Rem - Requirements and Entity Modeler Open-source, cross-platform UML editor written in C++, including Jacobson's extensions for Aspect-Oriented Software Development, and optimised for Model-Driven Architecture tasks.

by

Adrian Kosmaczewski

A Dissertation

Submitted to

The University of Liverpool

in partial fulfillment of the requirements

for the degree of

Master of Science

August 2008

Abstract

Rem - Requirements and Entity Modeler

Open-source, cross-platform UML editor written in C++, including Jacobson's extensions for Aspect-Oriented Software Development, and optimised for Model-Driven Architecture tasks.

Adrian Kosmaczewski

Supervisor: Dr. Shakil Ahmed

The **Rem** project aims to the creation of a cross platform (Linux, Windows & Mac OS X) UML(Booch, Jacobson & Rumbaugh 1998) tool, written in standard C++, suitable both for academia and the industry, released as an open source project, featuring a high-quality code base, thoroughly unit- and functionally tested, with a uniform, standard and easy-to-use user interface, and including extensions for Aspect-Oriented Software Development (AOSD)(Ng & Jacobson 2005).

The market of cross-platform UML editors is mostly held by software applications run-

ning on the Java Virtual Machine, often showing unstable behaviours, incompatible file formats, or a poor user experiences. All of these issues make the use of UML an unpopular and often painful experience altogether.

Even worse, most UML editors do not directly support AOSD constructs or technologies such as MDA (Warmer, Bast, Pinkley, Herrera & Kleppe 2003), even if they allow the use of stereotypes or tagged values to model them. To enhance the widespread of AOSD and MDA, a suitable tool is needed with direct support for both techniques. Finally, most of the non-Java based ones are not available in many operating systems at once, usually only supporting one or two at most.

The main objective is to create an UML editor in standard C++, to be used both in academia and in real-world projects, to be released afterwards as an open source project, running natively in the three major operating systems, and using a cross-platform file format.

Regarding its scholarly contributions, the project will innovate in its support for two important trends of the first decade of the 21st century, namely Aspect Orientation and Model-Driven Architectures. The final outcome of the project will benefit the whole community software engineers around the world to tackle more complex software projects, to better communicate among them, and to extend the platform in unforeseen ways.

The project will also highlight the complexities inherent to any cross-platform project, given that the final outcome of this project will run in at least 3 different operating systems, in 2 different and a priori incompatible processor architectures (x86 and PowerPC).

The open source nature of various of its components will finally show the feasibility of such a project in a relatively short amount of time, with minimalist resources, using free, open source and standard-based technologies whenever possible.

Declaration

I hereby certify that this dissertation constitutes my own product, that where the language of others is set forth, quotation marks so indicate, and that appropriate credit is given where I have used the language, ideas, expressions, or writings of another.

I declare that the dissertation describes original work that has not previously been presented for the award of any other degree of any institution.

Signed,

[illegible]

Adrian Kosmaczewski

Acknowledgments

The author would like to thank Dr. Shakil Ahmed for his guidance and support during the nine months of this dissertation, as well as during the “Object Oriented Programming and Design using C++” module. This project constitutes a complex exploration in the realm of cross-platform and standard C++, and this project would not have seen the daylight without his help and remarks. The support received by all members of the team from the University of Liverpool was an invaluable help in making this dissertation a reality; the author thanks each and every one of them for their time, remarks and guidance.

The author also wishes to thank the huge open-source community, without whom the whole project would have simply been impossible to create in such a short schedule:

- Julian Storer, author of the JUCE C++ GUI library[1], an amazing piece of software, extremely well documented and supported, which made the whole project so much easier to create and extend;
- Bill Hoffman, Ken Martin, Brad King, Dave Cole, Alexander Neundorf and Clinton Stimpson from the CMake project[2], without which porting **Rem** to Windows and Linux would have been a much more complex task;
- Amir Szekely, author of the NSIS project[3], for providing the community a strong

[1]http://www.rawmaterialsoftware.com/
[2]http://www.cmake.org/HTML/participants.html
[3]http://nsis.sourceforge.net/News

tool for installing and de-installing software on Windows;

- Guenter Obiltschnig, Peter Schojer and Aleksandar Fabijanic from the POCO C++ Library project[4], a strong, portable and lightweight foundation library which provided essential functionality to **Rem**;
- D. Richard Hipp, creator of the SQLite library[5], one of the most wildly popular and most widely used relational database management systems of all time;
- Ben Collins-Sussman, Brian W. Fitzpatrick and C. Michael Pilato, authors of the Subversion source code management tool[6] (and to the whole team responsible for its maintenance[7]), a fast and cross-platform tool which streamlined the development of **Rem** in unforeseen ways; moreover, Collins-Sussman and Fitzpatrick maintain the "Google Code" system infrastructure that hosts **Rem**;
- Dimitri van Heesch, creator of Doxygen[8], a code document extraction tool which provided great help to the author while creating **Rem**;
- Michael Feathers, creator of CppUnit[9], a cross-platform, solid unit-testing suite which helped stabilize and boost the development of the lower layers of **Rem**;
- Dave MacLachlan and others responsible for the CoverStory code coverage tool[10], which helped the author increase the breadth of unit testing.

Finally, in a more personal note, the author thanks his wife Claudia for her love, and the outstanding support, patience, understanding, and endless flow of coffee cups she provided during the three years of this Master's Degree program.

[4]http://pocoproject.org/poco/info/contributors.html
[5]http://www.hwaci.com/drh/index.html
[6]http://svnbook.red-bean.com/en/1.4/svn.preface.acks.html
[7]http://www.red-bean.com/svnproject/contribulyzer/
[8]http://www.stack.nl/~dimitri/doxygen/
[9]http://cppunit.sourceforge.net/cppunit-wiki
[10]http://code.google.com/p/coverstory/

To Claudia.

No happiness would ever be possible without you.

Contents

List of Tables

List of Figures

List of Code Fragments

Chapter 1

Introduction

This chapter provides a high-level overview of the **Rem** project, its objectives, constraints and major requirements.

1.1 Objectives

The **Rem** project aims to the creation of a cross platform (Linux, Windows & Mac OS X) UML[1] tool, written in standard C++, suitable both for academia and the industry, released as an open source project, featuring a high-quality code base, thoroughly unit- and functionally tested, with a uniform, standard and easy-to-use user interface, and including Ng & Jacobson (2005) extensions for Aspect-Oriented Software Development (AOSD).

1.2 Rationale

The market of cross-platform UML editors is mostly held by Java-based software applications, often showing unstable behaviours and incompatible file formats. Users of MagicDraw cannot safely load their diagrams into Poseidon UML, even if both packages claim to use the same standard for saving data. Not to mention the trouble to get some of these applications to work at all, given the high requirements and slow response of

[1]Booch et al. (1998)

the Java VM, and the poor user experience. All of these issues make the use of UML an unpopular and often painful experience altogether.

Even worse, most UML editors do not directly support AOSD constructs or technologies such as MDA - Warmer et al. (2003) -, even if they allow the use of stereotypes or tagged values to model them. To enhance the widespread of AOSD and MDA, a suitable tool is needed with direct support for both techniques. Finally, most of the non-Java based ones are not available in many operating systems at once, usually only supporting one or two at most.

The main objective is to create an UML editor in standard C++, to be used both in academia and in real-world projects, to be released afterwards as an open source project, running natively in the three major operating systems, and using a cross-platform file format.

The system should prove of utility to those designing systems using AOSD techniques, allowing them to model aspects, pointcuts and advices directly in use case, class and sequence diagrams, and also providing a code-generation API making easier for teams to work on big software projects, allowing them to automatize code generation tasks.

1.3 Name

The name **Rem** is not only the acronym for “Requirements and Entity Modeler”, but also a reference to Rem Koolhaas[2], one of the most important contemporary urbanists and architects.

Koolhaas’ vision, ideas and books (such as “S,M,L,XL”) have had a tremendous impact on architects and urbanists worldwide, and the **Rem** project vision is to have a lasting, positive impact in the software architecture activity as well.

The application icon, shown in figure 1.1, is a stylized representation of one of Koolhaas’ major works, the building of the Seattle Public Library[3].

[2] http://architecture.about.com/library/weekly/aa042200a.htm
[3] http://www.spl.org

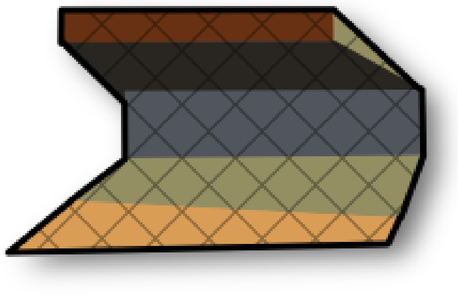


Figure 1.1: **Rem** Icon

1.4 Goals

Besides providing a functioning piece of software, the project will innovate in its support for two important trends of the first decade of the 21st century, namely Aspect Orientation and Model-Driven Architectures. The final outcome of the project will benefit the whole community of software engineers around the world to tackle more complex software projects, to better communicate among them, and to extend the platform in unforeseen ways.

The project will also highlight the complexities inherent to any cross-platform project, given that the final outcome of this project will run in at least 3 different operating systems (two versions of Unix plus at least one version of Windows), in 2 different and a priori incompatible processor architectures (x86 and PowerPC).

The open source nature of various of its components will finally show the feasibility of such a project in a relatively short amount of time, with minimalist resources, using free, open source and standard-based technologies whenever possible.

