



ECOLE SUPÉRIEURE PRIVÉE D'INGÉNIERIE  
ET DE TECHNOLOGIES

MINI JAVA PROJECT

---

## **HyperPath GPS based service discovery**

---

***Authors:***

Adel ATTIA  
Chedi TOUEITI

***Supervisor:***

Mr :  
Mohamed Ali BETTAIEB

*June 10, 2011*

## **Thanks**

*From Adel.*

***Attia Adel***

*To all the Open Source projects and communities that make knowledge accessible to everyone and our lives as developers more challenging and fulfilling.*

***Chedi Toueiti***

## Abstract

*“Computers have traditionally lacked the ability to influence and be influenced by the surrounding environment. With the growth of location sensitive services and applications, the physical and the virtual are merging. Location based computing is becoming real thanks to advances in mobile devices, wireless connectivity and location detection technologies” [17]*

A city offers thousands of social events and service promotions a day, and it is difficult to make choices or even to be aware of all of them. The combination of mobile phones and recommender systems can change the way one deals with such abundance. Mobile phones with positioning technology are now widely available, making it easy for service to broadcast their whereabouts. Recommender systems can now identify patterns in people’s movements in order to, for example, recommend events.

To answer this question, we try to build a mini management platform around various business services and social events in a large metropolitan area. We sample location estimations and combine the sample with services in the same area. Upon this data, we extract statistics and reports displaying consumers behaviours.

*“Basically, all the world around you is going to be in geo-coded databases, and you’re going to be walking through these electronic yellow pages.” [17]*

To achieve these goals and with the main constraint being the usage of the Java programming we decided to use a complete Open Source software development stack composed from:

- MySQL database
- GlassFish java application server
- Eclipse RCP for the front end application
- Google Android for the mobile client

# Contents

**Title**

**Thanks**

**Abstract**

<b>1</b>	<b>Introduction</b>	<b>1</b>
1.1	Project purpose . . . . .	2
1.1.1	Project definition . . . . .	2
1.1.2	Challenges . . . . .	2
1.2	State of the art . . . . .	3
1.2.1	Geomarketing . . . . .	3
1.2.2	Geomarketing applications . . . . .	3
1.3	Software solution . . . . .	5
1.4	Geolocalization . . . . .	6
1.4.1	Introduction . . . . .	6
1.4.2	Location-based service . . . . .	6
1.4.3	Automatic Position Identification . . . . .	6
1.4.4	GPS . . . . .	6
1.5	Introduction to Android . . . . .	8
1.5.1	History of Android . . . . .	8
1.5.2	Why Android? . . . . .	8
1.6	Introduction to Eclipse RCP . . . . .	10
1.6.1	Eclipse framework . . . . .	10
1.6.2	Why RCP ? . . . . .	10
1.7	Introduction to GlassFish . . . . .	12
1.7.1	Whats a java application server? . . . . .	12
1.7.2	Why GlashFish . . . . .	12

1.8	Introduction to BIRT . . . . .	13
1.8.1	Introduction to business intelligence . . . . .	13
1.8.2	The Current State of the BI Market . . . . .	13
1.8.3	What is BIRT ? . . . . .	13
1.8.4	Why BIRT? . . . . .	14
1.9	MySQL . . . . .	15
1.9.1	Introduction . . . . .	15
1.9.2	Top Reasons to Use MySQL . . . . .	15
<b>2</b>	<b>Architecture</b>	<b>17</b>
2.1	Application Server . . . . .	18
2.2	JEE . . . . .	18
2.2.1	Entities . . . . .	18
2.2.2	Persistence . . . . .	18
2.3	Web services . . . . .	21
2.3.1	Considering Web Services Architecture . . . . .	21
<b>3</b>	<b>Development process</b>	<b>23</b>
<b>4</b>	<b>Implementation</b>	<b>24</b>
4.1	Data model . . . . .	25
4.2	EJB Server Bean . . . . .	25
4.2.1	Annotations . . . . .	25
4.2.2	Abstraction of data access . . . . .	25
4.2.3	Using EclipseLink JPA Extensions for Mapping . . . . .	26
4.2.4	JPA Persistence Unit . . . . .	27
4.2.5	Calling the bean as a SOAP web service . . . . .	28
<b>5</b>	<b>Further work</b>	<b>30</b>
<b>6</b>	<b>Conclusion</b>	<b>31</b>
<b>A</b>	<b>Team collaboration workflow</b>	<b>32</b>
A.1	Revision control system . . . . .	33
A.1.1	Introduction . . . . .	33
A.1.2	Comparing solutions . . . . .	33
A.2	Git . . . . .	35
A.2.1	Introduction . . . . .	35
A.2.2	Installing Git tools . . . . .	35

A.2.3	Why git? . . . . .	38
<b>B</b>	<b>Setting Up the development environment</b>	<b>47</b>
B.1	Java application server setup . . . . .	48
B.1.1	Getting Glassfish . . . . .	48
B.1.2	Installing Glassfish server . . . . .	48
B.1.3	Installing MySql Database JDBC connector . . . . .	50
B.1.4	Adding the JDBC resources . . . . .	50
B.1.5	Tuning security . . . . .	52
B.2	Database setup . . . . .	53
B.2.1	Installing the MySql server . . . . .	53
B.2.2	Creating the database schema . . . . .	53
B.3	Eclipse IDE setup . . . . .	54
B.3.1	Installing Eclipse . . . . .	54
B.3.2	GlassFish plugin . . . . .	55
B.3.3	Creating the Eclipse Projects . . . . .	59
<b>C</b>	<b>Android simulator</b>	<b>62</b>
<b>D</b>	<b>Eclipse RCP to RAR</b>	<b>63</b>
D.1	Introduction . . . . .	64
D.2	Advantages . . . . .	64
D.2.1	Inter browser compatibility . . . . .	64
D.2.2	Single sourcing . . . . .	64
D.2.3	Integration with BIRT . . . . .	65
	<b>Bibliography</b>	<b>66</b>
	<b>Index</b>	<b>68</b>

# List of Figures

1.1	Some Geomarketing applications . . . . .	5
2.1	Solving Object-Persistence Impedance Mismatch . . . . .	18
2.2	Web Services Architecture . . . . .	21
4.1	Server package architecture . . . . .	29
A.1	GitHub new repository creation . . . . .	36
A.2	GitHub new repository details . . . . .	37
A.3	GitHub new repository adress . . . . .	38
B.1	Adding new server in Eclipse . . . . .	57
B.2	Creating Eclipse entreprise project . . . . .	60
D.1	Eclipse RAP architecture . . . . .	64

# **Chapter 1**

## **Introduction**



## **1.1 Project purpose**

The ability for mobile devices to provide access to information place them amongst the most rapidly growing technologies in the world. The increase of mobile phones equipped with location sensing technology and enabled to elicit user experiences presents a valuable opportunity.

Mobile web portals already allow users to tag, rate, and describe locations as they visit them, in order to aide the discovery of unknown locations of interest. Similarly, users location history can be mined in order to provide personalized recommendations for social events as they occur in the surrounding areas.

tacking the fact that several inexorable trends are driving the development of location based computing in account, we proposed the creation of a complete set of tools to manage and consume data flows around services and geolocation data using a set of Open Source technologies as part of our journey in the Java world.

### **1.1.1 Project definition**

This project aims to address the workflow surrounding the acquisition, analysis and reporting of GPS data collected via a mobile device and exposing these data to the final user using a custom back office interface for the administration side and an android client for the customers one.

The consumer or client will get a mobile application enabling him to define various services preferences and to subscribe to notification flows. The same application will play the role of tracking device and will synchronise with the central database.

For the administration part, a heavy client using the RCP technology will expose an interface for adding new services and categories, managing them and creating reports about the users activities and the buisness popularity. Such reports will be generated using the BIRT framework.

### **1.1.2 Challenges**

## **1.2 State of the art**

### **1.2.1 Geomarketing**

Is an advanced type of study that links a database to digital maps, with practical applications in the marketing field. The research offers new types of data analysis that can be applied to different domains : evaluating the suitability of a site, finding new opportunities as well as identifying the customers' geographical distribution in reference to a certain point of sale.

This information management system combines geographical and socio demographic data, both exclusive to Ipsos-Stat, translating them into comprehensive maps and showing the correlation between the locations and the different variables under study.

### **1.2.2 Geomarketing applications**

Geomarketing applications are crucial in the decision making processes of various businesses : the primary areas can be identified as retail / wholesale trade, in which Geomarketing helps to better identify the customers and their spending habits and study the site analysis and the existence of potential clients in specific trade zones. Geomarketing can benefit finance and insurance companies by helping them implement market share and competition analysis, evaluate their branch locations, identify new markets as well as study the demographic trends in an area. Geomarketing can help transportation and distribution institutions to optimize performance and reduce operating costs.

Cartography is currently revolutionising the way businesses operate, not just because of its ability to illustrate, but because it can produce meaningful and immediate interpretations of complex analyses involving many variables. On a single map, information from a variety of sources can be juxtaposed in layers to reveal correlations with a simple mouse click, for faster search and analysis operations.

### **Correlation with mobile devices marketing**

The findings from a research conducted at the end of 2010 among 5,013 US adult smartphone Internet users. Google commissioned this research with the objectives to better understand how smartphones are used in consumers' daily lives and how

smartphones have influenced the ways consumers search, shop and respond to mobile advertising. Key fact related to location-based deals are:

- 79% use to help with shopping
- 70% use while in the store
- 54% use to find a retailer
- 49% use to compare prices
- 48% use to get promotions and coupons
- 44% use to read reviews and product info
- 34% use to search in-store inventory

Which leads us to buy... 74% make a purchase based on a smartphone search:

- 76% purchase in Store
- 59% purchase online
- 35% purchase on their smartphones

*“The findings of the study have strong implications for businesses and mobile advertisers. Make sure you can be found via mobile search as consumers regularly use their phones to find and act on information. Incorporate location based products and services and make it easy for mobile customers to reach you because local information seeking is common among smartphone users. Develop a comprehensive cross-channel strategy as mobile shoppers use their phones in-store, online and via mobile website and apps to research and make purchase decisions. Last, implement an integrated marketing strategy with mobile advertising that takes advantage of the knowledge that people are using their smartphones while consuming other media and are influenced by it.” [14]*

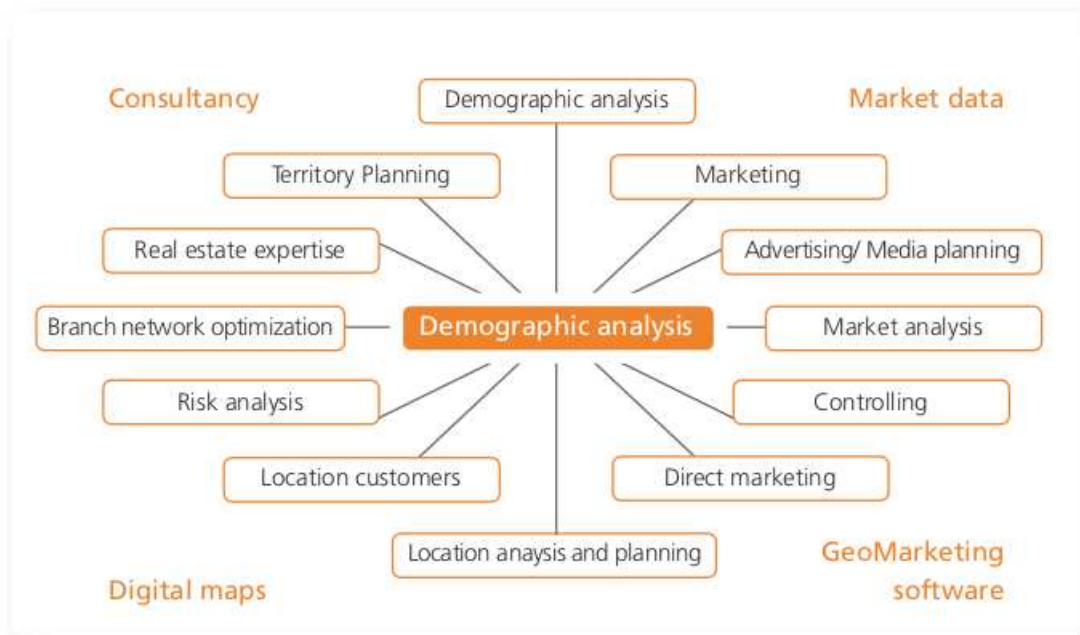


Figure 1.1: Some Geomarketing applications

### 1.3 Software solution

A crucial component of geomarketing is the visualization of data via a specialized geomarketing software application. Working with this software involves several straightforward steps:

- Selecting appropriate maps
- Importing data
- Data evaluation
- Exporting result

## 1.4 Geolocalization

### 1.4.1 Introduction

### 1.4.2 Location-based service

A location-based service (LBS) is an information or entertainment service, accessible with mobile devices through the mobile network and utilizing the ability to make use of the geographical position of the mobile device.

