Contents

Chapter 1: Rich Clients vs. Web Clients.....1

A Rich Client Definition *2*

Technical Aspects..... 2
Developer Aspects..... 3
Enterprise Aspects..... 3

A Web Client Definition..... *4*

Technical Aspects..... 4
Developer Aspects..... 5
Enterprise Aspects..... 6

A Rich Web Client Definition..... *7*

Technical Aspects..... 7
Developer Aspects..... 8
Enterprise Aspects..... 9

History Is Repeating Itself..... *10*

What to Choose Now?..... *11*

Chapter 2: Introducing Eclipse RAP 15

The RAP Vision..... *15*

Componentized and Event-Driven Design..... 15
Programming Using Java APIs 16
Developing for the Web Just As with Java SWT..... 17
Bringing Eclipse RCP to the Web..... 18
Customizing Web Applications with Plug-ins..... 18
Evolving RCP Applications Through Code Reuse 19

RAP Case Studies *20*

Scenario 1: Freedom of Choice 20
Scenario 2: Business-to-Customer Solutions 20
Scenario 3: Intranet Productivity Tools 21
Scenario 4: End Customer Solutions 23
Scenario 5: Business Solutions As Services 25

Pitfalls with RAP *27*

Wrong Expectations 27
Lower Performance 29
No Web in Web 30

Chapter 3: The RAP Architecture 31

The Runtime Layer *32*

The Server Side 32
The Client Side 33

Reimplemented APIs *35*

Standard Widget Toolkit 35
JFace 36
The Workbench 36

Issues and Solutions *37*

RAP Does Not Implement All APIs Yet 37
RAP Will Never Implement Certain APIs 38
RAP is Multiuser 39

RAP Plug-ins and Packages *40*

RAP Version History *41*

The RAP Community *42*

Chapter 4: Developing a RAP Application 45

Installing the Eclipse and RAP SDKs *45*

Running the RAP Sample Application *48*

Creating a Simple Application *52*

Creating an Entry Point 55
Creating a WorkbenchAdvisor 56
Creating a Perspective 57
Creating a View 58
Wrapping Up 60
Running the Application 61

Extending the Application *63*

Changing the Window Appearance 63
Creating a Menu Bar and a Coolbar 65
Creating a Table 67
Creating an Editor 72
Creating a Form for the Editor 77

Chapter 5: Single Sourcing 83

Pros and Cons of Single Sourcing *83*

Project Setup *85*

RAP Proof of Concept 87
Enabling RCP Support for a RAP Application 87
Developing for Both Platforms at the Same Time 88

Running the Mail Demo in RCP *88*

Running the Mail Demo in RAP *89*

Fixing Imports 90
Fixing Extension Points 91
Fixing Nonexistent APIs 93
Adding the Entrypoint 96

Running maildemo in RAP 97
Rerunning the RCP Version 97
Wrapping Up 99
More Single-Sourcing Techniques 100
Using Heavy Reflection 100
Using Interfaces and Reflection 102
Creating Unimplemented Classes 104
Patching RAP 107
Chapter 6: Advanced RAP Features 109
Changing the Look and Feel 109
Configuring RAP to Use a Different Theme 110
Applying the Theme 112
Branding the Application 113
Writing a Custom Widget 116
Creating a Java Widget 116
Creating a qooxdoo Widget 117
Creating a JavaScript-to-Java Connection 119
Creating a View 121
Creating a Resource Definition 122
Integrating the View 124
RAP Without the Workbench 125
Unit Testing in RAP 127
Chapter 7: RAP Deployment 131
Running RAP in Jetty in Equinox 131
Preparing the OSGi Runtime 131
Creating and Exporting a Feature 133
Running the Application in OSGi 135

Running RAP in Equinox in Tomcat..... 136
Preparing the Web Container 136
Creating and Exporting a Web Archive 136
Related Titles 141

Eclipse Rich Ajax Platform: Bringing Rich Clients to the Web

by Fabian Lange

Eclipse Rich Ajax Platform (RAP) is a great technology; the only problem is that there is no book available about the technology and how to use it. With this book, I want to fill this gap and show where and how Eclipse RAP can be used.