1.5 Scope

This project, as the final step towards the obtention of a Master's degree, has the following scope:

- Analysis & design of the system;
- Development of the application;

- Testing and quality management procedures;
- Release of the application to the public.

1.6 Project Outcome

The project will provide the following deliverables:

- Three binary versions of the application, ready to be used “out of the box” in the following operating systems:
  - Kubuntu Linux 7.10;
  - Mac OS X 10.5 “Leopard” (as a “Universal Binary”, for Intel and PowerPC, for both 32 and 64 bits architectures, packaged as a “DMG”, or disk image file);
  - Windows XP Service Pack 2 (including an “installer” for quick and easy installation and uninstallation of the application).
- The source code of the application, including building instructions, scripts and project files, highlighting dependencies and other requirements prior to building the software from scratch:
  - Makefiles for Linux;
  - Xcode project file for Mac OS X;
  - Visual C++ Express 2008 project file for Windows.
- The test suites for the source code, including:
  - The CppUnit unit code suites;
  - The functional unit test scripts.
- The complete API documentation, extracted from the source code using Doxygen.
- A project website, split in two parts:
  - A project page in the Google Code repository (http://code.google.com/) with the Subversion repository, wiki pages and other relevant information about the project;

  - A proper project website, whose URL will match that of the UML software.

1.6.1 Evaluation Criteria

The quality of the final deliverables will be evaluated with the following criteria:

- The application should run flawlessly in the three supported platforms, and the files generated in one platform should be fully readable in other platforms;
- Most AOP constructions should be supported;
- The tool should export code in at least 3 major programming languages.

1.7 Time Constraints

The final deadline for the **Rem** project is August 28th, 2008. Development of the system began on March 1st, 2008, and the first prototypes were available in June.

1.8 Similar Projects

There are similar systems available in the market, both as commercial and free software packages:

1.8.1 Commercial UML Tools

This is a short list of the most relevant commercial UML tools available at the moment of writing:

- IBM Rational Rose[4]
- Borland Together[5]
- MagicDraw UML[6]

[4] http://www.ibm.com/software/rational/
[5] http://www.borland.com/us/products/together/
[6] http://www.magicdraw.com/

- Gentleware Poseidon UML[7]
- Altova UModel[8]
- Visual Paradigm for UML[9]
- ARIS UML Designer[10]
- Sparx Systems Enterprise Architect[11]

1.8.2 Free Software UML Tools

In the world of “free software”[12] several other systems are also available. Here is a short list of the most important ones:

- ArgoUML[13]
- Umbrello[14]
- Dia[15]
- ObjectPlant[16]
- Fujaba Tool Suite[17]
- UMLGraph[18]
- BOUML[19]

The **Rem** project will have to compete in a crowded market; nevertheless, it will have the following distinctive characteristics:

- Support for AOSD constructions “off-the-box”;

[7] http://www.gentleware.com/products.html
[8] http://www.altova.com/products/umodel/uml_tool.html
[9] http://www.visual-paradigm.com/
[10] http://www.ids-scheer.com/
[11] http://www.sparxsystems.com.au/ea.htm
[12] Using Richard Stallman’s definitions, “free software” means “free” as in “free speech” but also as in “free beer”.
[13] http://argouml.tigris.org/
[14] http://uml.sourceforge.net/
[15] http://www.gnome.org/projects/dia/
[16] http://www.arctaedius.com/ObjectPlant/
[17] http://www.fujaba.de/
[18] http://www.umlgraph.org/
[19] http://bouml.free.fr/

- Extensible architecture, for providing new export formats (particularly programming languages);
- Support for three major operating systems since the first version, running at native speed in each one, without the need nor the overhead of a bulky virtual machine behind.

1.9 General Requirements

Through **Rem**, users will be able to perform the following tasks:

- Create use case, sequence and class diagrams;
- Design systems including AOSD extensions, with aspects, pointcuts and advices;
- Export diagrams using the XMI standard;
- Use the system in different platforms (Linux, Windows & Mac OS X) with a similar "look and feel";

1.10 Technical Requirements

Rem will have the following requirements and dependencies:

1.10.1 Standards

Rem will support major standards whenever applicable, including, but not limited to:

C++ or ISO/IEC 14882:2003;

XML Metadata Interchange or ISO/IEC 19503:2005, for data exchange with other UML tools;

Portable Network Graphics or ISO/IEC 15948:2003, for image exports of diagrams;

Other ISO standards where applicable, such as Unicode (ISO/IEC 10646) or date and time formats (ISO 8601).

About ISO C++

Particularly, to ensure that the C++ code has the strongest ISO support, at least three different compilers will be used to compile the source code: two versions of the GNU Compiler Collection gcc[20] on Mac OS X and Linux, and Microsoft's C++ compiler, available as part of the Visual C++ 2008 Express Edition development environment.

Support for other (free) C++ compilers (such as the Intel C++ Compiler 10.1[21] or the Open Watcom 1.7a[22] compiler) is not required.

1.10.2 Software

To run properly, **Rem** will not require particular software or hardware already installed in the end user machine, other than the operating system itself.

1.10.3 Hardware

Rem will not have special or particular hardware requirements other than those of the platform components.

1.10.4 Operating Systems

As previously stated, **Rem** will be guaranteed to run natively in the following operating systems:

1. Microsoft Windows XP Service Pack 2
2. Mac OS X 10.5 “Leopard”
3. Ubuntu Linux 7.10 “Gutsy Gibbon”

Even if **Rem** might be able to run in other versions of these three operating systems (particularly Windows Vista and older versions of Mac OS X, Ubuntu and other Linux distributions), given the short amount of time, there will be no direct support for them right now.

[20] http://gcc.gnu.org/
[21] http://www.intel.com/cd/software/products/asmo-na/eng/compilers/clin/277618.htm
[22] http://www.openwatcom.org/index.php/Main_Page

1.10.5 Application software

To develop **Rem**, the following tools will be used:

IDEs: Xcode[23], Microsoft Visual C++ 2008 Express Edition[24], KDevelop[25] and several others text editors, like TextMate[26].

SCM: Subversion[27]

File storage: SQLite[28]

Unit testing: CppUnit[29]

GUI toolkit: Juce[30]

Build: CMake[31]

Installation wizard: NSIS[32]

Code Documentation: Doxygen[33]

Some of these tools are all open-source, free and run both in Windows and Unix environments; this will help reducing the development costs, since any number of modern computer will be able to be set up as development platform, and no license fees are dues. In the other cases, the use of commercial tools will be encouraged where productivity and integration are required.

1.11 Training and Documentation

Due to the lack of time, the application will not be delivered with end-user documentation like help files. However, developers interested in enhancing the system will be able to access a wiki with relevant information.

[23] http://developer.apple.com/tools/xcode/
[24] http://www.microsoft.com/express/vc/
[25] http://www.kdevelop.org/
[26] http://macromates.com/
[27] http://subversion.tigris.org/
[28] http://www.sqlite.org/
[29] http://cppunit.sourceforge.net/
[30] http://www.rawmaterialsoftware.com/juce/
[31] http://www.cmake.org/
[32] http://nsis.sourceforge.net/
[33] http://doxygen.org/

1.12 Installation

Rem will have different installation methods, depending on the platform used; in each one, the following standard methods will be supported:

Windows: Through an installer.

Mac OS X: Through manual installation, using “DMG” (Disk Image) files, as is usual in this platform.

Linux: Through the “cmake — make” sequence used with the CMake build tool.

1.13 Communication and Visibility

Rem is a project geared towards visibility and communication. It will feature several different but complementary mechanisms, showing the project activity and current status at any time. This section presents the chosen mechanisms for reporting on the project progress:

- Embanet
- Project Website
- Project Documentation Wiki
- Issue Database
- Source Code Browser
- Project Blog and Twitter
- Project Forum

1.13.1 Embanet

Embanet will be the most important communication medium during the creation of **Rem**. It centralizes all the communication between the author and his Dissertation Advisor and Sponsor.

1.13.2 Project Website

Rem will have a strong web presence, offering complementary services during the development of the project. On one side, the main project website located at http://remproject.org/. On the other side, a Google Code repository at http://code.google.com/p/remproject/. Both will link to each other, offering a coherent set of information to users interested in using **Rem** or collaborating in the **Rem** project.

1.13.3 Project Documentation Wiki

The documentation wiki for the project will be hosted in the Google Code page, at this address: http://code.google.com/p/remproject/w/list. It will feature up-to-date information about the project, such as installation procedures, answers to common problems (in the form of a "F.A.Q." or "Frequently Asked Questions" list), guidelines for developers, and other useful information.

1.13.4 Issue Database

The bug database is also hosted in the Google Code page, and is the basis of all the quality management of the project. It will feature not only known bugs, but also "to-do" items, with a complete historic overview of each issue, how and when it has been solved.

1.13.5 Source Code Browser

Users interested in knowing the internal structure of **Rem** will be able to browse the source code directly on line, thanks to the Google Code infrastructure, in this location: http://code.google.com/p/remproject/source/browse. This browser allows user to see individual changesets, past revisions of a single file, and checkout a working copy of the system at any time.

1.13.6 Project Blog and Twitter

Hosted at http://remproject.org/, the project blog will feature important announcements about new releases and other news. The **Rem** Twitter page at http://twitter.com/remproject will feature instant announcements, typically new releases or otherwise important information usually not worth a complete blog posting.