LBS can be used in a variety of contexts, such as health, indoor object search, entertainment, work, personal life, etc... [22]

LBS include services to identify a location of a person or object, such as discovering the nearest banking cash machine or the whereabouts of a friend or employee. LBS include parcel tracking and vehicle tracking services. LBS can include mobile commerce when taking the form of coupons or advertising directed at customers based on their current location. They include personalized weather services and even location-based games. They are an example of telecommunication convergence. [23]

### 1.4.3 Automatic Position Identification

Though location-based services don't require devices that know where you are, there's no question such a feature enhances the experience. But how should the device get that information? One way is for users to identify their location, such as by selecting cross streets from a menu. That, however, requires a manual step, which can be difficult if the device doesn't have a good input method. It also means the device doesn't have a continuous, storable record of its position, which rules out a whole class of services. Moreover, in some cases users don't know where they are, so asking them isn't helpful.

### 1.4.4 GPS

There are two primary mechanisms for determining the location of a handheld device:

- Global Positioning System *GPS*
- Network-based triangulation

The US military developed GPS to help it locate its equipment and people, but the system quickly found significant civilian uses. GPS satellites orbit the earth in a configuration that means a person anywhere on the planet with a clear view of the sky above generally can track the signal from at least three of them. Receiver devices determine their location by measuring arrival time from each signal source.

In the recent past, price (especially the hardware cost) have been the main constraint for many small companies and individuals and the field had not retained the due attention and publicity. But, with the tsunami of mobile devices and smart phones equipped with GPS sensor and network connectivity on the market, the geo-localisation is back and the applications are more proliferous than ever. That's where the first challenge reside:

- Using a mobile device, acquire GPS data and send them to the main server for storage and analysis.
- Store the GPS data in the device in case of missing network connectivity and send them back to the server when restored.
- receive GPS data from the server and use them to guide the target through a specific path

Because data without interpretation is merely obscure sequence of numbers, any decent geo-localization system must have a complete set of tool for the reception, storage, analysis and custom business logic. That's why the proposed system will be provided with back office for managing the following tasks:

- Receive and save the GPS data send by the mobile target in a persisting storage system (database, file...)
- Using third party map data, render a graphical representation of the current target position and the past movements history of one or more target.
- Based on the user data, compute paths and send them as GPS coordinate to the mobile target.
- Provide complete reporting schema using the data acquired via the mobile target.

## 1.5 Introduction to Android

### 1.5.1 History of Android

Historically, developers, generally coding in low-level C or C++, have needed to understand the specific hardware they were coding for, generally a single device or possibly a range of devices from a single manufacturer. As hardware technology and mobile Internet access has advanced, this closed approach has become outmoded.

In more recent years, the biggest advance in mobile phone development was the introduction of Java hosted MIDlets. MIDlets are executed on a Java virtual machine, a process that abstracts the underlying hardware and lets developers create applications that run on the wide variety of devices that supports the Java run time. Unfortunately, this convenience comes at the price of restricted access to the device hardware.

Google acquired the start-up company Android Inc. in 2005 to start the development of the Android Platform. The key players at Android Inc. The Android SDK was first issued as an “*early look*” release in November 2007. In September 2008, T-Mobile announced the availability of the T-Mobile G1, the first smart-phone based on the Android Platform. A few days after that, Google announced the availability of Android SDK Release Candidate 1.0. In October 2008, Google made the source code of the Android Platform available under Apache’s open source license.

Android sits alongside a new wave of mobile operating systems designed for increasingly powerful mobile hardware. Windows Mobile, the Apple iPhone, and the Palm Pre now provide a richer, simplified development environment for mobile applications. However, unlike Android, they’re built on proprietary operating systems that in some cases prioritize native applications over those created by third parties, restrict communication among applications and native phone data, and restrict or control the distribution of third-party apps to their platforms.

### 1.5.2 Why Android?

Android has the potential for removing the barriers to success in the development and sale of a new generation of mobile phone application software. Just as the the standardized PC and Macintosh platforms created markets for desktop and server software.

Here are several Android platform development advantages you can get if you choose Android:

- **Affordability and customization.** The main Android platform development advantage of the Google Android OS it is open source, meaning that its source code is freely available to developers. And because the code is free, there are no licensing fees to deal with and Android app developers have more leeway in terms of developing apps.
- **A low barrier to entry in the mobile application market.** Because the technology is open sourced, Google Android apps are less expensive to produce compared to other mobile operating systems like the iPhone OS and Windows Mobile. Quite often the primary costs for building apps for Android include development and testing expertise of developers, cost of test devices and royalty fees in case you want to distribute your app to third-party app stores like the Android Market.



## 1.6 Introduction to Eclipse RCP

### 1.6.1 Eclipse framework

The Eclipse philosophy is simple and has been critical to its success. The Eclipse Platform was designed from the ground up as an integration framework for development tools. Eclipse also enables developers to easily extend products built on it with the latest object-oriented technologies.

Although Eclipse was designed to serve as an open development platform, it is architected so that its components can be used to build just about any client application. The minimal set of modules needed to build a rich client is collectively known as the Rich Client Platform (RCP).

### 1.6.2 Why RCP ?

The Standard Widget Toolkit (SWT) is the graphical widget toolkit used by eclipse. Originally developed by IBM, it was created to overcome the limitation of the Swing graphical user interface (GUI) toolkit introduced by Sun. Swing is 100% Java and employs a lowest common denominator to draw its components by using Java 2D to call low level operating system primitives. SWT, on the other hand, implements a common widget layer with fast native access to multiple platforms.

SWT's goal is to provide a common API, but avoid the lowest common denominator problem typical of the other portable GUI toolkits. SWT was designed for the following:

**Performance :** SWT claims higher performance and responsiveness, and lower system resource usage than Swing.

**Native look and feel :** Because SWT is a wrapper around native window systems such as GTK+ and Motif, SWT widgets have the exact same look and feel as native ones. This is in contrast to the Swing toolkit, where widgets are close copies of native ones. This is clearly evident just by looking at Swing application.

**Extensibility :** Critics of SWT may claim that the use of native code does not allow for easy inheritance and hurts extensibility. However, both Swing and SWT support for writing new widgets using Java code only.

Besides, the RCP technology is fully compatible with both **BIRT**<sup>1</sup> and **RAP**<sup>2</sup>, which make it the perfect candidate to the backoffice application whether it's deployed in full client mode or web.

---

<sup>1</sup>See Section 1.8 on page 13

<sup>2</sup>See RAP Appendix D.2.3 on page 65

## 1.7 Introduction to GlassFish

### 1.7.1 Whats a java application server?

A web application is a dynamic extension of a web or application server. Web applications are of the following types:

**Presentation-oriented:** A presentation-oriented web application generates interactive web pages containing various types of markup language (HTML, XHTML, XML, and so on) and dynamic content in response to requests. Development of presentation-oriented web applications is covered in Chapter 4, JavaServer Faces Technology through Chapter 9, Developing with JavaServer Faces Technology.

**Service-oriented:** A service-oriented web application implements the endpoint of a web service. Presentation-oriented applications are often clients of service-oriented web applications.

### 1.7.2 Why GlashFish

#### GlassFish advantages

With so many options in Java EE application servers, why choose GlassFish? Besides the obvious advantage of GlassFish being available free of charge, it offers the following benefits:

**Java EE reference implementation:** GlassFish is the Java EE reference implementation. This means that other application servers may use GlassFish to make sure their product complies with the specification. GlassFish could theoretically be used to debug other application servers.

**Supports latest versions of the Java EE specification:** As GlassFish is the reference Java EE specification, it tends to implement the latest specifications before any other application server in the market. As a matter of fact, at the time of writing, GlassFish is the only Java EE application server in the market that supports the complete Java EE 6 specification.

## **1.8 Introduction to BIRT**

### **1.8.1 Introduction to business intelligence**

To give it a formal definition I would say that business intelligence is any tool or method that allows developers to take data or information, process it, manipulate it, and associate it with related information and present it to decision makers. As for a simplified definition, it's presenting information to decision makers in a way that helps them make informed decisions.

### **1.8.2 The Current State of the BI Market**

you can divide the major players in this field into two categories: commercial offerings and open-source offerings. Each category has its own benefits and drawbacks. With the commercial offerings, typically you have familiar names such as Actuate and Business Objects, offering various tools aimed at different levels of business. Some of these tools are large and enterprise reporting platforms that have the ability to process, analyze, and reformat large quantities of data. One of the drawbacks of commercial offerings is the large price associated with them, both in terms of purchasing and in terms of running them.

Then, you have your open-source offerings. Currently there are three big names in the open-source reporting realm: JasperReport, Pentaho, and BIRT. Two of these projects, JasperReports and BIRT, are run by commercial companies who make their money by doing professional services for these offerings to small scale and private projects. Again, there are a number of pros and cons associated with open-source solutions. With open-source, you have full access to the source code of the platform you choose. This allows you to add in functionality, embed it with your existing applications, and actively participate in a development community that is oftentimes very large and around the world. There is little initial cost to open-source software in terms of purchasing, as open-source is free. The cons are that there is typically a cost associated with finding individuals who are knowledgeable in open-source.

### **1.8.3 What is BIRT ?**

BIRT (which stands for Business Intelligence and Reporting Tools) is actually a development framework. Adding the word "Tools" to the title acronym is appropriate; BIRT is in fact a collection of development tools and technologies, used for report

development, utilizing the BIRT framework. BIRT isn't necessarily a product, but a series of core technologies that products and solutions are built on top of, similar in fashion to the Eclipse framework.

#### **1.8.4 Why BIRT?**

As mentioned before, there are other open-source reporting platforms out there. So what makes BIRT stand out over JasperReports or Pentaho? With JasperReports, in reality, it comes down to tastes. I prefer the "What You See Is What You Get" (WYSIWYG) designer of BIRT over JasperReports. I also like the fact that it does not require compiling of reports to run, and that it is an official Eclipse project. However, that is not to say Jasper is not without strengths of its own. Jasper does do pixel-perfect rendering of reports, which is something that BIRT does not do. Later versions of Jasper also support the Hibernate HQL language.

## 1.9 MySQL

### 1.9.1 Introduction

In today's interconnected world, it's almost impossible to find a business that doesn't depend on information in some form or another. Be it marketing data, financial movements, or operational statistics, businesses today live or die by their ability to manage, massage, and filter information flow in order to achieve a competitive advantage.