I would like to thank the whole Eclipse RAP team, especially Frank Appel, for supporting me while writing this book. I spent some time during fall 2007 with Frank and his team trying to convert an Eclipse RCP application with RAP. This is where I got firsthand experience and expert advice on this great technology. Thank you for creating Eclipse RAP and providing me with valuable input.

I want also to thank Apress for allowing me to publish this book, especially Steve Anglin, Sofia Marchant, and Damon Larson for the great professional support during the creation of this book.

Additional thanks go to my employer, codecentric GmbH, and all of my colleagues who supported me in one way or another during the creation of this book. I am very proud of working with such a great team.

Very special thanks go to my lovely wife, Marie: thank you for supporting me in a way no one else could, day and night, encouraging me to write this book. I love you deeply.

Feel free to visit `www.rap-book.com` *or e-mail me at* `fabian@rap-book.com` *in case of any questions or comments.*

Chapter 1: Rich Clients vs. Web Clients

This chapter describes the properties of *rich clients* and *web clients*, and tries to establish a sound definition of each. The focus is on the differences and characteristics that are important for this book—that is, differences that matter for the Eclipse Rich Ajax Platform (RAP). *Eclipse RAP* combines these technologies, allowing you to create rich web clients from rich clients.

Each of the definitions is structured in three parts:

- *Technical aspects*, which describe the technology and patterns involved or used, and the implications they have.
- *Developer aspects*, which describe key properties, like programming language or tooling.
- *Enterprise aspects*, which basically try to identify why big companies should put money into a technology. Of course, enterprise aspects might be important for end users or in other scenarios as well; however, software and its related costs weigh much more in larger environments. Thus, small differences can impose larger consequences.

The definitions are intended to be generic and valid for all programming languages. However, readers of this book are more likely to be familiar with Java than with any other language, so the examples and references are based on Java.

Note If you are a developer and are just interested in the RAP technology and how to implement it, you might want to skip directly to Chapter 2; but keep in mind that your customers either might have read this chapter or may need advice on finding a solution based on their requirements. For these reasons, you might want to read this chapter first.

A Rich Client Definition

- *Rich client*: Also knows as a desktop application, native application, thick client, or fat client

Technical Aspects

Typically, applications that are considered rich clients don't run in an emulator or browser, but run natively on the operating system of the user's computer. The majority of these rich clients are written in C++, Java, or .NET. Such rich clients have nearly unrestricted access to system resources like memory, storage, input devices (e.g., keyboard and mouse), and output devices (e.g., printer and screen). Only certain functionality, like modifying memory used by other applications, can be restricted by the operating system to prevent malicious applications compromising the system. This access to many system resources allows the application to perform a wide range of tasks, which include operations that can utilize the CPU completely for a noticeable amount of time (e.g., multimedia editing).

Rich clients offer a large feature set optimized to work on a well-defined range of use cases. Often, these applications contain many more features than the user actually needs to perform her job. The look and feel is often designed to be very similar to the host operating system, which makes it easier for users to learn how to use the application, because they can recognize common usage patterns across different applications.

Another feature of rich clients is extensibility using *plug-ins*. Plug-ins are additions provided by vendors or third parties that are able to hook into APIs provided by the rich client and deliver additional functionality.

Usually, data manipulated with the applications is local. If a network is involved at all, it is often just used to pull data, which is then stored locally for processing and sent back to a server later on. This design allows the application to be used offline without any network connection.

Rich client applications are typically able interact with each other using drag-and-drop functionality or other technologies like Microsoft OLE (for Windows), or Bonobo and KParts (for Linux).

Developer Aspects

From a developer point of view, rich clients are easy to implement, because the programming languages and operating systems are very mature and offer a lot of APIs to develop the required functionality. That means that developers don't have to expend as much effort as they used to, as they can reuse existing or provided functionality and can deal with business logic most of the time. They also have access to advanced tooling that helps with the creation, testing, and installation of rich client applications. As the computers running these applications nowadays are powerful enough to run applications that waste CPU power or memory, developers no longer have to spend large amount of time optimizing applications for lesser CPU or memory usage.

A Java, C++, or .NET developer can develop, test, and maintain an entire application, because there is no second technology involved, which would require a different set of competencies.