1.13.7 Project Forum

There will be a forum at http://remproject.org/forum/ for users to ask questions, get feedback from other users or from the author, and to serve as a complementary, user-driven resource for the documentation.

1.14 Quality Assurance

Rem will be an important tool in the daily activities of software developers; as such, it is extremely important to provide a high-quality tool, documented and extensible, and having resilience to crashes and data loss problems. This sectino will provide an overview on the tasks that will ensure a reasonably high level of quality in the **Rem** system.

1.14.1 Guidelines

Rem's quality assurance procedures will follow these guidelines:

- Follow most existing standard best practices;
- Establish and apply coding conventions;
- Use Test-Driven Development (TDD) techniques;
- Use the code coverage tools to know the percentage of the code covered by tests;
- Use a mature source code control tool to manage versioning;
- Use several compilers;
- Use a bug tracking database system;

- Using build process automatisation;
- Document and extract the public APIs of the project.

1.14.2 Metrics

Rem features strong functional requirements:

- The application should run flawlessly in all supported platforms, and the files generated in one platform should be fully readable in other platforms;
- Most AOP constructions should be supported;
- The tool should export code in at least 3 major programming languages.

Given these requirements, the short deadline, the size of the team and the dynamicity of the development, the **Rem** project will be monitored using the following metrics:

Project progress: These metrics are rather simple, such as respect of milestones, number of completed features, or the project size in LOC. To do this, the Ohcount utility[34] will be used.

Test coverage: When writing unit tests, it is important to know the relative proportion of methods and functions that are being tested, the ideal being a 100% test coverage. The gcov utility[35] provides such metrics.

Documentation: The API documentation will be extracted with Doxygen[36] at each new system build.

1.14.3 Quality Assurance Principles

This section will outline general principles to ensure the highest possible quality for **Rem**.

[34]Ohcount is an open source code line counter, found at http://labs.ohloh.net/ohcount

[35]gcov is part of the GNU system, and its documentation can be found here: http://gcc.gnu.org/onlinedocs/gcc/Gcov.html

[36]http://doxygen.org/

Coding Guidelines

The **Rem** project will follow closely two different coding guidelines:

1. A custom-generated one, using the Coding Standard Generator at http://www.rosvall.ie/CSG/ which can be found in section 5.4 of this document;

2. “High Integrity Coding Practices”, found at http://www.codingstandard.com/.

These guidelines are both based in previous books and articles about “best practices” for C++ develoment, for example those mentioned by Meyers (2005) or more recently by Stephens, Diggins, Turkanis & Cogswell (2005). Duffy (2004) also has a whole section of his book dedicated to the creation of strong C++ frameworks at the beginning of a project.

Comments and Inline Documentation

All code members (classes, namespaces, fields, methods, files) will feature standard comments, that will be extracted during the build procedure with the Doxygen[37] tool, to be offered as a separate download to developers interested in contributing to the project.

Issue Database

The Google Code infrastructure provides an issue database, useful for tracking known problems throughout the system, at this address: http://code.google.com/p/remproject/issues/list.

Unit Testing

Together with the application source code, a complete set of unit tests (targeting the CppUnit[38] unit test suite) will be provided. These unit tests will ensure that every sub-system of **Rem** performs its duties flawlessly, and will prove of great value before doing

[37] http://doxygen.org/
[38] http://cppunit.sourceforge.net/

refactoring - as mentioned by Fowler, Beck, Brant, Opdyke & Roberts (1999) -, new features development, and general maintenance.

1.14.4 Source Control System

The source code of the system **Rem** system will be stored using the Subversion versioning source control system; the Subversion repository will be hosted using Google Code's hosting capabilities, in an off-site fashion, with strong backup and security policies.

Chapter 2

Background and review of literature

To create **Rem**, a number of literary resources were used, in different fields and areas. This chapter provides an overview of the main points of reference used in the creation of this software.

2.1 UML Diagramming

2.1.1 Introduction

UML stands for “Unified Modeling Language”; it is defined as a “standard language for writing software blueprints” as described by Booch et al. (1998, page 13). The UML was created for visualizing, specifying, constructing and documenting software artifacts built in any object-oriented software building environment.

Bell (2003*a*) explains that the UML was jointly created by three experts in the field of object-oriented software engineering, Jim Rumbaugh, Ivar Jacobson, and Grady Booch, who joined their efforts and individual notation systems in a common, “unified” modeling language.

2.1.2 Types of UML Diagrams

There are 13 types of UML diagrams available in version 2.0 of the standard, which can be grouped in two basic groups, as classified by Ambler (2007):

- Behavior Diagrams
  - Activity
  - State Machine
  - Use Case
  - Interaction
    - Communication
    - Interaction
    - Sequence
    - Timing
- Structure Diagrams
  - Class
  - Composite Structure
  - Component
  - Deployment
  - Object
  - Package

2.1.3 The UML and Rem

The primary use case of **Rem** is to generate (in its first release) three different kinds of UML diagrams, namely

- Use Case Diagrams
- Class Diagrams

- Sequence Diagrams

The choice of these three diagram types has to do with the need to define a small feature set for this project, with the personal experience of the author, who has witnessed the use of these 3 diagrams as the most common pattern in most software projects, and finally with the opinion of Ambler (2007) who recommends learning of these types of diagrams as part of the basic UML skillset.

Use Case Diagrams

"Use Case" diagrams serve to model dynamic aspects of systems, and have a primary role as conveyors of the behavior of the system, from the point of view of the user.

Use Case diagrams are commonly used during requirements capture and documentation, as described by Bennett (2005, pages 146-155). The friendly layout of Use Cases, easily understandable by both technical and non-technical users, make them a handy and easy-to-use tool to document the functional requirements of a system.

This characteristic is one of the reasons why **Rem**, as an acronym, stands for "Requirements and Entity Modeler".

Class Diagrams

"Class" diagrams show a static view of the system, based on entities such as classes, interfaces, as well as their various relationships, either inheritance, collaboration or other, as Booch et al. (1998, page 107) explains. Bell (2004*a*) further states that these diagrams are those that most closely translate to source code, since they represent the common entity set described by any object-oriented programming language.

Class diagrams were first proposed by Booch (1993, pages 171-199), and an early version of these, including support for "parameterized classes" (later known as "templates"), can be seen in his seminal book "Object-Oriented Analysis and Design with Applications".

Although simple to understand, Bell (2003*b*) states that not all software developers are able to clearly distinguish some subtile details, and it is as such recommended to learn about in detail.

Sequence Diagrams

Finally, “Sequence” diagrams are explained by Booch et al. (1998, page 246) as required to explicitly describe different scenarios during the lifetime of an instance or group of instances during runtime.

However, Bell (2004*b*) indicates that the diagram can also be used by business experts to document complex interactions between different entities, in a non-technical environment.

2.2 Aspect-Oriented Programming

2.2.1 Definition

Aspect-Oriented Programming (AOP) and Software Development (AOSD) is a relatively new trend in software engineering, described by Kiczales, Lamping, Mendhekar, Maeda, Lopes, Loingtier & Irwin (1997, page 4) as an orthogonal approach to both the procedural and object-oriented paradigms. AOP aims to centralize the definition and implementation of “aspects” or elements from software which are pervasive throughout systems, such as instrumentation, security management or resource management, allowing developers to dynamically modify the behavior of production systems after deployment in a controlled way, and reducing complexity and maintenance costs.

As O’Regan (2004) explains, there are four core concepts in AOP:

Cross-cutting concerns: Even though most classes in an OO model will perform a single, specific function, they often share common, secondary requirements with other classes. For example, we may want to add logging to classes within the data-access layer and also to classes in the UI layer whenever a thread enters or exits a method. Even though the primary functionality of each class is very different, the code needed to perform the secondary functionality is often identical.

Advice: This is the additional code that you want to apply to your existing model. In our example, this is the logging code that we want to apply whenever the thread enters or exits a method.

Point-cut: This is the term given to the point of execution in the application at which cross-cutting concern needs to be applied. In our example, a point-cut is reached when the thread enters a method, and another point-cut is reached when the thread exits the method.

Aspect: The combination of the point-cut and the advice is termed an aspect. In the example below, we add a logging aspect to our application by defining a point-cut and giving the correct advice.

To illustrate the problem of code modularity and cross-cutting concerns, Kiczales & Hilsdale (2003, slides 4 and 5) show an example based on the Apache Tomcat project[1]; figure 2.1 shows an example of well-modularized code, namely the URL pattern matching code, however figure 2.2 shows an example of a cross-cutting concern, namely logging, whose code is scattered all over the source code base.