More often than not, all this data finds a home in a business' relational database management system (RDBMS), a software tool that assists in organizing, retrieving, and cross-referencing information. A large number of such systems are currently available, and you've probably already heard of some of them: Oracle, Sybase, Microsoft Access, and PostgreSQL are well-known names. These database systems are powerful, feature-rich software applications, capable of organizing and searching millions of records at high speeds; as such, they're widely used by businesses and government offices, often for mission-critical purposes.

Recently, though, more and more attention has focused on a relatively new entrant in this field: MySQL. MySQL is a high-performance, multithreaded, multiuser RDBMS built around a client-server architecture. Over the last few years, this fast, robust, and user-friendly database system has become the de facto choice for both business and personal use, notably on account of its advanced suite of data management tools, its friendly licensing policy, and its worldwide support community of users and engineers. This introductory chapter will gently introduce you to the world of MySQL by taking you on a whirlwind tour of MySQL's history, features, and technical architecture.

### 1.9.2 Top Reasons to Use MySQL

**Scalability and Flexibility :** The MySQL database server provides the ultimate in scalability, sporting the capacity to handle deeply embedded applications with a footprint of only 1MB to running massive data warehouses holding terabytes of information.

**High Performance :** A unique storage-engine architecture allows database professionals to configure the MySQL database server specifically for particular applications, with the end result being amazing performance results. With high-speed load utilities, distinctive memory caches, full text indexes, and

other performance-enhancing mechanisms, MySQL offers all the right ammunition for today's critical business systems.

**Web and Data Warehouse Strengths :** MySQL is the de-facto standard for high-traffic web sites because of its high-performance query engine, tremendously fast data insert capability, and strong support for specialized web functions like fast full text searches. These same strengths also apply to data warehousing environments where MySQL scales up into the terabyte range for either single servers or scale-out architectures.

**Comprehensive Application Development :** One of the reasons MySQL is the world's most popular open source database is that it provides comprehensive support for every application development need. Within the database, support can be found for stored procedures, triggers, functions, views, cursors, ANSI-standard SQL, and more. MySQL also provides connectors and drivers (ODBC, JDBC, etc.) that allow all forms of applications to make use of MySQL as a preferred data management server. It doesn't matter if it's PHP, Perl, Java, Visual Basic, or .NET, MySQL offers application developers everything they need to be successful in building database-driven information systems.

**Management Ease :** MySQL offers exceptional quick-start capability with the average time from software download to installation completion being less than fifteen minutes. This rule holds true whether the platform is Microsoft Windows, Linux, Macintosh, or UNIX. Once installed, self-management features like automatic space expansion, auto-restart, and dynamic configuration changes take much of the burden off already overworked database administrators. MySQL also provides a complete suite of graphical management and migration tools that allow a DBA to manage, troubleshoot, and control the operation of many MySQL servers from a single workstation.

**Open Source Freedom:** Pure awesomeness !

# **Chapter 2**

## **Architecture**



## 2.1 Application Server

## 2.2 JEE

### 2.2.1 Entities

### 2.2.2 Persistence

#### Object-Persistence

Java-to-data source integration is a widely underestimated problem when creating enterprise Java applications. This complex problem involves more than simply reading from and writing to a data source. The data source elements include tables, rows, columns, and primary and foreign keys. The Java and Java EE elements include entity classes, business rules, complex relationships, and inheritance. In a nonrelational data source, you must match your Java entities with EIS records or XML elements and schemas. These differences (as shown in the following figure) are known as the object-persistence impedance mismatch.

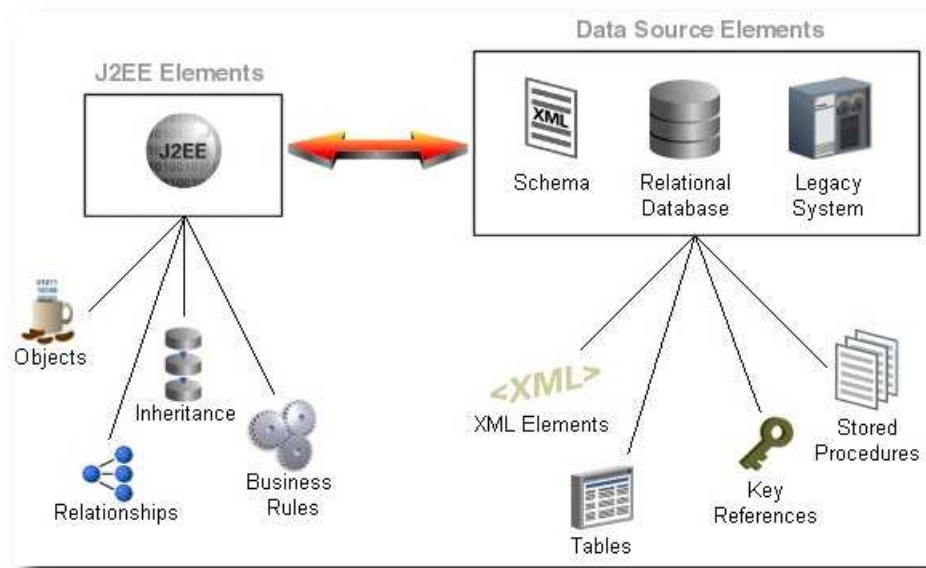


Figure 2.1: Solving Object-Persistence Impedance Mismatch

## What Is Java Persistence API?

The Java Persistence API (JPA) is a lightweight framework for Java persistence based on Plain Old Java Object (POJO). JPA is a part of EJB 3.0 and 3.1 specifications. JPA provides an object relational mapping approach that lets you declaratively define how to map Java objects to relational database tables in a standard, portable way. You can use this API to create, remove and query across lightweight Java objects within both an EJB 3.0/3.1-compliant container and a standard Java SE 5/6 environment.

JPA includes the following concepts:

- Entity—any application-defined object with the following characteristics can be an entity:
  - it can be made persistent
  - it has a persistent identity (a key that uniquely identifies an entity instance and distinguishes it from other instances of the same entity type. An entity has a persistent identity when there is a representation of it in a data store)
  - it is partially transactional in a sense that a persistence view of an entity is transactional (an entity is created, updated and deleted within a transaction, and a transaction is required for the changes to be committed in the database). However, in memory entities can be changed without the changes being persisted.
  - it is not a primitive, a primitive wrapper, or built-in object. An entity is a fine-grained object that has a set of aggregated state that is typically stored in a single place (such as a row in a table), and have relationships to other entities.
- Entity metadata—describes every entity. Metadata could be expressed as annotations (specifically defined types that may be attached to or place in front of Java programming elements) or XML (descriptors).
- Entity manager—enables API calls to perform operations on an entity. Until an entity manager is used to create, read, or write an entity, the entity is just a regular nonpersistent Java object. When an entity manager obtains a reference to an entity, that entity becomes managed by the entity manager. The set of managed entity instances within an entity manager at any given

time is called its persistence context—only one Java instance with the same persistent identity may exist in a persistence context at any time.

You can configure an entity manager to be able to persist or manage certain types of objects, read or write to a particular database, and be implemented by a specific persistence provider. The persistence provider supplies the backing implementation engine for JPA, including the EntityManager interface implementation, the Query implementation, and the SQL generation. Entity managers are provided by an EntityManagerFactory. The configuration for an entity manager is bound to the EntityManagerFactory, but it is defined separately as a persistence unit. You name persistence units to allow differentiation between EntityManagerFactory objects. This way your application obtains control over which configuration to use for operations on a specific entity. The configuration that describes the persistence unit is defined in a persistence.xml file.

The following description expresses relationships between JPA concepts:

- Persistence creates one or more EntityManagerFactory objects
- each EntityManagerFactory is configured by one persistence unit
- EntityManagerFactory creates one or more EntityManager objects
- One or more EntityManager manages one PersistenceContext

## **EclipseLink**

EclipseLink is an advanced, object-persistence and object-transformation framework that provides development tools and run-time capabilities that reduce development and maintenance efforts, and increase enterprise application functionality. Using EclipseLink, you can integrate persistence and object-transformation into your application, while staying focused on your primary domain problem. EclipseLink addresses the disparity between Java objects and data sources. EclipseLink is a persistence framework that manages relational, object-relational data type, EIS, and XML mappings in a seamless manner. This lets you rapidly build applications that combine the best aspects of object technology and the specific data source. EclipseLink lets you do the following:

- Persist Java objects to virtually any relational database supported by a JDBC-compliant driver.

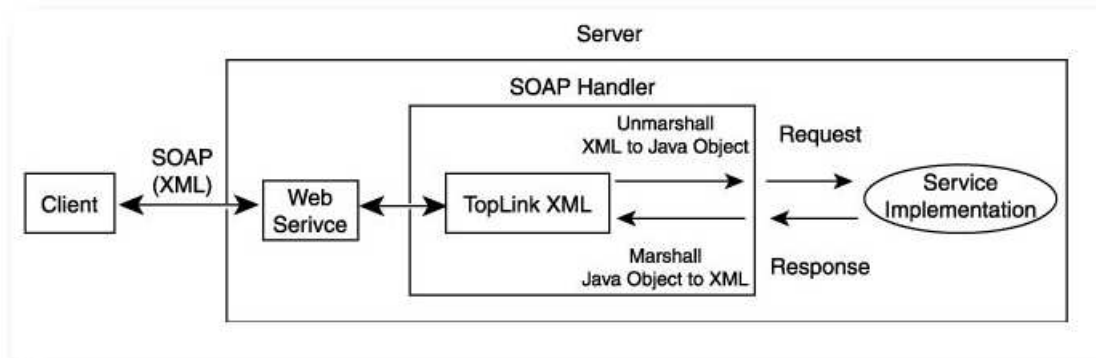


Figure 2.2: Web Services Architecture

- Persist Java objects to virtually any nonrelational data source supported by a Java EE Connector architecture (JCA) adapter using indexed, mapped, or XML enterprise information system (EIS) records.
- Perform in-memory conversions between Java objects and XML Schema (XSD) based XML documents using JAXB.
- Map any object model to any relational or nonrelational schema, using EclipseLink and JPA integration with several tools including Eclipse Dali, Oracle JDeveloper, or NetBeans. The Workbench, a graphical mapping tool, is also provided for backward compatibility for projects using the EclipseLink native API.

## 2.3 Web services

### 2.3.1 Considering Web Services Architecture

A Web services architecture is similar to the Three-Tier Architecture or Session Bean Architecture architecture, however, in a Web services architecture, you encapsulate business logic (the service) in a Web service instead of (or in addition to) using session beans. In a Web services architecture, clients communicate with your application using SOAP messages (XML over HTTP).

As in any architecture, you can use EclipseLink to persist objects to relational or EIS data sources. However, in a Web services architecture, you can also use EclipseLink to map your object model to an XML schema for use with the Web service or as the Web service XML serializer.

## **Chapter 3**

### **Development process**

# **Chapter 4**

## **Implementation**

## **4.1 Data model**

## **4.2 EJB Server Bean**

### **4.2.1 Annotations**

#### **Introduction**

Annotation is a simple, expressive means of decorating Java source code with metadata that is compiled into the corresponding Java class files for interpretation at run time by a JPA persistence provider to manage persistent behavior.

A metadata annotation represents a Java language feature that lets you attach structured and typed metadata to the source code. Standard JPA annotations are in the `javax.persistence` package.

#### **Advantages and Disadvantages of Using Annotations**

Using annotations provides several advantages:

- They are relatively simple to use and understand.
- They provide in-line metadata within with the code that it describes; you do not need to replicate the source code context of where the metadata applies.

The primary disadvantage of annotations is that the metadata becomes unnecessarily coupled to the code. changes to metadata require changing the source code.

