked Objects concepts mean having to put UI information into the domain model?

RCP Viewer's metamodel goes to some lengths to distinguish what is intrinsic to the domain model and what is an extension to it for the purpose of UI rendering.

A central tenet of RCP Viewer (indeed, of any Naked Objects-like framework) is that is should be able to render domain objects without any extensions. This is absolutely key since it reduces the feedback loop from business user (domain expert) and domain programmer: the domain programmer must be able to get the application up and running for review without having to specify any UI information.

Thus, providing UI information is really a matter of fine-tuning the rendering once the core domain analysis has been done. One obvious example is specifying the field order, or order of actions.

As an interesting side note, Trygve Reenskaug - the originator of the MVC pattern and external examiner to Richard's thesis - has noted that an original intent of MVS was that every model object should have a default rendering of itself. Unfortunately, they just didn't get around to implementing it - and the rest, as they say, is history.

(For a presentation by Trygve on MVC, see: ***. He gives Naked Objects a mention).

How do I deal with Legacy Domain Objects?

*** virtual domain objects.

Why doesn't RCP Viewer use XML deployment descriptors / why does it rely so much on annotations?

Because we want it to be easy to allow the domain programmer to enter information that they are likely to know/uncover in a single place. If we used deployment descriptors then at best we would need to provide a separate specialized editor so that the domain programmer could update such metadata without having to hack XML. Even that is too much of a disruption - if the domain programmer is "in the flow" capturing their domain model as a POJO, then they should be easily able to capture other relevant information in the same way.

How would I represent a group of widgets in a UI?

Use a immutable 1:1 composition association to a aggregated class referenced by its aggregate.

For example, a Customer has a Name; the Name is itself a collection of title/firstName/lastName. The association from Customer to Name is immutable 1:1 association with composition (aggregation by value) semantics.

How do I deal with large collections (eg PensionScheme to Payments) ?

The preferred solution is generally not to model such as navigable in the domain model. It probably doesn't make sense anyway. Instead, let them be accessed via a query to a Repository.

References

- Trygve2003: http://heim.ifi.uio.no/~trygver/2003/javazone-jaoo/MVC_pattern.pdf

Chapter 3. Standard Programming Model

The "programming model" is the set of coding conventions and annotations that a domain programmer uses to both capture the core domain model and to extend its semantics with other information specific to a particular UI.

RCP Viewer has several programming models,each one building on the rest:

- the *standard* programming model is restricted solely to capturing the structure and relationships of domain classes. The intent is that all semantics captured through the programming model could (reasonaly easily) be justified as being part of domain analysis. If you are aware of the Eclipse Modelling Framework (EMF), think of the standard programming model as corresponding to what can be stored in EMF without EAnnotations.
- the *extended* programming model builds upon the standard programming model, and adds in a small number of semantics that do represent UI information, though only such information as could be used by any framework implementing the naked objects pattern. One example here is @PositionedAt attribute for field and action ordering.
- The *rcpviewer* programming model builds upon both the standard and extended programming models, and adds another set of semantics that repreesnt UI information (likely to be) pertinent only to RCP Viewer. An example is the @ImageUrlAt attribute that is used for building ImageDescriptors.

This chapter focuses on coding conventions and annotations that make up the *standard* programming model.

Domain Class

any class annotated @Domain (not working as of yet: implement IDomainMarker).

Use @Named

Use @DescribedAs

Value Objects

In built are the 8 primitives, String and java.util.Date.

Even though Date is not immutable, it should be considered to be (none of its mutating methods should be invoked).

Any class annotated @Value (not working as of yet: implement IValueMarker).

Attributes

*** getter, setter, isUnset, unset

Use @Named

Use @DescribedAs

Operations (Actions)

*** any public method not annotated with @Programmatic

*** can be instance of static

Use @Named

Use @DescribedAs

Operation Parameters

Use @Named

Chapter 4. Extended Programming Model

This chapter focuses on coding conventions and annotations that make up the *extended* programming model. See *** for a discussion on the different programming models.

Root Class

Starting point for use cases

Plural Name

PositionedAt

State Management

*** Use of @StateOf(...)

Immutable

*** applies to domain classes

NonPersistable

*** applies to domain classes

Internationalization

Every class and class member has an implicit Id whose format - broadly - is akin to a Javadoc reference. For example, com.mycompany.Customer would represent the class of Customer, while com.mycompany.Customer#firstName represents the firstName attribute of Customer.

This id may be used to access internationalized names (cf @Named) and descriptions (cf @DescribedAs) from a ResourceBundle. The key is the id.name or id.desc; for example:

```
com.mycompany.Customer#firstName.name=Vorname

com.mycompany.Customer#firstName.desc=Der Vorname dieses Kunden

com.mycompany.Customer#surname.name=Familienname

com.mycompany.Customer#surname.desc=Der Familienname dieses Kunden
```

Chapter 5. RcpViewer Programming Model

This chapter focuses on coding conventions and annotations that make up the *rcpviewer* programming model. See *** for a discussion on the different programming models

ImageDescriptors

Eclipse RCP requires an ImageDescriptor for most features (classes, attributes and operations) so that they may be rendered with appropriate icons.

Domain programmer can provide the image icon either implicitly or explicitly.

- The implicit approach is to place a .png, .gif or .jpg in the same source directory as the class that is being described
- The explicit approach is to use the @ImageUrlAt annotation which explicitly specifies the URL that holds the image.

The former approach is recommended since it is much less fragile.

Searchable

Indicating that the domain object should be shown on the search dialog.