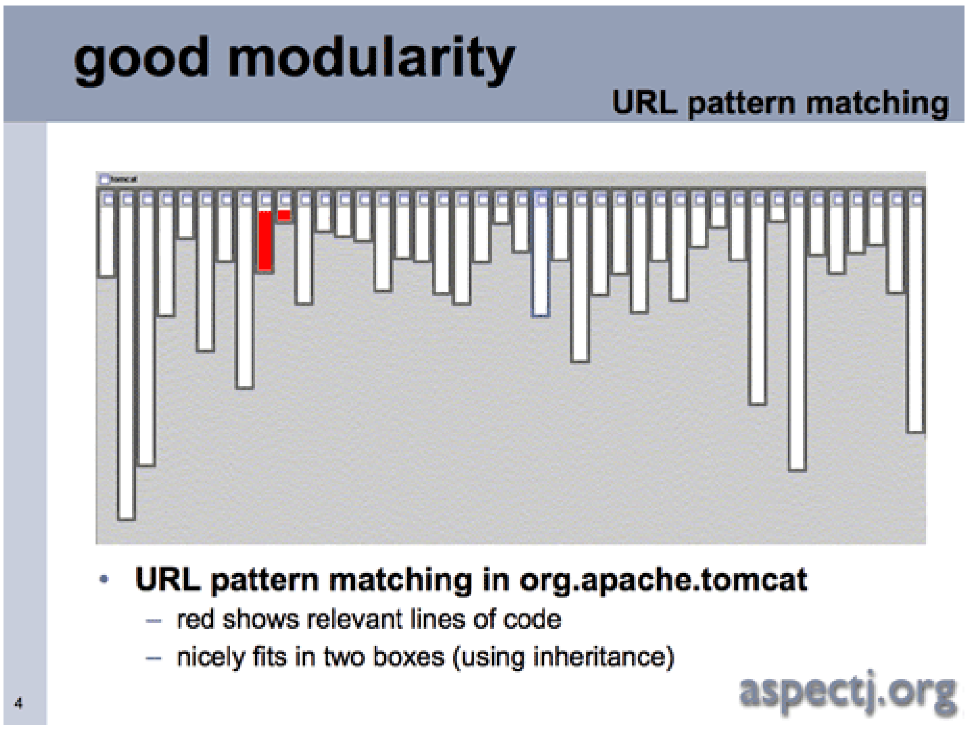
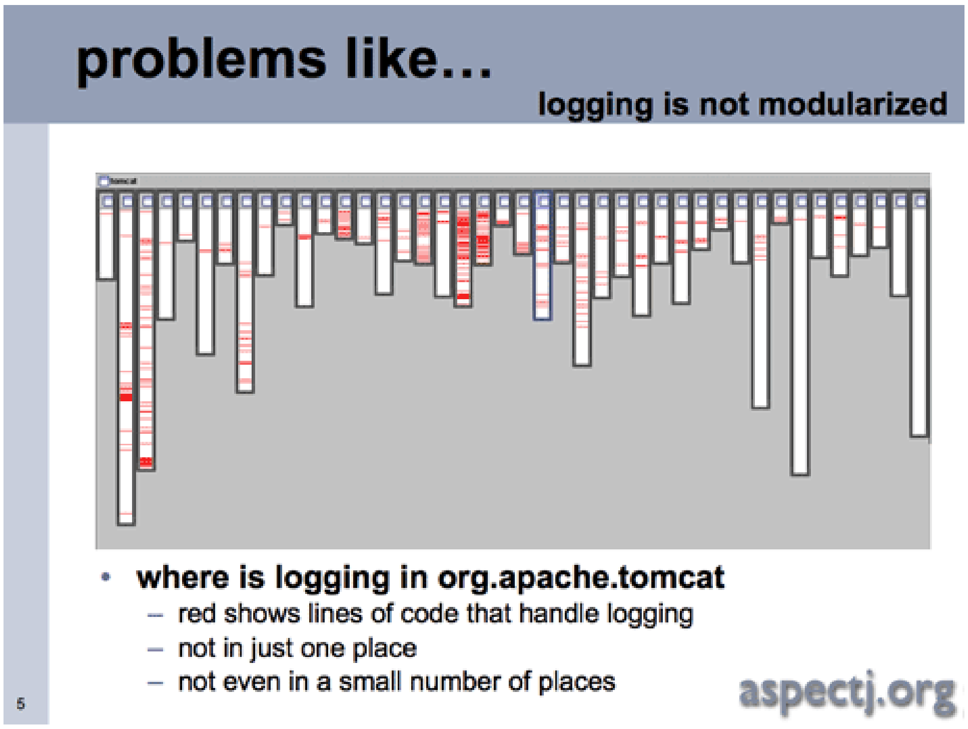


Figure 2.1: URL pattern matching in org.apache.tomcat (Kiczales & Hilsdale 2003, slide 4)

However, Kiczales (2004) points out that AOP is not a “Silver Bullet”:

[1] http://tomcat.apache.org/



problems like…

logging is not modularized

- where is logging in org.apache.tomcat
  - red shows lines of code that handle logging
  - not in just one place
  - not even in a small number of places

aspectj.org

5

Figure 2.2: Logging in org.apache.tomcat (Kiczales & Hilsdale 2003, slide 5)

Another way to understand AOP is in terms of how it works. A common misconception associated with this perspective is to equate all of AOP with just one part of the supporting mechanisms. This kind of error is analogous to saying that OOP is just abstract data types. Probably the most common mechanism error is to equate AOP with interceptors.(...) It's true that AOP does use functionality like interceptors and Lisp advise. It also incorporates techniques from reflection, multiple inheritance, multi-methods and others. However, AOP has an explicit focus on crosscutting structure and the modularization of crosscutting concerns. The rule of thumb to avoid mechanism errors? Remember that AOP is more than any one mechanism - it's an approach to modularizing crosscutting concerns that's supported by a variety of mechanisms, including pointcuts, advice and introduction.

2.2.2 AOP and UML

The integration of AOP concepts into the UML standard is an ongoing process that has not finished yet. Dean Wampler (2003, page 5) puts AOP in the context of the Model Driven Architecture (MDA) initiative of the Object Management Group (OMG), explaining that "Aspects appear in all levels of the development process. For example, security, persistence of data, transactions, and high availability concerns appear at the requirements level all the way down to the implementation." In this sense, AOP would require specific modeling approaches to get its benefits, which are not described in this paper.

To bridge the gap, Stein, Hanenberg & Unland (2002, page 5) propose a graphical representation of AOP constructs in Use-Case, Sequence and Interaction diagrams. Jacobson (2003, page 18) explains that Use-Case diagrams are those best suited to provide a graphical approach to the AOP paradigm, and the same thesis is at the heart of the book written by Ng & Jacobson (2005, page 33). They explain that concern tangling is easily visible right at the stage of use-case design, all the way to class design, as shown in figure 2.3.

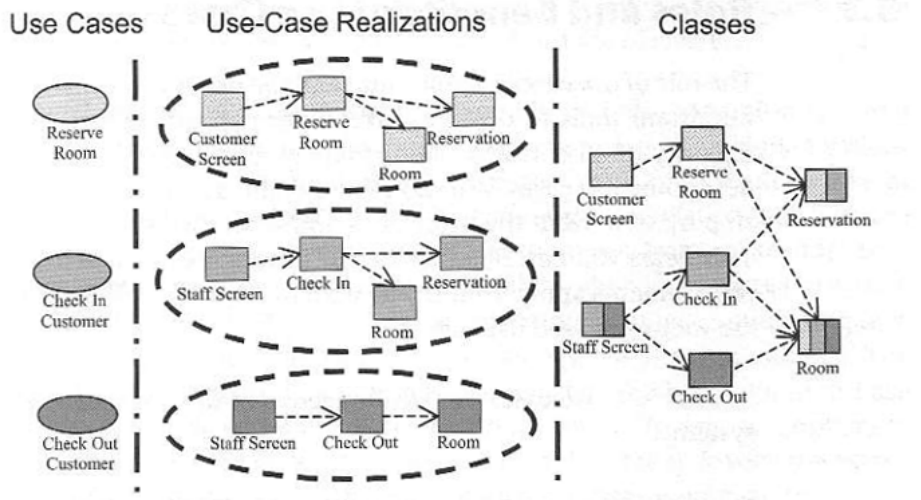


Figure 2.3: Use Case Realizations and AOP (Ng & Jacobson 2005, page 33)

2.3 C++ Programming

2.3.1 Introduction

C++ is a multiparadigm programming language, and according to Tiobe (2008), occupies the third place in the list of the most widely used programming languages at the time of this writing. Meyers (2005, page 11) describes C++ as a federation of languages, each with its own complexity and characteristics:

- C (The "Forgotten Trojan Horse", as defined by Johnson (2004))
- Object-Oriented C
- Template C++
- The Standard Template Library

As a particular experiment on C++ programming, **Rem** uses the four "federation members", adding to the mix the requirement for cross-platform conformity.

2.3.2 Libraries

Stroustrup (2002, page 1) has said that "without a good library, most interesting tasks are hard to do in C++; but given a good library, almost any task can be made easy". This basic fact has pushed the author to consider the use of a strong, cross-platform set of libraries, in order to achieve some basic goals for the project:

User Inteface: **Rem** must offer a reasonably similar interface in Windows, Linux and Mac OS X;

Foundation: **Rem** must compile and run seamlessly in each platform;

Data: **Rem** must use a portable file format, so that diagrams created in one platform can be used in the other without changes.

There are virtually thousands of different libraries that fulfill these goals, and the criteria to choose one for each of the above goals had to be taken with a strong rationale.

The candidates for each goal were:

- User Interface:

    1. Qt
    2. wxWidgets
    3. Juce
- Foundation:
    1. Adaptive Communication Environment (ACE) (Schmidt 2007)
    2. Portable Components (POCO) (Informatics 2008)
    3. Portable Types (PTypes) (Melikyan 2008)
    4. Boost (Dawes, Abrahams & Rivera 2008)
    5. Platinum (Duerner, Maekitalo & Indrayanto 2007)
    6. VR Juggler Portable Runtime (VPR) (Cruz-Neira 2008)
- Data:
    1. SQLite
    2. XML
    3. Binary files

Choice Rationale

Given the large choice of C++ libraries, it is very important to have strong reasons to choose one over the other, since the choice of a library has a definitive impact on all the source code of an application.