### **4.2.2 Abstraction of data access**

Mapping relational databases to objects is a well-understood task by now, thus there are plenty of supporting tools, and normally those tools generate code. Eclipse itself comes with a tool to create entities from tables, so do other IDEs and of course the major UML tools can do that as well.

On the other hand, when you access the database via JPA, you either ask the `EntityManager` for entities that you have an id for, or you create query objects, execute them and get lists of objects as results. If you use queries (which we do), you express them in the Java Persistence Query Language.



### 4.2.3 Using EclipseLink JPA Extensions for Mapping

The Java Persistence API (JPA), part of the Java Enterprise Edition 5 (Java EE 5) EJB 3.0 specification, greatly simplifies Java persistence and provides an object relational mapping approach that allows you to declaratively define how to map Java objects to relational database tables in a standard, portable way that works both inside a Java EE 5 application server and outside an EJB container in a Java Standard Edition (Java SE) 5 application.

EclipseLink persistence provider applies the first annotation that it finds; it ignores other mapping annotations, if specified. If EclipseLink persistence provider does not find any of the mapping annotations from the preceding list, it applies the defaults defined by the JPA specification: not necessarily the `@Basic` annotation.

Here we use various annotation to configure the persistent behavior of your entities:

Listing 4.1 : Emails JPA entity

```
1 @Entity
2 @XmlRootElement
3
4 @Table(
5     name      = "emails",
6     schema    = "",
7     catalog   = "hyperPath",
8
9     uniqueConstraints = {
10         @UniqueConstraint(columnNames = {"address"})
11     })
12
13 @NamedQueries({
14     @NamedQuery(
15         name = "Emails.findAll",
16         query = "SELECT e FROM Emails e"),
17     @NamedQuery(
18         name = "Emails.findById",
19         query = "SELECT e FROM Emails e WHERE e.id = :id"),
20     @NamedQuery(
21         name = "Emails.findByAddress",
22         query = "SELECT e FROM Emails e WHERE e.address = :address")
23 })
24
25 public class Emails implements Serializable {
26     private static final long serialVersionUID = 1L;
27     @Id
28     @NotNull
29     @Basic(optional = false)
30     @Column(name = "id", nullable = false)
31     @GeneratedValue(strategy = GenerationType.IDENTITY)
32     private Integer id;
33
34     @NotNull
```

```

35 @Basic(optional = false)
36 @Size(min = 1, max = 100)
37 @Column(name = "address", nullable = false, length = 100)
38 private String address;
39
40 @ManyToMany(mappedBy = "emailsList")
41 private List<Entities> entitiesList;
42
43 /**
44  * Rest of the Email entity class (POJO) See Emails.java
45  * .....
46  */
47 }

```

## 4.2.4 JPA Persistence Unit

A persistence unit configures various details that are required when you acquire an entity manager. You specify a persistence unit by name when you acquire an entity manager factory. You configure persistence units in JPA persistence descriptor file *persistence.xml*.

In this file, you can specify the vendor extensions that this reference describes by using a *<properties>* element.

Listing 4.2 : Persistence unit persistence.xml

```

1 <?xml version="1.0" encoding="UTF-8"?>
2
3 <persistence version="2.0"
4   xmlns="http://java.sun.com/xml/ns/persistence"
5   xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
6   xsi:schemaLocation="http://java.sun.com/xml/ns/persistence
7   http://java.sun.com/xml/ns/persistence/persistence_2_0.xsd">
8
9   <persistence-unit
10     name="HyperPathServerPU"
11     transaction-type="JTA">
12     <provider>
13       org.eclipse.persistence.jpa.PersistenceProvider
14     </provider>
15
16     <jta-data-source>
17       jdbc/HyperPathData
18     </jta-data-source>
19
20     <exclude-unlisted-classes>
21       false
22     </exclude-unlisted-classes>
23
24     <properties>
25       <property
26         name="eclipseLink.ddl-generation"

```

```

27     value="create-tables"/>
28 </properties>
29
30 </persistence-unit>
31 </persistence>

```

1

## 4.2.5 Calling the bean as a SOAP web service

There is an easy way to access Service Beans: we simply annotate their class as web service. If we don't care about the exact schema of the WDSL that gets generated, it is enough to use one simple annotation like this:

Listing 4.3 : WebService Bean

```

1 @WebService(serviceName = "EmailsServices")
2 @Stateless()
3 public class EmailsServices {
4     @Resource
5     private UserTransaction utx;
6
7     @PersistenceUnit
8     EntityManagerFactory emf;
9
10    /**
11     * List all emails
12     */
13    @WebMethod(operationName = "listAllEmails")
14    public List<Emails> listAllEmails() throws
15        Exception,
16        NonexistentEntityException,
17        RollbackFailureException
18    {
19        emf = Persistence.createEntityManagerFactory("HyperPathServerPU");
20        EmailsJpaController controller = new EmailsJpaController(utx, emf);
21        return controller.findEmailsEntities();
22    }
23
24    /**
25     * Other web services, see EmailsServices.java
26     * ....
27     */
28 }

```

<sup>1</sup>Avoiding Clear text Passwords: EclipseLink does not support storing encrypted passwords in the persistence.xml file. For a Java EE application, you do not need to specify your password in the persistence.xml file. Instead, you can specify a data-source. This datasource is specified on the application server, and can encrypt the your password with its own mechanism.

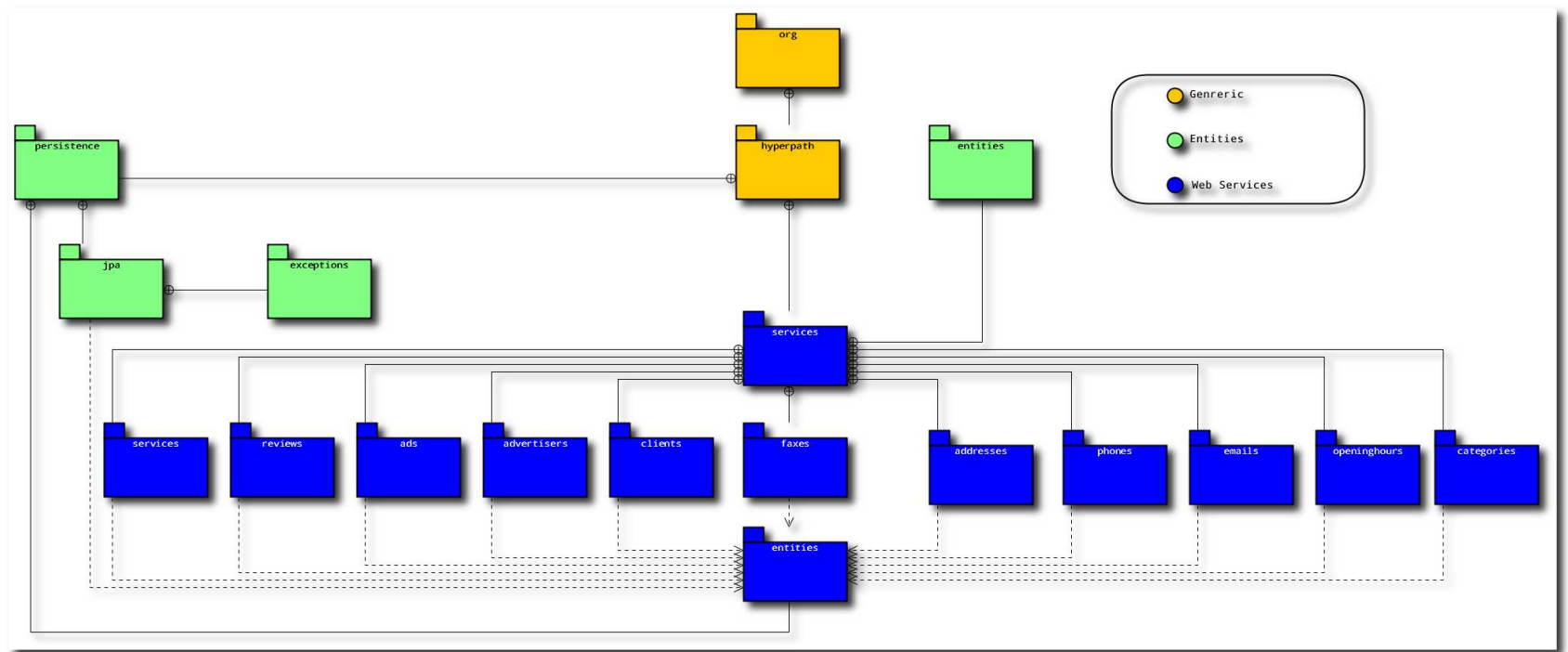


Figure 4.1: Server package architecture

## **Chapter 5**

### **Further work**

# **Chapter 6**

## **Conclusion**

## **Appendix A**

### **Team collaboration workflow**

## **A.1 Revision control system**

### **A.1.1 Introduction**

In computer software engineering, revision control is any practice that tracks and provides control over changes to source code. Software developers sometimes use revision control software to maintain documentation and configuration files as well as source code.

As teams design, develop and deploy software, it is common for multiple versions of the same software to be deployed in different sites and for the software's developers to be working simultaneously on updates. Bugs or features of the software are often only present in certain versions (because of the fixing of some problems and the introduction of others as the program develops). Therefore, for the purposes of locating and fixing bugs, it is vitally important to be able to retrieve and run different versions of the software to determine in which version(s) the problem occurs. It may also be necessary to develop two versions of the software concurrently (for instance, where one version has bugs fixed, but no new features (branch), while the other version is where new features are worked on (trunk).

Moreover, in software development, legal and business practice and other environments, it has become increasingly common for a single document or snippet of code to be edited by a team, the members of which may be geographically dispersed and may pursue different and even contrary interests. Sophisticated revision control that tracks and accounts for ownership of changes to documents and code may be extremely helpful or even necessary in such situations.[21]

### **A.1.2 Comparing solutions**

#### **Distributed vs Centralized**

Distributed revision control (DRCS) takes a peer-to-peer approach, as opposed to the client-server approach of centralized systems. Rather than a single, central repository on which clients synchronize, each peer's working copy of the codebase is a bona-fide repository. Distributed revision control conducts synchronization by exchanging patches (change-sets) from peer to peer. This results in some important differences from a centralized system:

- No canonical, reference copy of the codebase exists by default; only working copies.



- Common operations are fast, because there is no need to communicate with a central server.

Rather, communication is only necessary when pushing or pulling changes to or from other peers. Each working copy effectively functions as a remote backup of the codebase and of its change-history, providing natural protection against data loss.

Other differences are as follows:

- There may be many "central" repositories.
- Code from disparate repositories are merged based on a web of trust, i.e., historical merit or quality of changes.
- Numerous different development models are possible, such as, for example, development / release branches or a Commander / Lieutenant model, allowing for efficient delegation of topical developments in very large projects.
- Lieutenants are project members who have the power to dynamically decide which branches to merge.
- Network is not involved in most operations.
- A separate set of "sync" operations are available for committing or receiving changes with remote repositories.
- Because changes are stored individually in small units, it becomes possible to merge changes between different versions automatically, even if changes have been added to two or more branches.
- Typically, DVCS are based on storing directory trees in place of individual files, which makes it much easier to change names and locations of files, and to move source code sections into separate files.

DVCS proponents point to several advantages of distributed version control systems over the traditional centralised model:

- Allows users to work productively even when not connected to a network
- Makes most operations much faster since no network is involved
- Allows participation in projects without requiring permissions from project authorities, and thus arguably better fosters culture of meritocracy instead of requiring "committer" status
- Allows private work, so users can use their revision control system even for early drafts they do not want to publish
- Avoids relying on a single physical machine as a single point of failure.