In the Java world, there are three main players for creating rich clients:

- Eclipse Rich Client Platform
- NetBeans Platform
- Spring Rich

Enterprise Aspects

Rich clients need to be installed, maintained, and updated on each user's workstation. While solutions exist for managing the application maintenance (like HP OpenView, IBM Tivoli, or Microsoft Systems Management Server), users are almost always able to bypass the mechanisms of these solutions. In extreme situations, outdated software

can expose security risks or corrupt data, so it is important to supply users with the most recent version of their applications.

While green IT concepts advertise that end user workstations should be very small to reduce costs and power consumption, rich clients are not ideally designed for this. Rich client applications often need a powerful CPU or a lot of memory, but do not utilize powerful hardware most of the time. To be cost efficient, rich clients would need to move heavy operations to the server side where they can be scaled more efficiently, so that the client computers just need to be capable of handling the few remaining lightweight operations.

In spite of these considerations, software and hardware costs are usually less important than the costs of wasted working time when users have to wait for their applications to respond. In the end, slow applications cost more than what would be spent on enabling users to work as efficiently as possible.

The use of plug-ins with rich clients enables more standardization in a company. It's possible to provide the same foundation application to every department, and, for example, provide sales support for the sales department and financial functionality for accounting using plug-ins. This pattern allows for greater source code reuse than separate applications would.

A Web Client Definition

- *Web client*: Also known as a web application, Internet/intranet application, web user interface, and thin client

Technical Aspects

Contrary to rich clients, web clients do not run on top of the computer operating system, but inside the web browser. This imposes many restrictions on web clients. Components cannot be drawn directly on the

screen; instead, HTML and CSS, which are the markup languages a browser is able to understand, have to be used to lay out the application. The original concept of HTML did not include multimedia or a high degree of interactivity, so many of these features have been added with plug-ins. However, for web applications, it is not predictable whether a certain plug-in is installed on the client side and exactly what functionality the plug-in delivers.

Classical web applications use the network heavily, because the browser basically shows a screen that has been created remotely, on the server. This slows down interaction between the user and the application, as each interaction requires a server roundtrip. Additionally, the entire screen must be re-requested from the server on each roundtrip. Implicitly, this already indicates the main disadvantage of web clients: they cannot work without a network connection and are impacted by the quality of service the network connection provides. Even with a fast network, much data is transferred on each request, which reduces application performance.

Usually, it is said that the advantage of web applications is that they are good cross-platform applications. They can often be used on mobile phones and kiosk systems—basically anywhere a web browser is installed. However, this is somewhat true as well for rich clients that, for example, just need a virtual machine, or a recompilation on the target platform to run.

Web clients do not need to be installed on the user's hardware, which makes it possible for users to access the application even on a machine where the application should not or cannot be installed.

Developer Aspects

The main language of web clients is HTML, combined with a bit of CSS for better-looking interfaces. The complete layout of the screens has to be done either by the application developer or a web designer. HTML and

CSS offer only limited support for creating user interfaces that are usable at a variety of screen resolutions and that integrate into the native look and feel of the user's operating system. Additional issues arise from the fact that end users can change many display settings of the browser and have incompatible browsers or browser versions installed.

To enable user interaction with the application, developers need to provide some kind of server-side logic that is able to render the required HTML and deal with the data submitted by the user using HTML forms. This need sparked such lightweight scripting languages as Perl and PHP, which were well suited for this job. However, scripting languages often fail to provide concepts that are required to develop structured and maintainable source code.

From the Java point of view, much effort has been spent to create a sound server-side solution for web applications with the Servlets, JSP, and JSF standards. .NET also provides server-side solutions based on ASP.

Enterprise Aspects

From an enterprise perspective, web clients solve software maintenance issues. A single server installation is used by all corporate users, which improves data integrity. For example, a tax rate change can be deployed once to the server and all bills created with the application on the server will be correct. With rich clients, some users would be able to create bills with an incorrect tax rate from their local machine, because their application won't have been updated yet. But this is only true as long all users have a similar browser setup for corporate use; otherwise, cross-browser issues could interfere with the application.