For the user interface layer, the Juce library was chosen because it is one of the only GUI C++ libraries available using multiple inheritance, and this single fact has allowed its author to create statically-bound binaries which are often smaller than those created with similar libraries; as Jewell (2006) explains,

> If you download the Win32 version of the demo, you might be surprised to discover that the EXE file tips the scales at a sylphlike 751 KBytes. That's quite impressive for a stand-alone program developed with a cross-platform library and no DLL dependencies. However, inquisitive bugger that I am, a quick sniff with HexEdit instantly revealed that Mr Storer had used the

UPX compressor to scrunch the demo executable. Even so, at around 1.7 Mbytes (the uncompressed size), I'm still impressed. Static builds with other cross-platform libraries are often very much larger.

The criteria to choose the library for the "Foundation" layer was based in the analysis made by Distler (2007), in which the "POCO" library is selected as the most portable, fast, better maintained, smaller and easier to use of all the libraries tested.

As for the data file format, SQLite has been chosen, since it is described as "cross-platform - you can freely copy a database between 32-bit and 64-bit systems or between big-endian and little-endian architectures." (Hipp, Kennedy & Harrelson 2008*a*). Moreover, many other software packages use SQLite for this purpose (Hipp, Kennedy & Harrelson 2008*b*).

2.3.3 Template Metaprogramming and the Standard Template Library

As explained above, **Rem** uses SQLite as its main storage format. However, to simplify the manipulation of SQLite databases, **Rem** features an implementation of the Active Record pattern, described by Fowler (2002*b*). **Rem**'s implementation of the pattern was heavily inspired by the homonym class found in the Ruby on Rails web development framework (Hansson 2008).

The storage::ActiveRecord class used by **Rem** is built using template metaprogramming techniques, defined by Alexandrescu, Meyers & Vlissides (2001, page 6) as " a good candidate for coping with combinatorial behaviors". **Rem** uses several patterns based on template metaprogramming:

"Curiously Recurring Template Pattern": first described by Coplien (1995) and later explained by Josuttis & Vandevoorde (2003) to define the utility::Singleton template class.

"Property Pattern": (Duffy 2004, pages 47-60) used to store the individual values of each field together with the column name, creating dynamic objects whose structure can be changed at runtime, and which generate their own SQL code whenever needed.

"Template Methods in Template Class": given that the structure of ActiveRecord instances is dynamic, the types of their properties are also dynamic; to retrieve the values stored in the fields, the template class exposes template methods, as explained by Angelidis (2005).

The Standard Template Library is defined by Deitel & Deitel (2005, page 1112) as defining "powerful, template-based, reusable components that implement many common data structures, and algorithms used to process those structures". **Rem** uses template metaprogramming techniques, together with the STL, across all the supported platforms, providing an extensive level of code reuse and modularity with a small source code base.

2.3.4 Multiple Inheritance

C++ is one of the few object-oriented programming languages supporting the paradigm of multiple inheritance. Stroustrup (1987, page 10) describes the implementation rationale, explaining the need for virtual inheritance, both public and private.

Schaerli, Ducasse, Nierstrasz & Black (2003, page 4) explains the tradeoffs and possible problems created by this approach, suggesting new programming constructs called "Traits" to solve the problem:

> One of the problems with multiple inheritance is the ambiguity that arises when conicting features are inherited along different paths. A particularly problematic situation is the "diamond problem" (also known as "fork-join inheritance") that occurs when a class inherits from the same base class via multiple paths.

Cline (2006) explains that the use of "public virtual" inheritance solves most of the compilation and linkage problems created by the "diamond problem".

2.3.5 Cross-Platform Issues

Rem has a mandatory core requirement to generate executables from the same source code base, running seamlessly in Windows, Linux and Mac OS X. As explained by Stuart, Dascalu & Jr. (2005, page 2), this creates a number of problems, usually solved in two common ways:

> The first approach involves the use of separate segments (branches) of code, each written for a specific target. (...) The second approach is charac-

terized by extensive use of preprocessor commands, which leads to several shortcomings, (...)

In the case of **Rem**, the major issues identified with the support of multiple platforms were:

Data Types: C++ compilers are available in virtually all microprocessor architectures and operating systems. However, the internal storage used for data depends is not specified by the standard, and as such, depends on the underlying architecture. As Obiltschnig (2006, page 1) states, “As with integer types, the C++ standard does not specify a storage size and binary representation for floating-point types.”. Moreover, Kalev (2007) shows that the length of “long” and “long double” integer data types is different in 32 and 64 bits platforms.

Library Availability: Given the complex task of creating cross-platform software, not all commercial or open source C++ libraries support portability to the platforms required by **Rem**. This includes not only the libraries used for the GUI, but also those used for complex data structures, data storage and unit testing.

Data Portability: Given the difference in endianness between x86 and POWER architectures, the file format to be used in data exchange must take into account the alignment of data in each. This rules out the use of binary files, as explained by Lewis (2008). The use of SQLite databases solves this problem, since the file format is handled completely by the library, abstracting this issue for **Rem**.

Build Toolkits: Each platform has its own preferred set of tools; for example, Linux software is most commonly built using Makefiles, while both the creators of Windows and Mac OS X have their own Integrated Development Environments (IDEs), Visual Studio and Xcode, respectively. However, while developing cross-platform software, it is important to minimize the job for producing an executable, and as such, tools like GNU Autoconf Automake and Libtool (FSF 2008), premake (Perkins 2008), Rake (Weirich 2006) and CMake (Kitware 2008) offer cross-platform solutions to centralize and automatize the creation of binaries.

In the case of **Rem**, table 2.3.5 shows the characteristics of the underlying architectures of the target platforms:

A more comprehensive description of the various solutions found to these problems can be read in section 5.1.2 of this document.

Operating System	CPU Architecture	Registers	Endianness	Instruction Set
Windows XP SP 2	x86, P6, NetBurst	32 bits	Little-endian	CISC
Mac OS X 10.4	PowerPC G4	32 bits	Big-endian	RISC
Mac OS X 10.5	PowerPC G5	64 bits	Big-endian	RISC
Mac OS X 10.5	Core Duo 2	64 bits	Little-endian	RISC
Ubuntu Linux 7.10	x86	32 bits	Little-endian	CISC

Table 2.1: Comparison of the Target Platforms; source: Stokes (1999)

Finally, to centralize and automatize the creation of **Rem** binaries, the CMake tool was used. CMake allows not only to generate "Makefiles" and project files for most IDEs, but also provides an integrated packaging and testing solution ready to use.

2.4 Software Quality and Testing

The quality management approach used for **Rem** is inspired by some of its characteristics:

One-person effort: In spite of its complexity, **Rem** has been designed and developed by only one person. This forced the author to automatize most tedious tasks, making him concentrate in the most important sections of the project: the end-user functionality and the quality of the source code.

Multi-platform effort: The same source code must run seamlessly in three rather different software platforms. The quality efforts must be themselves portable.

These characteristics have been translated in the following quality management practices:

- In his classic "The Mythical Man-Month", Brooks (1995, page 20) wrote about "a simple rule of thumb for scheduling a software task:"

  1/3 planning
  1/6 coding
  1/4 component test and early system test
  1/4 system test, all components in hand.

- As Glass (2002, page 101-103) explains in its "Fact 35: Test automation rarely is. That is, certain testing processes can and should be automated. But there is a lot of the testing activity that cannot be automated". This is certainly true for **Rem**, given the short schedule, the small size of the "team" in charge and also because of the inherent complexity of testing GUIs. **Rem** has a strong test suite, targeted to the middle and lower layers of the software.

- It is well-known that the use of a statically-typed, compiled programming language as C++ does not make software less prone to quality problems, as explained by Eckel (2003):

  > If a program compiles in a statically typed language, it just means that it has passed some tests (...) But there's no guarantee of correctness just because the compiler passes your code.(...) The only guarantee of correctness, regardless of whether your language is statically or dynamically typed, is whether it passes all the tests that *define the correctness of your program*.

- Spolsky (2001) explains that daily builds and smoke tests are a fundamental part of software craftmanship. **Rem** can be built from source code with individual "build" script files for each supported platform, generating not only the executables but also the documentation and the installer packages, all in a single operation.

- Richardson & Gwaltney (2005, pages 88-97) shows how code reviews, coupled to refactoring techniques and automated unit testing suites (Fowler et al. 1999, page 89) can enhance the quality of the code. **Rem** has undergone several code reviews by the author, the most important of which has been during the revision 104 (Kosmaczewski 2008*b*) in which the whole inheritance structure of the "storage" namespace was reviewed, as stated on the change log: "Major refactoring of the ActiveRecord family of classes; removed the use of the 'curiously recurring pattern'; now an 'Abstract base class' helps implementing 'sister-class method calls' between the HasMany, the BelongsTo and the ActiveRecord classes. All 32 tests running properly after the change!".

- The use of source code control software is a key quality management practice, as proved by Gunderloy (2004, pages 34-50).

2.5 Free and Open Source Software

Arguably, the Free and Open Source Software (FOSS) movement is one of the most powerful forces of the software market since the beginning of the 21st century. FOSS has been defined by Raymond (2001, pages 71-72) as follows:

> Under the guidelines defined by the Open Source Definition, an open-source license must protect an unconditional right of any party to modify (and re-distribute modified versions of) open-source software.