- Still permits centralized control of the "release version" of the project
- For FLOSS software projects, it becomes much easier to create a project fork from a project that is stalled because of leadership conflicts or design disagreements.

Software development author Joel Spolsky describes distributed version control as “*possibly the biggest advance in software development technology in the ten years.*”

As a disadvantage of DVCS, one could note that initial cloning of a repository is slower compared to centralized checkout, because all branches and revision history are copied. This may be relevant if access speed is low and the project is large enough. Another problem with DVCS is the lack of locking mechanisms that is part of most centralized VCS and still plays an important role when it comes to non-mergable binary files such as graphic assets.

## A.2 Git

### A.2.1 Introduction

Git is a distributed revision control system with an emphasis on speed. Git was initially designed and developed by Linus Torvalds for Linux kernel development. Every Git working directory is a full-fledged repository with complete history and full revision tracking capabilities, not dependent on network access or a central server. Git is free software distributed under the terms of the GNU General Public License version 2

### A.2.2 Installing Git tools

#### Installing Git

If you want to install Git on Linux via a binary installer, you can generally do so through the basic package-management tool that comes with your distribution. If you're on Fedora, you can use yum:

```
$ yum install git-core
```

Or if you're on a Debian based distribution like Ubuntu, try apt-get:

```
$ apt-get install git-core
```

## Install Eclipse Egit plugin

EGit is an Eclipse Team provider for the Git version control system. The EGit project is implementing Eclipse tooling on top of the JGit Java implementation of Git.[18]

Use the Eclipse Update manager to install the EGit plugin from <http://download.eclipse.org/egit/updates>. The latest features can be found in the nightly build but this build is not as stable as the official update site target <http://download.eclipse.org/egit/updates-nightly>.

## Creating a GitHub repository

Every time you make a commit with Git, it is stored in a repository (a.k.a. “repo”). To put your project up on GitHub, you’ll need to have a GitHub repository for it to live in.

Click New Repository.

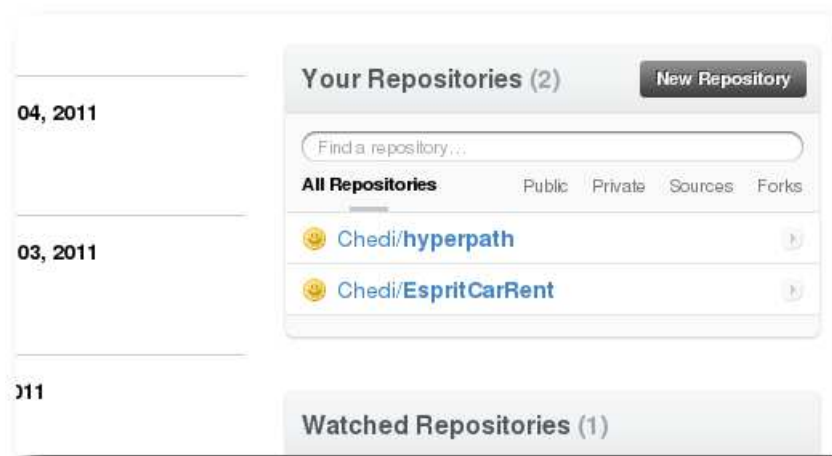


Figure A.1: GitHub new repository creation

Fill out the information on this page. When you’re done, click “Create Repository.”

Once this done you can use the Git repository adress displayed on the top with your favorite git client.

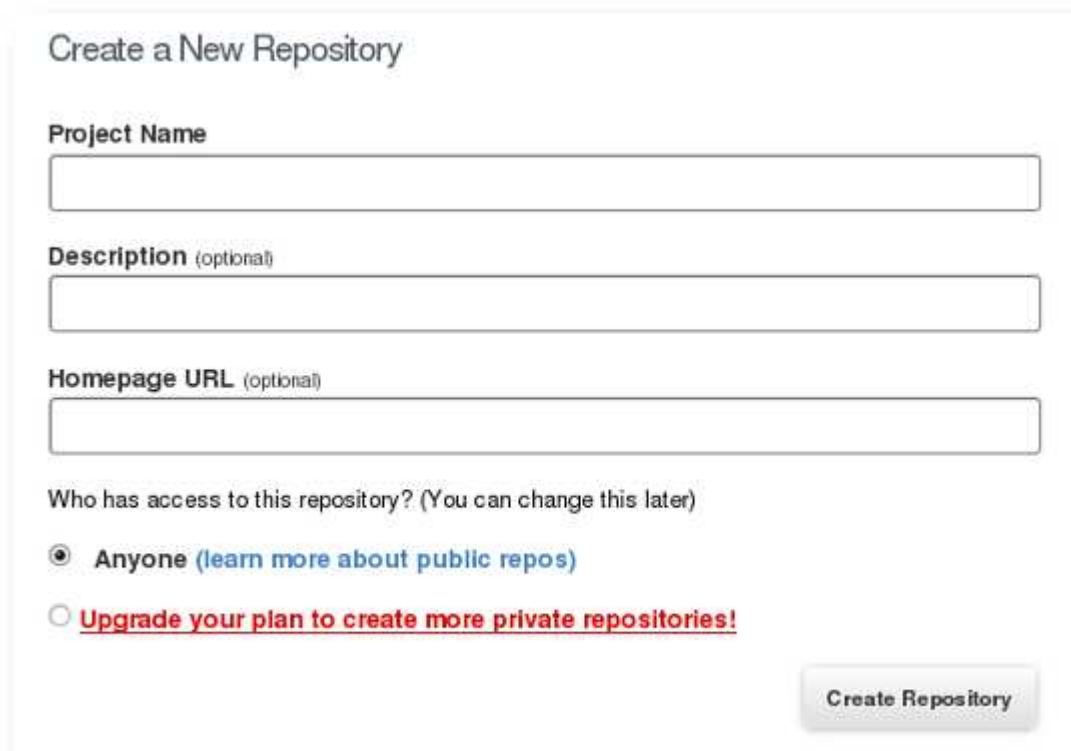
The image shows a screenshot of the GitHub 'Create a New Repository' form. At the top, the title 'Create a New Repository' is displayed. Below it are three input fields: 'Project Name', 'Description (optional)', and 'Homepage URL (optional)'. Under the 'Description' field, there is a note: 'Who has access to this repository? (You can change this later)'. There are two radio button options: the first is 'Anyone (learn more about public repos)' with a blue link, and the second is 'Upgrade your plan to create more private repositories!' in red text. A 'Create Repository' button is located at the bottom right of the form.

Figure A.2: GitHub new repository details

**Create a README for your repo** While a README isn't a required part of a GitHub repo, it is a good idea to have one. READMEs are a great place to describe your project or add some documentation such as how to install or use your project.

If you include a file with the filename "README" in your repo, it will automatically be shown on your repo's front page. GitHub supports a number of different README formats. Usually the README will result in a basic text file but other formats like .markdown or .textile can be used to render HTML content like links and headers.



Figure A.3: GitHub new repository adress

### A.2.3 Why git?

For a thorough discussion on the pros and cons of Git compared to centralized source code control systems, see the web. As a developer, I prefer Git above all other tools around today.

#### **cheap branching and easy merging**

Probably the most compelling feature of Git, since it often fundamentally changes the way that many developers work, is Gits branching model. Instead of the popular VCS branching method of simply cloning into a seperate directory for a branch, Git lets you switch between branches in a single working directory. Add to that the fact that creating and switching between branches is nearly instant, not all of your branches need to be shared, and it's easy to stash partially completed work - means that the way you work can be incredibly different.

Instead of only having branches for major development line departures, Git developers routinely create, merge and destroy multiple branches a week, or even per day. Often each feature or bug you are working on can have its own branch, merged in only when it is complete. This model allows you to experiment quickly, easily and safely - without having to go through hoops to get back to where you where. It enables and encourages a non-linear development cycle, where you can work on multiple lines of thought in parallel without them stepping on each other.

Many developers feel that this is so incredibly helpful and has changed their

workflow and productivity so much that they dub it the “killer feature” of Git.

Git really changed the way developers think of merging and branching. From the classic CVS/Subversion world I came from, merging/branching has always been considered a bit scary and something you only do every once in a while.

But with Git, these actions are extremely cheap and simple, and they are considered one of the core parts of your daily workflow, really. As a consequence of its simplicity and repetitive nature, branching and merging are no longer something to be afraid of. Version control tools are supposed to assist in branching/merging more than anything else.

## **Offline and fast**

Git is fully distributed, which means that it can work almost entirely offline. In stark contrast to VCS tools like Perforce or Subversion, Git does nearly all of its operations without needing a network connection, including history viewing, difference viewing and committing.

This also means that Git is very fast compared to those systems partially due to the fact that none of these operations has any dependency on network latency. For example, take a look at how fast the simple ‘log’ command takes to run in Git and in Subversion.

Git at 0.3 seconds vs Subversion at 3.7 seconds. That is a difference of a full order of magnitude. You’ll find similar differences with nearly any command comparison. For example, adding the popular famfamfam icon set and committing them. Since you can separate the commit from the network ‘push’ in Git, this action takes a quarter of a second in Git, but 45 seconds in Subversion.

Even if you needed to push to a shared repository at that time as well, it still takes far, far less time than Subversion.

If you just want to commit and keep working, you’re looking at a huge time difference - one that severely changes your workflow. Most commands in Git seem instantaneous - no more typing ‘svn commit’ and then going for a cup of coffee.

## **Small**

Git is also very space efficient. For example, if you import the Django project from SVN into Git and compare their checkout/clone sizes, Git comes out very favorably.

```
$ du -d 1 -h
44M ./django-git
```

53M ./django-svn

Interestingly, it is even smaller than the Subversion checkout, which is pretty amazing, considering that the Git clone contains the entire history of the project - every version of every file back to the first commit, whereas the Subversion checkout is just the last version of the project.

### **Decentralized but centralized**

The repository setup that we use and that works well with this branching model, is that with a central “truth” repo. Note that this repo is only considered to be the central one (since Git is a DVCS, there is no such thing as a central repo at a technical level).

Each developer pulls and pushes to origin. But besides the centralized push-pull relationships, each developer may also pull changes from other peers to form sub teams. For example, this might be useful to work together with two or more developers on a big new feature, before pushing the work in progress to origin prematurely.

### **The main branches**

At the core, the development model is greatly inspired by existing models out there. The central repo holds two main branches with an infinite lifetime: master develop. The master branch at origin should be familiar to every Git user. Parallel to the master branch, another branch exists called develop.

We consider origin/master to be the main branch where the source code of HEAD always reflects a production-ready state.

We consider origin/develop to be the main branch where the source code of HEAD always reflects a state with the latest delivered development changes for the next release. Some would call this the “integration branch”. This is where any automatic nightly builds are built from.

When the source code in the develop branch reaches a stable point and is ready to be released, all of the changes should be merged back into master somehow and then tagged with a release number. How this is done in detail will be discussed further on.

Therefore, each time when changes are merged back into master, this is a new production release by definition. We tend to be very strict at this, so that theoretically, we could use a Git hook script to automatically build and roll-out our software to our production servers everytime there was a commit on master.

### Supporting branches

Next to the main branches master and develop, our development model uses a variety of supporting branches to aid parallel development between team members, ease tracking of features, prepare for production releases and to assist in quickly fixing live production problems. Unlike the main branches, these branches always have a limited life time, since they will be removed eventually.

The different types of branches we may use are:

- Feature
- branches
- Release
- branches
- Hotfix
- branches

Each of these branches have a specific purpose and are bound to strict rules as to which branches may be their originating branch and which branches must be their merge targets. We will walk through them in a minute.