A further advantage is that sensitive data is stored only on the server, and just the set of data being used by the user is transferred from the central storage over the network.

The network dependency of web clients is of less impact for enterprise applications, as internal networks are fast enough to power many simultaneous users. For users working at a customer's site, these applications were impossible to use in the past; however, nowadays wireless networks enable remote users to work with web clients. Still, the issue of poor-quality wireless networks (or in some places, no network access) remains.

Due to the limited functionality of web clients, many companies are using web clients only for read-only data, like phone books or branch/department information. These types of applications do not need much functionality because data maintenance and updates are usually taken care of directly by superusers on the main databases.

A Rich Web Client Definition

- *Rich web client*: Also known as a rich Internet application, Ajax client, Web 2.0 client, and fat thin client

Technical Aspects

Since the beginning of the Web 2.0 era, many old web client technologies have been evolving quickly and the definition of *web client* has changed fundamentally. The revised usage of JavaScript allows web applications to modify static content and interact with page elements. By using Ajax as a transport protocol for asynchronous requests, it has become possible to interact with the server while staying on the same screen, which means that users can continue to work while the application fetches data or updates parts of its screen based on the outcome of a server-side computation. This basically removes the disadvantage of unresponsive applications that always refresh to load data from the server.

In *rich web clients*, state is not only kept on the server side, but also on client side. Usually, data state is managed on the server side, while application state, which does not need to be persisted longer than a browser

session, is handled on the client side. Also, it follows the separation-of-concerns pattern, as user-relevant state is just managed by that user on that user's computer.

While network connection is still critical for rich web clients (or maybe even more critical than for traditional web clients, as in total more requests are made at shorter intervals, which are not very tolerant of timeouts), some solutions are emerging, like Google Gears, that let rich web applications continue to work without a network connection by queuing requests to the server in a local storage.

Some people consider very well-designed applications or applications with visual effects to be rich applications, as it's actually not that easy to decide from a user's point of view what qualifies as a rich web application. As a rule of thumb, you could say the following: *if a web application uses JavaScript to load data asynchronously, it is a rich web application*.

Developer Aspects

Manually creating HTML markup is no longer the main method of designing web applications. JavaScript has taken over the lead role, wrapped by some frameworks that make it similar to a traditional programming language, by dealing with cross-browser issues with HTML, CSS, and JavaScript and providing consistent APIs. Additionally, browser manufacturers have worked on adhering to standards, which guarantee that regardless of the browser used, applications can look and work the same.

While the main programming language for the user interface will be often JavaScript, the sever side just generates basic HTML and serves the data used by the application in XML or JSON.

Still, JavaScript has not gained much more functionality than it had already in traditional web clients, which might prevent some features from being implemented in pure JavaScript. For example, many features require Adobe Flash, which is perhaps the most commonly used plug-in for

multimedia functionality. So, in the end, developers of rich web applications often need a broad technology knowledge.

Eclipse RAP and Google Web Toolkit are two frameworks that try to solve server- and client-side programming in pure Java and just generate the appropriate HTML and JavaScript dynamically. This would allow the developers to focus on one development language and environment.

Enterprise Aspects

Rich web clients usually impose more requirements on the web browser. As the technology is still evolving quickly, recent web browsers should be used with rich web clients. On the one hand, it's good for standardization purposes that corporations usually have the same browser installed on all workstations; however, this may be an older version that does not work well with rich web clients. For example, Internet Explorer 6 can still be found in many corporations as the default browser, which does not work very well with rich web clients. The main reason for this is that companies often still use early rich web clients with special ActiveX functionality that made the applications work in Internet Explorer 6. Upgrading to Internet Explorer 7 would be beneficial for many new rich web clients, but in some cases it would make the existing rich web clients work incorrectly. This is a big issue, as existing applications usually have higher priorities than new applications, and it makes the total cost of deploying new applications higher than expected.

Because JavaScript is employed in rich web clients slightly beyond its original intentions, it is not a very stable runtime environment. For business-critical applications that are used throughout an entire working day, this could be an issue, as a browser crash could cause a user to lose some of his work. However, browser manufacturers are working to make JavaScript execution more robust and fix memory leaks.