Weber (2004, page 227) provides a complete overview of the political and economical impact of FOSS in markets, proposing it as an “experiment in social organization around a distinctive notion of property rights”. The capacity to freely download, use, modify and redistribute FOSS has fostered a huge community of projects, many of which were used during the development of **Rem**. This has had the benefit of reducing the number of defects in it, since, as Graham (2004, page 149) explains,

> But the advantage of open source isn’t just that you can fix it when you need to. It’s that everyone can. Open source software is like a paper that has been subjected to peer review.

However, not all authors agree on the apparent benefits of FOSS; Glass (2002, pages 51-55) mentions his concerns, as part of its “Fact 19: Modification of reused code is particularly error-prone”, arguing that “it is easy to access open-source code to modify it, but the wisdom of doing so is clearly questionable, unless the once-modified version of the open-source code is to become a new fork in the system’s development, never to merge with the standard version again”.

Rice (2007, page 268) has an even stronger view about the issue of quality management in the FOSS world, stating that “From a legal perspective, the open source movement puts the users of software in an even more precarious position than does proprietary software manufacturers. (...) This means if a standard of care for software developers come to fruition, the contributors to open source software applications would in many respects remain unaccountable for any breach of this standard.”

2.6 User Interface Design

One of the core requirements for **Rem** is to adhere to common GUI paradigms, thus reducing the steepness of the learning path required to mastering it. In this sense it was important, during the development of **Rem**, to treat GUI design as a first-class priority, avoiding the "blooper 77" of treating user interface as low priority (Johnson & Nielsen 2000, page 424):

> The user interface is not a feature that can be dropped to meet a schedule or budget constraint. It pervades and affects the entire product. If a product's user interface is shoddy, the product is shoddy, because the user interface is the part of the project that customers experience.

Finally, it is important to identify the application category which **Rem** belongs to. The main task of **Rem** is the creation of diagrams; in this sense, it fits in a broad category of "Builders and Editors" as defined by Tidwell (2005, page 243):

> They are, first and foremost, canvases - they offer the user an empty shell she can creatively fill, plus the tools to fill it with.

Following Tidwell, editors and builders, have four essential elements, all of which are present in **Rem**:

WYSIWYG Editing: **Rem** allows users to create diagrams that look exactly the way they build them, and which can be printed and otherwise transferred without losing the initial layout.

Direct Manipulation: **Rem** allows users to add, modify and delete elements from the canvas without any other limits than those created by the requirements of a UML tool.

Modes: **Rem** can change its behavior depending on the context, and it provides clues about these changes as feedback to the user.

Selection: **Rem** users can select one or many items, deselect them, and perform operations on the selected items all at once.

Chapter 3

Theory

This chapter provides a description of the theories and knowledge elements used to create **Rem**.

3.1 Design Patterns

The term “Design Pattern” is a term borrowed to the architect and urbanist Christopher Alexander, and adapted to the software engineering field by Beck & Cunningham (1987) in 1987:

> We propose a radical shift in the burden of design and implementation, using concepts adapted from the work of Christopher Alexander, an architect and founder of the Center for Environmental Structures. Alexander proposes homes and offices be designed and built by their eventual occupants. These people, he reasons, know best their requirements for a particular structure. We agree, and make the same argument for computer programs.

The concept of “Design Patterns” was finally adopted by the software engineering community in 1995 with the issue of the book “Design patterns : elements of reusable object-oriented software” by Gamma, Helm, Johnson & Vissides (1995), also known as the “Gang of Four” or “GoF” book. The idea of design patterns became a “buzzword” lately, and several important books now hold the word “pattern” in the title[1]:

[1]A search on Google about books featuring the words “design pattern” holds, at the time of this writing, more than 24’000 results: http://books.google.com/books?q=design%20pattern

- “Modern C++ Design: Applied Generic and Design Patterns” (Alexandrescu et al. 2001)
- “Patterns of Enterprise Application Architecture” (Fowler 2002*a*)
- “Head First Design Patterns” (Freeman, Freeman & Bates 2004)
- “Designing Interfaces : Patterns for Effective Interaction Design” (Tidwell 2005)

However trendy, patterns serve their purpose to convey meaning, allowing engineers to describe complex constructions with few words. Several different design patterns have been used throughout the **Rem** project:

Model-View-Controller: The Model-View-Controller design pattern was described for the first time in 1979 by Trygve Reenskaug as an architectural pattern found in the Smalltalk programming language, created by Alan Kay in Xerox PARC in 1972. It promotes the separation of software code in three distinct layers, families or entity types:

Models: Models represent the entities (and collections thereof) that the software ultimately is about; in the case of an accounting software package, typical models could be Customer, Invoice, Payment or Account.

Views: Classes belonging to this family provide a visual representation of the model entities, dealing with the direct interaction with the end user. They could or not consist of GUI code, like for example in the case of command-line utilities, where the views provide an interface reading from and writing to the console. In the case of web-based systems, views are often in charge of producing the HTML code required for users to interact with the system.

Controllers: The controllers are in charge of interfacing between models and views; they route events created in the views by the user, triggering changes in the model, and also notify the views about changes in the underlying model.

Active Record: (Fowler 2002*b*) defines the Active Record design pattern as a medium to serialize and deserialize simple object graphs into a relational database; many frameworks, particularly in those targeting the rapid development of web appli-

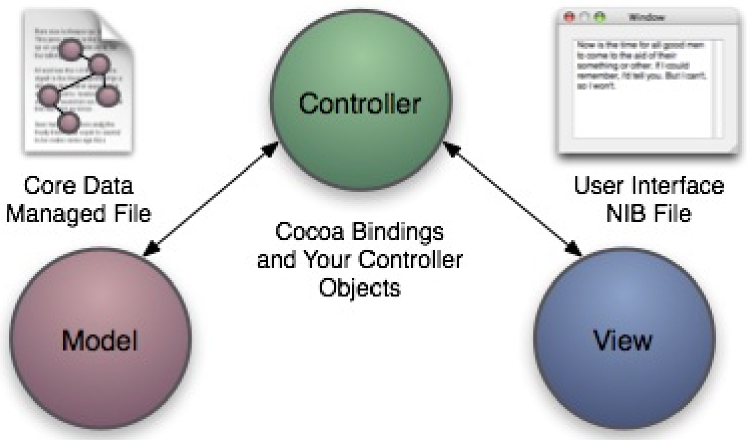


Figure 3.1: MVC pattern in the Apple Cocoa framework (Apple 2006)

cations feature an ORM[2] based in this pattern, including Hibernate[3], Rails[4] and Django[5].

Observer: The Observer design pattern is described by Gamma et al. (1995, page 293) as

> Define a one-to-many dependency between objects so that when one object changes state, all its dependents are notified and updated automatically.

In the case of **Rem** the pattern is used extensively, thanks to the decoupling of producers and receivers of events via the Poco::NotificationCenter class, and the Poco::Notification-based classes.

Singleton: The Singleton design pattern is described by Gamma et al. (1995, page 127) as

> Ensure a class only has one instance, and provide a global point of access to it.

Rem features a template-based, thread-unsafe utility::Singleton class, which is used by others to become singleton instances thanks to the static get() method, which returns a reference to the single instance of the class:

[2] "Object-Relational Mapper"
[3] http://hibernate.org
[4] http://rubyonrails.org
[5] http://djangoproject.org

```
template <class T>
T& Singleton<T>::get()
{
    static T _instance;
    return _instance;
}
```

Listing 3.1: utility::Singleton::get() static method

Subclasses inheriting from this class must set their constructor private, and offer a “friend” level access to the get() static method:

```
class CommandDelegate : public ApplicationCommandTarget,
                        public Singleton<CommandDelegate>
{
public:
        // (...)

private:
    CommandDelegate();
    friend CommandDelegate& Singleton<CommandDelegate>::get();
```

Listing 3.2: Code using the utility::Singleton template class

Property: The Property pattern is described by Duffy (2004, page 47) as follows:

> A property is a template class that has a name and a value. Furthermore, it has a number of member functions, two of which are functions to `set` and `get` the encapsulated value. The underlying types of the name and the value are generic, which means that programmers can instantiate these types with their own specific types. In short, the Property pattern leads to high levels of reusability.

Rem combines the power of this pattern with the Poco::Any type, allowing the storage namespace to become a simple yet powerful ORM to the project:

```
class AnyProperty : public Property<string, Any>
```

```
{
public:
    AnyProperty();
```

Listing 3.3: Definition of the storage::AnyProperty class

3.2 Advanced C++ Software Development

3.2.1 Cross-platform Software Development

Rem is built in C++, one of the most widely used programming languages in the industry. This use is somehow limited by the support of the standard in common compilers; however, much effort has been put in ensuring that every line of code compiles and executes similarly across platform boundaries, and the result has been a success.

Among the different problems faced in cross-platform source bases, **Rem** has faced the following:

- Data types
- Deployment
- Compiler and linker options
- Compiler support of the C++ standard
- Library availability

A description of the solutions found for these problems can be found in section 5.1.2 of this document.