By no means are these branches “special” from a technical perspective. The branch types are categorized by how we use them. They are of course plain old Git branches.

### **Feature branches**

May branch off from: develop Must merge back into: develop Branch naming convention: anything except master, develop, release-\*, or hotfix-\*

Feature branches (or sometimes called topic branches) are used to develop new features for the upcoming or a distant future release. When starting development of a feature, the target release in which this feature will be incorporated may well be unknown at that point. The essence of a feature branch is that it exists as long as the feature is in development, but will eventually be merged back into develop (to definitely add the new feature to the upcoming release) or discarded (in case of a disappointing experiment).

Feature branches typically exist in developer repos only, not in origin.

### **Creating a feature branch**

When starting work on a new feature, branch off from the develop branch.



```
$ git checkout -b myfeature develop
Switched to a new branch "myfeature"
```

Incorporating a finished feature on develop

Finished features may be merged into the develop branch definitely add them to the upcoming release:

```
$ git checkout develop
Switched to branch 'develop'
$ git merge --no-ff myfeature
Updating eal1b82a..05e9557
[Summary of changes]
$ git branch -d myfeature
Deleted branch myfeature (was 05e9557).
$ git push origin develop
```

The `--no-ff` flag causes the merge to always create a new commit object, even if the merge could be performed with a fast-forward. This avoids losing information about the historical existence of a feature branch and groups together all commits that together added the feature. Compare:

In the latter case, it is impossible to see from the Git history which of the commit objects together have implemented a feature—you would have to manually read all the log messages. Reverting a whole feature (i.e. a group of commits), is a true headache in the latter situation, whereas it is easily done if the `--no-ff` flag was used.

Yes, it will create a few more (empty) commit objects, but the gain is much bigger than that cost.

Unfortunately, I have not found a way to make `--no-ff` the default behaviour of `git merge` yet, but it really should be.

Release branches

May branch off from: develop Must merge back into: develop and master  
Branch naming convention: release-\*

Release branches support preparation of a new production release. They allow for last-minute dotting of i's and crossing t's. Furthermore, they allow for minor bug fixes and preparing meta-data for a release (version number, build dates, etc.). By doing all of this work on a release branch, the develop branch is cleared to receive features for the next big release.

The key moment to branch off a new release branch from develop is when develop (almost) reflects the desired state of the new release. At least all features that are targeted for the release-to-be-built must be merged in to develop at this point in time. All features targeted at future releases may not—they must wait until after the release branch is branched off.

It is exactly at the start of a release branch that the upcoming release gets assigned a version number—not any earlier. Up until that moment, the develop branch reflected changes for the “next release”, but it is unclear whether that “next release” will eventually become 0.3 or 1.0, until the release branch is started. That decision is made on the start of the release branch and is carried out by the project’s rules on version number bumping. Creating a release branch

Release branches are created from the develop branch. For example, say version 1.1.5 is the current production release and we have a big release coming up. The state of develop is ready for the “next release” and we have decided that this will become version 1.2 (rather than 1.1.6 or 2.0). So we branch off and give the release branch a name reflecting the new version number:

```
$ git checkout -b release-1.2 develop
Switched to a new branch "release-1.2"
$ ./bump-version.sh 1.2
Files modified successfully, version bumped to 1.2.
$ git commit -a -m "Bumped version number to 1.2"
[release-1.2 74d9424] Bumped version number to 1.2
1 files changed, 1 insertions(+), 1 deletions(-)
```

After creating a new branch and switching to it, we bump the version number. Here, bump-version.sh is a fictional shell script that changes some files in the working copy to reflect the new version. (This can of course be a manual change—the point being that some files change.) Then, the bumped version number is committed.

This new branch may exist there for a while, until the release may be rolled out definitely. During that time, bug fixes may be applied in this branch (rather than on the develop branch). Adding large new features here is strictly prohibited. They must be merged into develop, and therefore, wait for the next big release. Finishing a release branch

When the state of the release branch is ready to become a real release, some actions need to be carried out. First, the release branch is merged into master (since every commit on master is a new release by definition, remember). Next,

that commit on master must be tagged for easy future reference to this historical version. Finally, the changes made on the release branch need to be merged back into develop, so that future releases also contain these bug fixes.

The first two steps in Git:

```
$ git checkout master
Switched to branch 'master'
$ git merge --no-ff release-1.2
Merge made by recursive.
[Summary of changes]
$ git tag -a 1.2
```

The release is now done, and tagged for future reference. Edit: You might as well want to use the -s or -u <key> flags to sign your tag cryptographically.

To keep the changes made in the release branch, we need to merge those back into develop, though. In Git:

```
$ git checkout develop
Switched to branch 'develop'
$ git merge --no-ff release-1.2
Merge made by recursive.
[Summary of changes]
```

This step may well lead to a merge conflict (probably even, since we have changed the version number). If so, fix it and commit.

Now we are really done and the release branch may be removed, since we don't need it anymore:

```
$ git branch -d release-1.2
Deleted branch release-1.2 [was ff452fe].
```

## Hotfix branches

May branch off from: master Must merge back into: develop and master Branch naming convention: hotfix-\*

Hotfix branches are very much like release branches in that they are also meant to prepare for a new production release, albeit unplanned. They arise from the

necessity to act immediately upon an undesired state of a live production version. When a critical bug in a production version must be resolved immediately, a hotfix branch may be branched off from the corresponding tag on the master branch that marks the production version.

The essence is that work of team members (on the develop branch) can continue, while another person is preparing a quick production fix. Creating the hotfix branch

Hotfix branches are created from the master branch. For example, say version 1.2 is the current production release running live and causing troubles due to a severe bug. But changes on develop are yet unstable. We may then branch off a hotfix branch and start fixing the problem:

```
$ git checkout -b hotfix-1.2.1 master
Switched to a new branch "hotfix-1.2.1"
$ ./bump-version.sh 1.2.1
Files modified successfully, version bumped to 1.2.1.
$ git commit -a -m "Bumped version number to 1.2.1"
[hotfix-1.2.1 41e61bb] Bumped version number to 1.2.1
1 files changed, 1 insertions(+), 1 deletions(-)
```

Don't forget to bump the version number after branching off! Then, fix the bug and commit the fix in one or more separate commits.

```
$ git commit -m "Fixed severe production problem"
[hotfix-1.2.1 abbe5d6] Fixed severe production problem
5 files changed, 32 insertions(+), 17 deletions(-)
```

### Finishing a hotfix branch

When finished, the bugfix needs to be merged back into master, but also needs to be merged back into develop, in order to safeguard that the bugfix is included in the next release as well. This is completely similar to how release branches are finished.

First, update master and tag the release.

```
$ git checkout master
Switched to branch 'master'
$ git merge --no-ff hotfix-1.2.1
```

```
Merge made by recursive.  
[Summary of changes]  
$ git tag -a 1.2.1
```

Edit: You might as well want to use the -s or -u <key> flags to sign your tag cryptographically.

Next, include the bugfix in develop, too:

```
$ git checkout develop  
Switched to branch 'develop'  
$ git merge --no-ff hotfix-1.2.1  
Merge made by recursive.  
[Summary of changes]
```

The one exception to the rule here is that, when a release branch currently exists, the hotfix changes need to be merged into that release branch, instead of develop. Back-merging the bugfix into the release branch will eventually result in the bugfix being merged into develop too, when the release branch is finished. (If work in develop immediately requires this bugfix and cannot wait for the release branch to be finished, you may safely merge the bugfix into develop now already as well.)

Finally, remove the temporary branch:

```
$ git branch -d hotfix-1.2.1  
Deleted branch hotfix-1.2.1 [was abbe5d6].
```

## **Appendix B**

### **Setting Up the development environnement**

## B.1 Java application server setup

### B.1.1 Getting Glassfish

GlassFish can be downloaded from [GlassFish home](#). To download it, simply click on the link for your platform, The file should start downloading immediately. After the file finishes downloading, we should have a file called something such as `glassfish-v3-unix.sh`, `glassfish-v3-windows.exe`, or `glassfish-v3.zip`. The exact filename will depend on the exact GlassFish version and platform.

### B.1.2 Installing Glassfish server

Installing GlassFish is an easy process. However, GlassFish assumes some dependencies are present on your system.

#### GlassFish dependencies

In order to install GlassFish 3, a recent version of the Java Development Kit (JDK) must be installed on your workstation (JDK 1.6 or a newer version required), and the Java executable must be in your system path. The latest JDK can be downloaded from <http://java.sun.com/>. Please refer to the JDK installation instructions for your particular platform at:

<http://java.sun.com/javase/6/webnotes/install/index.html>.

#### Performing the installation

Once the JDK has been installed, installation can begin by simply executing the downloaded file (permissions may have to be modified to make it executable):

```
./glassfish-v3-unix.sh
```

The actual filename will depend on the version of GlassFish downloaded. The following steps need to be performed in order to successfully install GlassFish:

1. After running the previous command, the GlassFish installer will start initializing.
2. After clicking Next, we are prompted to accept the license terms.
3. The next screen in the installer prompts us for an installation directory. The installation directory defaults to a directory called `glassfishv3` under our home directory. It is a reasonable default, but we are free to change it.

4. The next page in the installer allows us to customize Glassfish's administration and HTTP ports. Additionally, it allows us to provide a username and password for the administrative user. By default, no username and password combination is required to log into the admin console. This default behavior is appropriate for development boxes. We can override this behavior by choosing to provide a username and password in this step in the installation wizard.
5. At this point in the installation, we need to indicate if we would like to install the GlassFish update tool. The update tool allows us to easily install additional GlassFish modules. Therefore, unless disk space is a concern, it is recommended to install it. If we access the internet through a proxy server, we can enter its host name or IP address and port at this point in the installation.
6. Now, we are prompted to either select an automatically detected Java SDK or type in the location of the SDK. By default, the Java SDK matching the value of the JAVA\_HOME environment variable is selected.
7. After installation finishes, we are asked to register our copy of GlassFish. At this point, we can link our GlassFish installation to an existing Sun Online Account, create a new Sun Online Account or skip installation.

### Verifying the installation

To start GlassFish, change the directory to [glassfish installation directory]/glassfishv3/bin and execute the following command:

```
./asadmin start-domain domain1
```

A few seconds after executing the previous command, we should see a message similar to the following at the bottom of the terminal:

```
Name of the domain started: [domain1] and  
its location: [/home/chedi/Application/glassfishv3/glassfish/domains/HyperPath].  
Admin port for the domain: [4848].  
We can then open a browser window and type the following URL in the browser's  
location text field: http://localhost:8080.
```



### B.1.3 Installing MySql Database JDBC connector

Download MySQL JDBC driver from <http://dev.mysql.com/downloads/connector/j/3.1.html> once Glassfish is installed you will need to make sure it can access MySQL Connector/J. To do this copy the MySQL Connector/J JAR file to the directory

`GLASSFISH_INSTALL/glassfish/lib`

and Restart the Glassfish Application Server.

You are now ready to create JDBC Connection Pools and JDBC Resources.

### B.1.4 Adding the JDBC resources

#### Creating a Connection Pool

- In the Glassfish Administration Console, using the navigation tree navigate to Resources, JDBC, Connection Pools.
- In the JDBC Connection Pools frame click New. You will enter a two step wizard.
- In the Name field under General Settings enter the name for the connection pool, for example enter MySQLConnPool.
- In the Resource Type field, select javax.sql.DataSource from the drop-down listbox.
- In the Database Vendor field, select MySQL from the drop-down listbox. Click Next to go to the next page of the wizard.
- You can accept the default settings for General Settings, Pool Settings and Transactions for this example. Scroll down to Additional Properties.
- In Additional Properties you will need to ensure the following properties are set:

**ServerName** The server you wish to connect to. For local testing this will be localhost.

**User** The user name with which to connect to MySQL.

**Password** The corresponding password for the user.

**DatabaseName** The database you wish to connect to.

- Click Finish to exit the wizard. You will be taken to the JDBC Connection Pools page where all current connection pools, including the one you just created, will be displayed.
- In the JDBC Connection Pools frame click on the connection pool you just created. Here you can review and edit information about the connection pool.
- To test your connection pool click the Ping button at the top of the frame. A message will be displayed confirming correct operation or otherwise. If an error message is received recheck the previous steps, and ensure that MySQL Connector/J has been correctly copied into the previously specified location.

Now that you have created a connection pool you will also need to create a JDBC Resource (data source) for use by your application.

### **Creating a JDBC Resource**

Your Java application will usually reference a data source object to establish a connection with the database. This needs to be created first using the following procedure.

- Using the navigation tree in the Glassfish Administration Console, navigate to Resources, JDBC, JDBC Resources. A list of resources will be displayed in the JDBC Resources frame.
- Click New. The New JDBC Resource frame will be displayed.
- In the JNDI Name field, enter the JNDI name that will be used to access this resource, for example enter jdbc/MySQLDataSource.
- In the Pool Name field, select a connection pool you want this resource to use from the drop-down listbox.
- Optionally, you can enter a description into the Description field.
- Additional properties can be added if required.
- Click OK to create the new JDBC resource. The JDBC Resources frame will list all available JDBC Resources.

### **B.1.5 Tuning security**

## B.2 Database setup

The MySQL database has become the world's most popular open source database because of its high performance, high reliability and ease of use.

### B.2.1 Installing the MySQL server

On Fedora the MySQL database installation is an easy process, you just have to:

- Use yum to install both mysql command line tool and the server: `yum install mysql mysql-server`
- Enable the MySQL: `service:/sbin/chkconfig mysqld on`
- Start the MySQL server: `/sbin/service mysqld start`
- Set the MySQL root password: `mysqladmin -u root password 'new-password'`<sup>1</sup>

Additionally you can install the <http://wb.mysql.com/mysql-workbench> which is a visual database design tool developed by MySQL and the highly anticipated successor application of the DBDesigner4 project.<sup>2</sup>:

```
yum install mysql-workbench
```

**Note:** If you plan not using both the database and the GlassFish server on the same machine, please adjust the firewall rules of the database machine so the 3306 port is enabled. In most cases, just issuing the command:

```
iptables -A INPUT -i eth0 -p tcp -m tcp -dport 3306 -j ACCEPT
```

will do the trick, if not you have to check the firewall rules.

### B.2.2 Creating the database schema

---

<sup>1</sup>The quotes around the new password are required.

<sup>2</sup>You will need the workbench if you would like to open the database diagram in the code repository

## B.3 Eclipse IDE setup

Eclipse is a popular open source Integrated Development Environment (IDE) that is widely adopted in the developer community.

### B.3.1 Installing Eclipse

Because we are both using Fedora Linux operating system, we do not require external resources to install the Eclipse IDE as it came bundled in the install DVD. Nevertheless, you need to select the following package in order to get it up and running:<sup>3</sup>

1. eclipse-changelog
2. eclipse-jdt
3. eclipse-rcp
4. eclipse-egit
5. eclipse-pde
6. eclipse-birt
7. eclipse-dtp
8. eclipse-swt

You can install them by using the command:

```
yum install [package name]
```

Or via the graphic interface for adding packages. In the case, you are not using **Fedora** or any other Linux variant you could always download the Eclipse application from the eclipse.org site. Exactly you need to download the "**Eclipse IDE for Java EE Developers**" available at this url:

<http://www.eclipse.org/downloads/packages/eclipse-ide-java-ee-developers/heliossr2><sup>4 5</sup>

---

<sup>3</sup>Some extra packages will be detected by the install system, but those listed here are the must have to compile and run this project

<sup>4</sup>This URL is relative to the current version of eclipse as, so make sure to go to the correct URL if it get updated

<sup>5</sup>This version include almost all the plugins needed to compile and run the application, we recommend you to add the git plugin to be able to download the code from GitHub

## B.3.2 GlassFish plugin

The GlassFish Tools Bundle for Eclipse is a software bundle that includes GlassFish Server v2.1 and v3 Prelude, Eclipse IDE 3.4 and the plug-in that enables them to work together. It provides a ready-to-use environment that helps you get started with developing applications on Eclipse and deploying them to GlassFish Server.

To Install on Linux Platform:

1. Go to **Help** → **Install** new software
2. Add new repository :  
<http://dlc.sun.com.edgesuite.net/glassfish/eclipse/helios>  
to the Available software sites and name it GlassFish or any convenient name.  
<sup>6</sup>
3. Select the newly created site from the "Work with" combobox. You should see them the GlassFish bundle.
4. Select the GlassFish plugin and continue to the end of the installation using next to the end.
5. Restart the IDE.

### Create a new GlassFish Server

The GlassFish Tools Bundle for Eclipse, by default, installs a new instance of GlassFish v2.1 Server and GlassFish v3 Prelude Server for you. If you want to add a new Server to Eclipse IDE, use the following procedure:

1. Select Servers view.
2. Select New, from Server popup menu. New Server page is displayed
3. Provide GlassFish Server details.
4. Click Next and complete Server configuration. For example, provide values for the following:
  - Domain Directory
  - Domain Name

---

<sup>6</sup>don't forget to change the last part of the url to your version of eclipse

- Administrator ID
  - Administrator password
5. Click Next. Add and Remove Projects page is displayed. Select the existing projects, you want to add to the new Server.
  6. Click Finish.

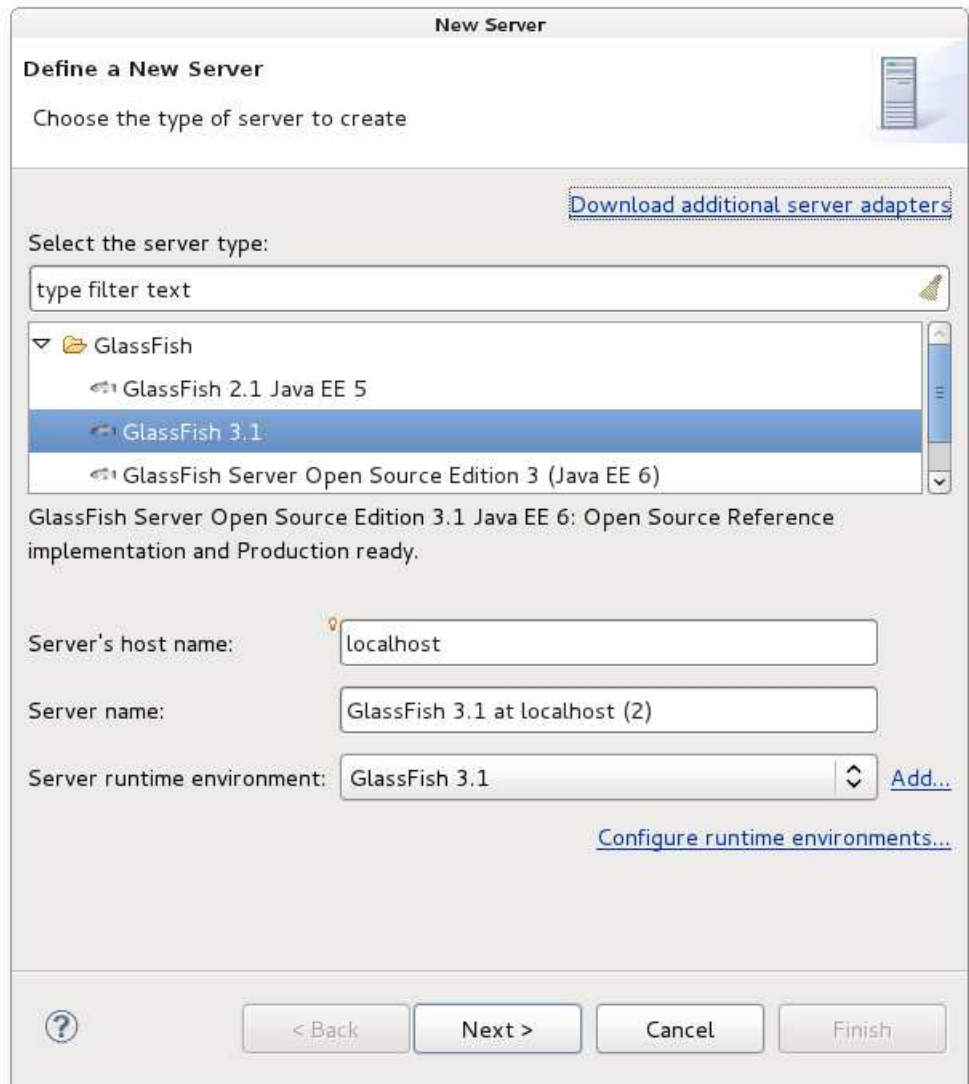


Figure B.1: Adding new server in Eclipse



### **To publish and clean Projects**

To publish applications to the GlassFish Server, use the following procedure:

1. Select Servers view.
2. Select GlassFish Server.
3. Select Publish from Server popup menu.

To discard all published projects, and republish from scratch, use the following procedure:

1. Select Servers view.
2. Select GlassFish Server.
3. Select Clean... from Server popup menu.
4. Click OK to accept the warning regarding discarding the published state.

### **To access GlassFish Server Administration Console**

GlassFish Sever offers many administrative features that are not accessible from Eclipse IDE. These features are available from GlassFish Server Administration Console which is accessible from Eclipse IDE. The following procedure describes the process of accessing GlassFish Server Administration Console, from Eclipse IDE.

<sup>7</sup>

1. Select Servers view.
2. Select Server popup menu → GlassFish Enterprise Server → GlassFish Administration Console.

---

<sup>7</sup>The administration console could be accessed via any navigator using the url <http://localhost:4848> (assuming that your GlassFish server is installed on the local machine, you could change it to the adequate server name if accessing from an outside location)

### **B.3.3 Creating the Eclipse Projects**

#### **The container project (EAR project)**

There are different Eclipse project types that can be used with the JEE 6. In the most flexible setup, you use a so-called Enterprise Application Project, also called an EAR Project, after the artifact it produces, an Enterprise Archive or EAR.

An EAR project is nothing but a container, intended to bundle the artifacts of the contained projects, along with any libraries that they use and that are not already part of the Java Enterprise Edition. We always begin with an EAR project, and then we create those projects, where we do the actual work, adding each of them to the EAR. Later, for deployment, we can export the EAR project as one single EAR file that contains all JARs from the contained projects.

In the JEE perspective you have the Project Explorer on the left side. If you have a fresh workspace, then the Project Explorer will be empty. Either from the File menu or from the Project Explorer context menu choose New / Enterprise Application Project. This opens the New EAR Application Project dialog.