3.2.2 Template Metaprogramming

One of the most complex yet powerful features of C++ is **Template Metaprogramming**. The use of template metaprogramming in **Rem** was motivated primarily by the goal of achieving a real separation between the SQLite database backend, using an implementation of the Active Record design pattern. The implementation used by **Rem** uses template metaprogramming to achieve the generation of static methods, used to perform the following tasks:

- Retrieval of an existing instance in file:

```
template <class T>
T* ActiveRecord<T>::findById(const ID id)
{
    stringstream query;
    query << "SELECT * FROM ";
    query << T::getTableName();
    query << " WHERE id = ";
    query << id;
    query << ";";

    T* item = NULL;
    SQLiteWrapper& wrapper = SQLiteWrapper::get();
    bool ok = wrapper.open();
    if (ok)
    {
        string table = T::getTableName();
        map<string, string> schema = wrapper.getTableSchema(table);
        ok = wrapper.executeQuery(query.str());
        wrapper.close();
        if (ok)
        {
            vector<AnyPropertyMap>* maps = getPropertyMaps(schema);
            vector<AnyPropertyMap>::iterator iter;
            for (iter = maps->begin(); iter != maps->end(); ++iter)
            {
                item = new T(*iter);
            }
            delete maps;
        }
    }
    return item;
}
```

Listing 3.4: storage::ActiveRecord::findById() static method

- Retrieval of all existing instances in file:

```
template <class T>
vector<T>* ActiveRecord<T>::findAll()
{
    stringstream query;
    query << "SELECT * FROM ";
    query << T::getTableName();
    query << ";";

    string q = query.str();
    return getVectorByQuery(q);
}
```

Listing 3.5: storage::ActiveRecord::findAll() static method

- Deletion of an existing instance:

```
template <class T>
void ActiveRecord<T>::remove(const ID id)
{
    stringstream query;
    query << "DELETE FROM ";
    query << T::getTableName();
    query << " WHERE id = ";
    query << id;
    query << ";";

    SQLiteWrapper& wrapper = SQLiteWrapper::get();
    bool ok = wrapper.open();
    if (ok)
    {
        ok = wrapper.executeQuery(query.str());
        wrapper.close();
    }
}
```

Listing 3.6: storage::ActiveRecord::remove() static method

Another use of this technique allowed the creation of parent-child relationships, where a single parent owns a set of children instances, and each child has a pointer to its parent. This is done through the “HasMany” and “BelongsTo” template classes, whose names are inspired from the same classes in the Ruby on Rails web development framework:

```
class Diagram : public ActiveRecord<Diagram>
              , public BelongsTo<Project>
              , public HasMany<Element>
{
public:
```

Listing 3.7: Definition of the metamodel::Diagram class

The advantage of template metaprogramming resides in the fact that the same infrastructure is available for all classes, can be reused with very little effort, and offers effectively isolates models, controllers and views, following the MVC design pattern described previously.

3.2.3 Multiple Inheritance

C++ is one of the few programming languages available that supports the paradigm of multiple inheritance. While often criticized, **Rem** has benefited from it in several ways:

Higher code modularity: There is a greater number of classes, each with well-defined responsibilities, and they can be combined in clever ways to achieve the required behavior.

Ease of maintenance: Each class has a distinct, limited set of responsibilities, and this helps maintenance, reducing the possible number of places where a correction should be made.

Code readability: The definition of each class provides a clear meaning of its responsibilities, thanks to the enumeration of parent classes.

However, multiple inheritance does not come without trouble, particularly in the case of virtual methods. To solve some problems encountered during the development of **Rem**, the author had to resort to virtual base classes, as well as the use of the “dynamic_cast<>” operator, to solve the “dreaded diamond” problem (Cline 2006):

```
template <class T>
class ActiveRecord : public virtual Persistable
{
public:
```

Listing 3.8: Definition of the storage::ActiveRecord class

```
template <class P>
void BelongsTo<P>::setParent(Persistable* parent)
{
    // Here we enforce the interface that, for polymorphic
    // reasons, cannot be enforced in the method signature!
    _parent = dynamic_cast<P*>(parent);
}
```

Listing 3.9: storage::BelongsTo::setParent() method

Chapter 4

Analysis and Design

This chapter contains the results of the analysis and design process which led to the creation of **Rem**.

4.1 Requirements Model

This section provides a requirements model of the **Rem** system, including use case diagrams, and detailed use case descriptions for some key use cases.

4.1.1 Description

Rem is a desktop-based application, running natively in Windows, Mac OS X and Linux, providing users the capability to:

- Create new UML projects, consisting of several related diagrams. These projects group together different diagrams so that they can be exported or manipulated at the same time, for example to provide code skeletons, image files or other to perform operations.
- Create new UML diagrams, either “isolated” or as part of a project, as defined by the UML standard, of the following types:

  Use-case diagrams: these diagrams will allow users to model system features and requirements from an end-user perspective. These diagrams will support inclusion, extension, actor inheritance, subsystem grouping and stereotypes.

Class diagrams: these diagrams show the internal structure and relationship of classes used to design a software system. They will feature public and private members (fields or methods), inheritance relationships, containment, aggregation and stereotypes.

Sequence diagrams: these diagrams highlight the key interactions between different entities at runtime. They will feature synchronous and asynchronous method calls, invocations to "self", sequencing, imbrication and stereotypes.

- Open existing diagram files, even if created in different operating systems;
- Export diagram files to other formats:
  - XMI (XML Metadata Interchange)
  - PNG (Portable Network Graphics)
  - Other export formats as required.
- Export projects to code, in the following programming languages:
  - Ruby;
  - Python;
  - C++;
  - Java;
  - Other export languages as required.
- Use the application through the command-line, in batch mode, allowing for quick scripting of common tasks;

Besides these core features, **Rem** will also support the following operations, considered standards in graphical operating system environments nowadays:

- Maximize the application window;
- Minimize the application window;
- Close the current diagram, without being forced to save the diagram to disk;
- Move diagram elements on the screen freely;

- Copy and paste visual elements, in the same diagram, from and to other diagrams.
- Change the properties of the different diagram elements, using a visual interface;
- Quit the application;

4.1.2 Subsystems

Rem will consist of different subsystems:

GUI subsystem interfaces with the JUCE GUI toolkit, in charge of on-screen drawing of visual elements, as well as interaction with the user.

Command-line subsystem allowing users to interact with the application using a command-line interface, suitable for scripting common operations.

UML Metamodel subsystem containing the meta-entities representing diagrams, actors, relationships and other objects used to build diagrams.

Storage and export subsystem based on a cross-platform file format, based on a serialization of in-memory entities for later retrieval, and providing several other .

Figure 4.1 shows the organization of code in namespaces, packages or executables for the **Rem** system.

4.1.3 Non-functional Requirements

Rem will have the following non-functional requirements:

Easy installation: users will be able to install the application using the platform's standard way of doing installations:

- DMG (Disk Image) files in Mac OS X;
- Installation Wizard for Windows;
- Binaries + source code for Linux.

Non-administrative requirements: users will not need administrative privileges to use this application.

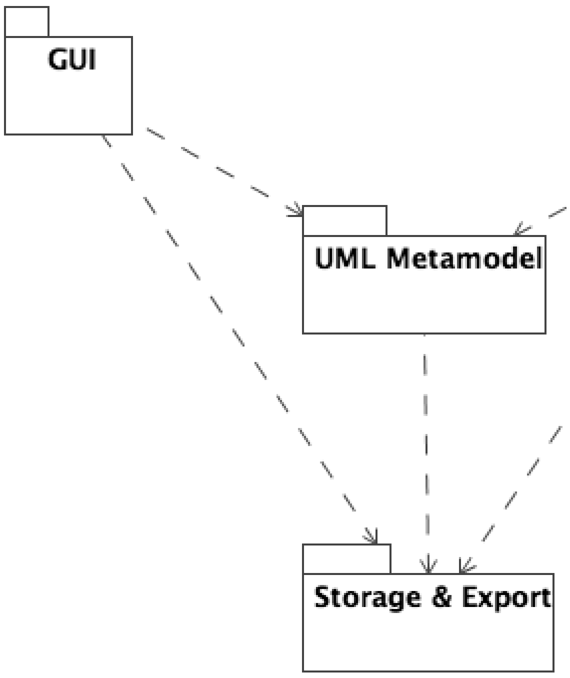


Figure 4.1: Package Diagram

Licensing: The system will be distributed using the MIT license[1], allowing users to modify, extend and redistribute **Rem** to suit their needs and expectations.

Startup: **Rem** should have a smooth start-up procedure, allowing users to be immediately productive when the application starts up.

Language: The application must be available in English as a first version, and the system must be extensible so that other languages can be added easily later, without modifying the application behavior.

Tracing: The system must be traceable using an internal logging system; the logging system must be able to be changed dynamically during runtime (flat file, database, system event log, etc).

[1] http://www.opensource.org/licenses/mit-license.php

4.1.4 Physical Architecture

Rem will feature a layered architecture, where components in one layer only have direct dependencies on those represented in the layer immediately below. From “top to bottom” (where “top” represents the visual elements of the application) these are the main layers of the system:

1. UI layer + command line application
2. UI Controllers
3. Business model logic + UML models
4. Storage subsystem
5. Export subsystem (static formats + MDA stuff)

4.1.5 Actors

As this is a single-user desktop application, there will be only one possible actor for all use cases: the end user. This end user will not require administrative privileges to use the system. Figure 4.2 shows the use cases exposed by **Rem** for this single actor.