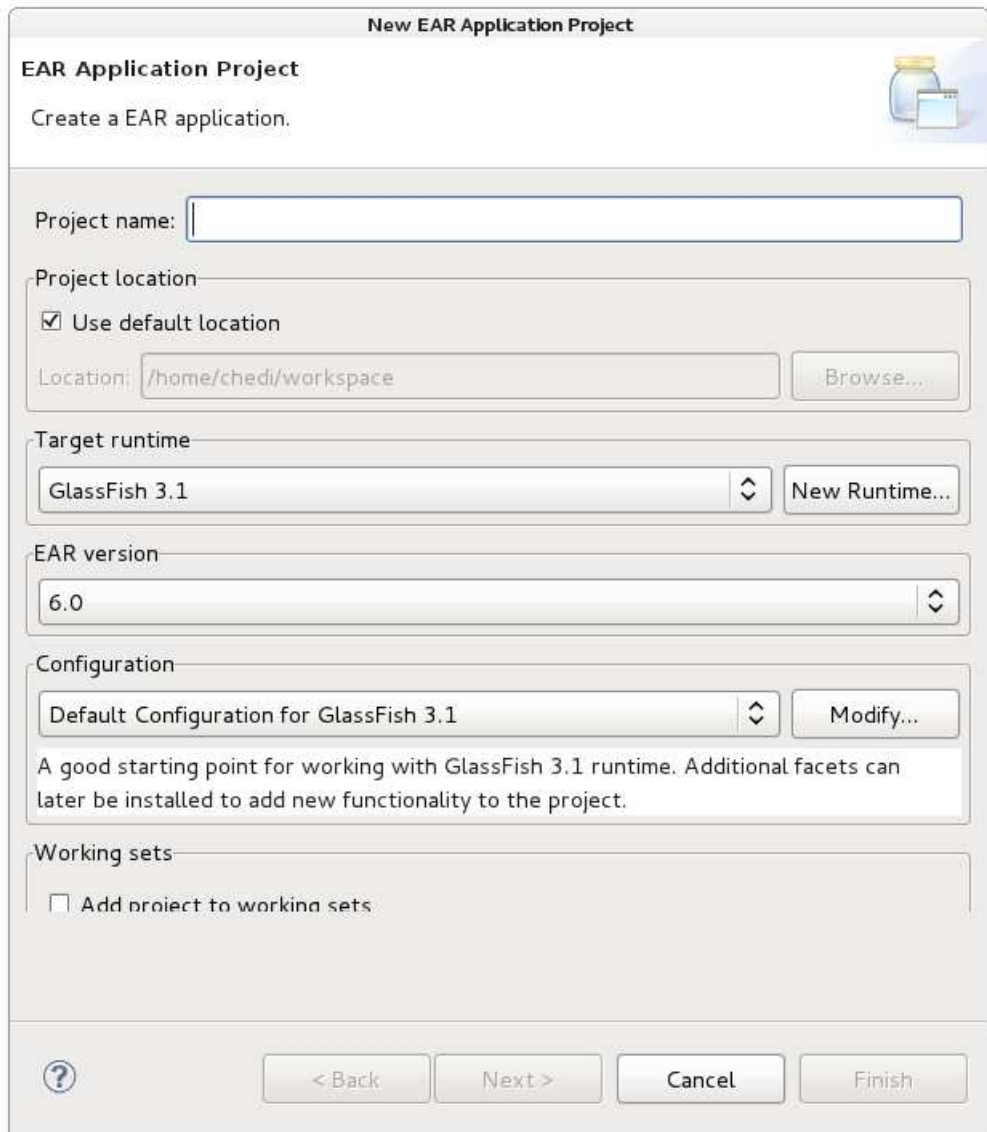


Figure B.2: Creating Eclipse enterprise project

## **The EJB project**

Enterprise Java Beans (EJB) are really just normal Java classes. It is when EJBs are instantiated under the control of an EJB container, that all sorts of interesting things begin to happen automatically.

For instance you can leave field variables in your beans uninitialized, and as long as they have the right annotations, the container injects proper instances, and for performance reasons it can do that from a pool of pre-allocated, momentarily idle instances.

One of the most important aspects of EJB containers is, that they can inject datasources (for instance JDBC datasources) and other resources, that the program only references by name, and that are really defined in the application server.

When you use JPA, an Entity is again a simple Java class with getters and setters for its fields, a POJO. You make it an entity by doing two things: annotating it with JPA annotations, and letting an EntityManager fetch it from or persist it to the database. You get such an EntityManager again via injection, when you get it, it is already associated with a persistence context that manages transactions and caches entities.

## **Appendix C**

### **Android simulator**

## **Appendix D**

### **Eclipse RCP to RAR**

## D.1 Introduction

The Eclipse Rich Ajax Platform (RAP) allows Java developers to build rich, Ajax-enabled Java applications that can be run on the desktop or the web from a single code base.

RAP uses the Eclipse development model, plug-ins with the Eclipse workbench extension points and a widget toolkit with SWT API. Which means that existing Eclipse RCP applications can be run as web applications with just a few modifications.

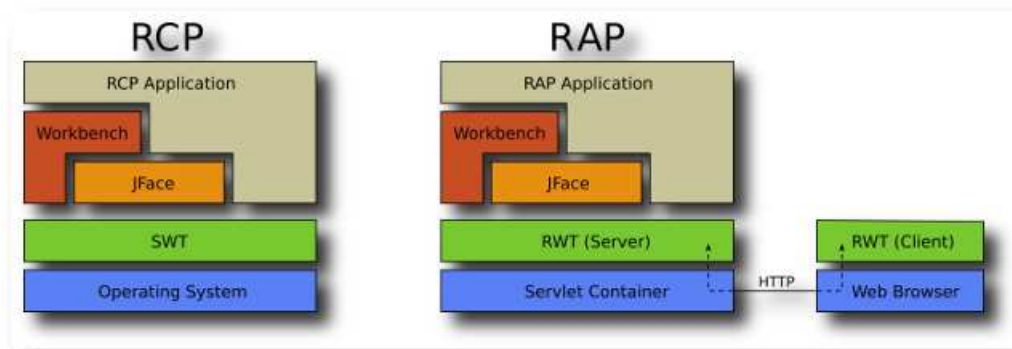


Figure D.1: Eclipse RAP architecture

## D.2 Advantages

### D.2.1 Inter browser compatibility

RAP runs out of the box in all common web browsers. No browser add-ons are required. The server part of RAP can be deployed on all Servlet Containers that implement the Servlet API 2.3 through 3.0. This includes Tomcat, Jetty, Glassfish, JBoss and WebSphere.

### D.2.2 Single sourcing

A popular use case for RAP is the development of rich clients and web clients from a single code base, also called "Single Sourcing". This allows Java and Eclipse

developers to reuse their existing skills. Furthermore, RAP maximizes code reuse by including the largest-possible web-enabled subset of the Rich Client Platform.

*“Write once, run everywhere is the main objective of the Java<sup>TM</sup> programming language ”[11].*

This holds true most of the time for running the same program on different operating systems. But as technology evolves the Eclipse Foundation strives to bring this slogan to a new level. Two technology projects, housed under the Eclipse.org umbrella, namely RAP2 and eRCP3, provide an alternative runtime environment for RCP applications.

This means that the application - normally running as a desktop application on a personal computer - can now be deployed to other runtime environments. For example let us assume we have a ready to launch application based on RCP. By switching the runtime from RCP to RAP, it becomes possible to launch the application on an application server where clients can use the application with a Web 2.0 centric interface on a browser.

There is no need for the user to install any further add-ons or plugins. In order to achieve full compatibility between the platforms, many concepts implemented in SWT [13] need to be adapted to other runtimes. These are hidden behind the public API which remains synchronous across all runtime projects. It may also be possible to change the structure of the application code to fit the current runtime environment. Depending on the specific case, this may be achievable through normal refactorings. The refactorings are not onerous and generally lead to better quality bundles irrespective of the single-sourcing requirement.

### **D.2.3 Integration with BIRT**

Besides a rich user interaction many applications need to display a big amount of data sets as diagrams or reports as part of their applications. In order to bridge the gap the BIRT project was created as part of the eclipse ecosystem.

That BIRT integrates well with classic RCP applications is a well known fact. But the need for rich internet applications is still growing. And here the RAP comes into play. As a platform for developing Web 2.0 applications with the same patterns as for RCP it paves the way for single sourcing applications running on both platforms.



# Bibliography

- [1] allaboutgeomarketing. allaboutgeomarketing.com. <http://www.allaboutgeomarketing.com>, 2011.
- [2] Scott chacon. *Pro Git*. Apress, 2009.
- [3] eclipsesource.com. What is eclipse rap. <http://eclipsesource.com/en/eclipse/eclipse-rap-overview/>, 2011.
- [4] Geoconcept. Géomarketing. <http://www.geoconcept.com/Geomarketing.html>, 2011.
- [5] Geoconcept. Géomarketing. <http://www.geoconcept.com/Geomarketing.html>, 2011.
- [6] Geoconcept. Zone de chalandise. <http://www.geoconcept.com/Zone-de-chalandise.html>, 2011.
- [7] geomarketingafrica. what is geomarketing. <http://www.geomarketingafrica.com/what-is-geomarketing.html>, 2011.
- [8] gitref.org. Git reference. <http://gitref.org/>, 2011.
- [9] David Heffelfinger. *Java EE 6 with GlassFish 3 Application Server*. Packt Publishing, 2010.
- [10] Patrick W. Daly Helmut Kopka. *A Guide to L<sup>A</sup>T<sub>E</sub>X and Electronic Publishing*. Addison Wesley, Massachusetts, 2004.
- [11] Tim Lindholm Frank Yellin Frank Yellin The Java Team Mary Campione Kathy Walrath Patrick Chan Rosanna Lee Jonni Kanerva James Gosling James Gosling James Gosling James Gosling Bill Joy Bill Joy Guy Steele

Guy Steele Gilad Bracha Ken Arnold, David Holmes and Gilad Bracha. *Java language specification, third edition*. Prentice Hall PTR, 2005.

- [12] Jon Loeliger. *Version Control with Git*. O'Reilly Media, Inc, 2009.
- [13] Steve Northover and Mike Wilson. *SWT: the standard widget toolkit*. Addison-Wesley, 2004.
- [14] Dai Pham. Smartphone user study shows mobile movement under way. <http://googlemobileads.blogspot.com/2011/04/smartphone-user-study-shows-mobile.html>, 2011.
- [15] Sun Microsystems. *GlassFish Tools Bundle for Eclipse User Guide*, 2009.
- [16] Vikram Vaswani. *MySQL Database Usage and Administration*. The McGraw-Hill Companies, 2009.
- [17] Kevin Werback. Location-based computing: Wherever you go, there you are. Technical Report 18, EDventure Holdings Inc., June 2000. <http://www.edventure.com>.
- [18] wiki.eclipse.org. Egit/user guide. [http://wiki.eclipse.org/EGit/User\\_Guide](http://wiki.eclipse.org/EGit/User_Guide), 2011.
- [19] wiki.eclipse.org. Rap/birt integration. [http://wiki.eclipse.org/RAP/BIRT\\_Integration](http://wiki.eclipse.org/RAP/BIRT_Integration), 2011.
- [20] wikipedia.org. Git. [http://en.wikipedia.org/wiki/Git\\_\(software\)](http://en.wikipedia.org/wiki/Git_(software)), 2011.
- [21] wikipedia.org. Revision control. [http://en.wikipedia.org/wiki/Revision\\_control](http://en.wikipedia.org/wiki/Revision_control), 2011.
- [22] wikipedia. Geo marketing. [http://en.wikipedia.org/wiki/Geo\(marketing\)](http://en.wikipedia.org/wiki/Geo(marketing)), 2011.
- [23] wikipedia. Location-based service. [http://en.wikipedia.org/wiki/Location-based\\_service](http://en.wikipedia.org/wiki/Location-based_service), 2011.

# Index

Android, 8

History, 8

Why Android?, 8

BIRT

Intergration in RAP applications, 65

Introduction, 13

Eclipse

RAP, 64

Integration with BIRT, 65

Inter browser compatibility, 64

Single sourcing, 64

RCP

Introduction, 10

RAP, 64

SWT

History, 10

GlassFish

Advantages, 12

Introduction, 12

MySQL

Advantages, 15

Data warehousing, 16

Development, 16

Freedom, 16

Management, 16

Performance, 15

Scalability, 15

Introduction, 15