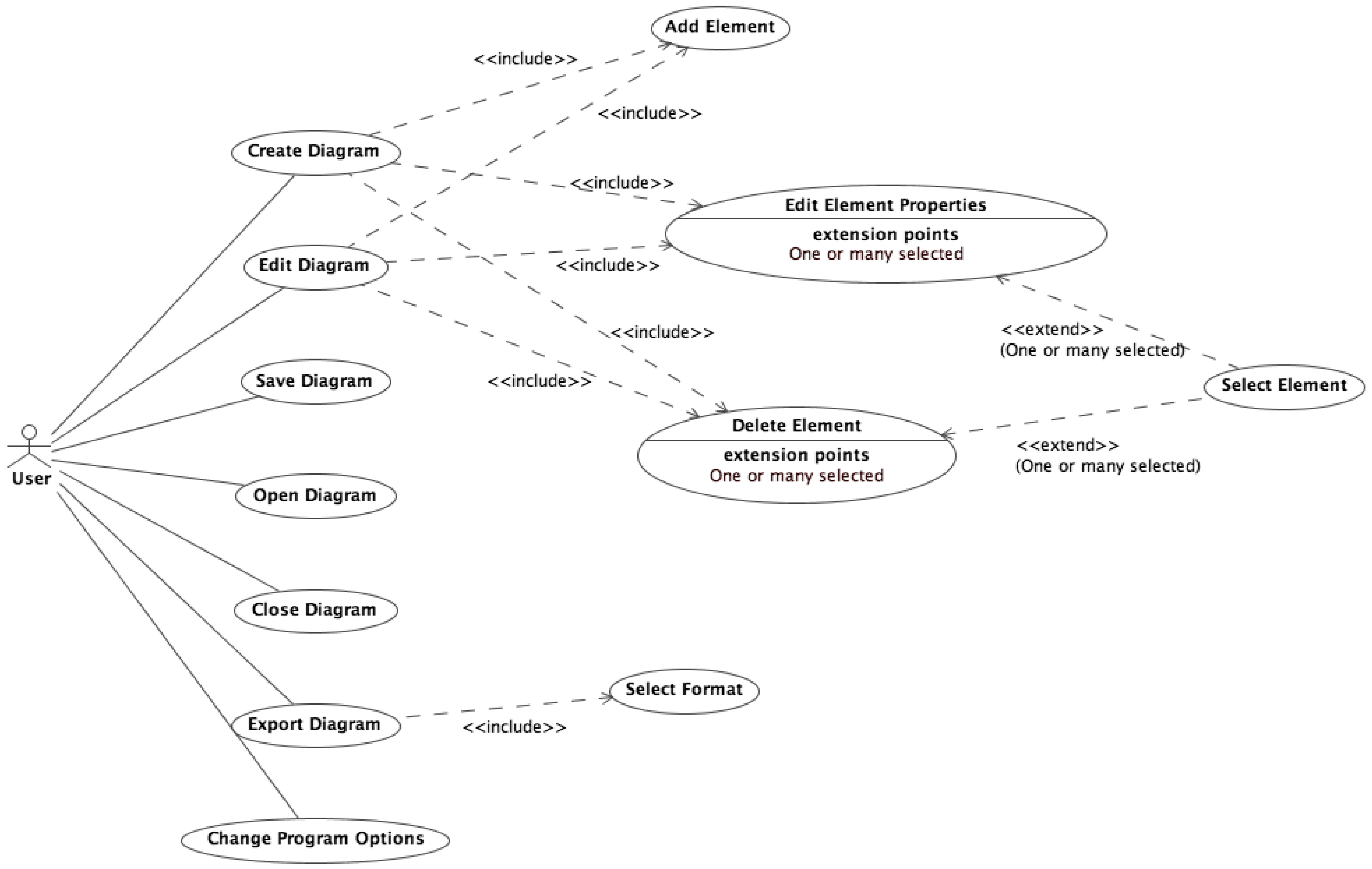


Figure 4.2: Use Case Diagram

4.1.6 GUI Subsystem

The **Rem** GUI Subsystem is the most visible part of the system. It provides a designer interface, where users can select items from a palette, put them on a canvas, move them, change their properties and create and delete diagrams.

The user interface will support most visual standards, making it easy to learn and use. Figure 4.3 shows a mockup of the user interface.

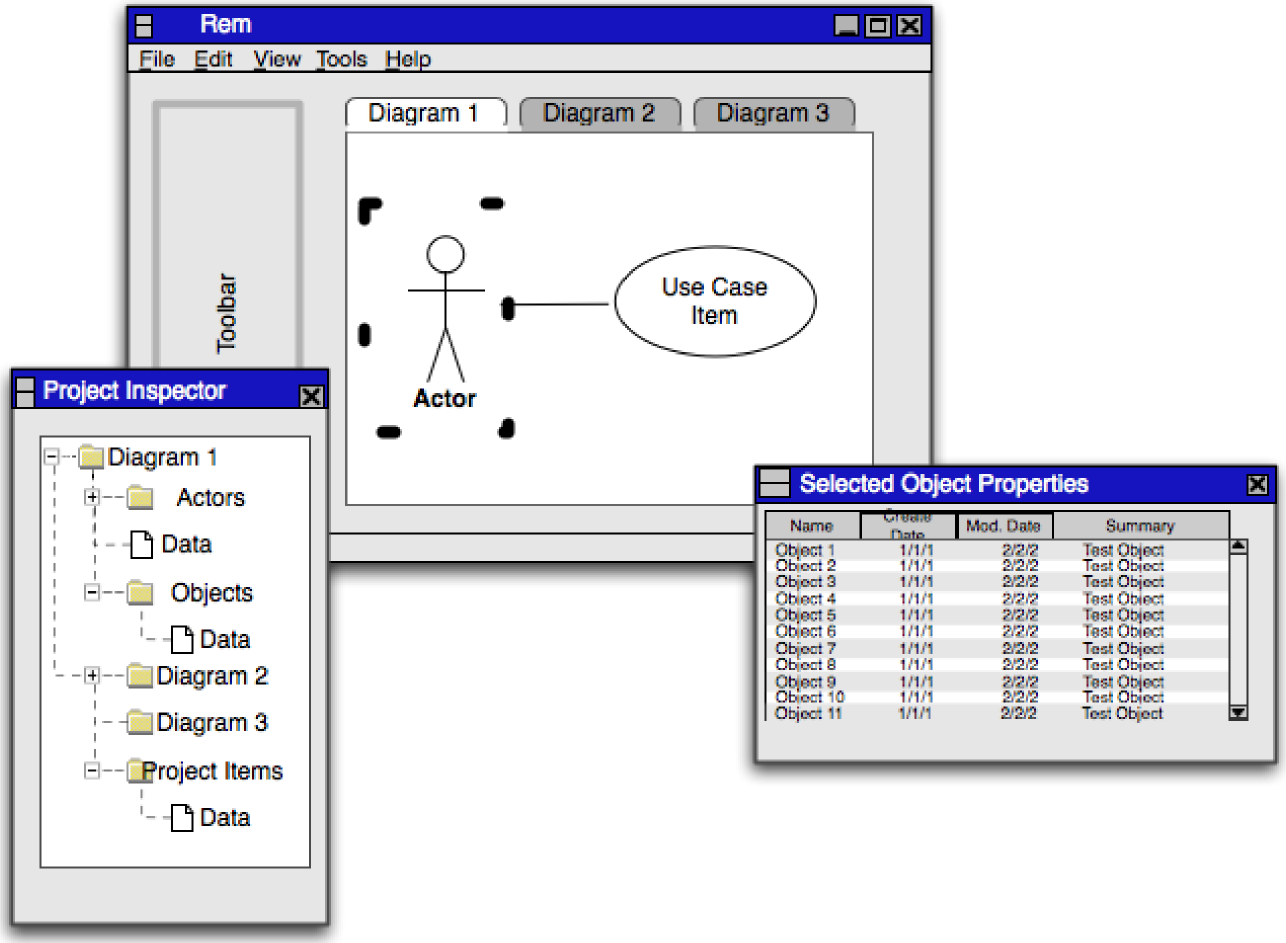
Following the nomenclature of graphical user interfaces, **Rem** fits in the "Builders and Editors" category; as such, it will follow the patterns described by Tidwell (2005):

1. Edit-in-Place;
2. Smart Selection;
3. Composite Selection;
4. One-Off Mode;
5. Spring-Loaded Mode;
6. Constrained Resize;
7. Magnetism;
8. Guides;
9. Paste Variations.

The GUI subsystem will be based in the Juce[2] library, a portable, lightweight yet extremely powerful C++ library, available as an open source or as a commercial product.

Juce uses its own set of "widgets", instead of providing a wrapper around the native platform's ones (like wxWidgets[3] does, nor it tries to imitate them, like Qt[4] does. Juce makes heavy use of the multiple inheritance capability of C++ to provide a small, fast, lightweight library, offering other capabilities at the same time (such as networking).

[2]http://www.rawmaterialsoftware.com/juce/
[3]http://wxwidgets.org/
[4]http://trolltech.com/products/qt



Name	Create Date	Mod. Date	Summary
Object 1	1/1/1	2/2/2	Test Object
Object 2	1/1/1	2/2/2	Test Object
Object 3	1/1/1	2/2/2	Test Object
Object 4	1/1/1	2/2/2	Test Object
Object 5	1/1/1	2/2/2	Test Object
Object 6	1/1/1	2/2/2	Test Object
Object 7	1/1/1	2/2/2	Test Object
Object 8	1/1/1	2/2/2	Test Object
Object 9	1/1/1	2/2/2	Test Object
Object 10	1/1/1	2/2/2	Test Object
Object 11	1/1/1	2/2/2	Test Object

Figure 4.3: User Interface

4.1.7 Command Line Subsystem

This subsystem offers a command-line interface, that offers a set of commands and parameters allowing power users to script common operations. This tool is targeted towards savvy users, able to use a terminal window without problems. It is faster than the GUI and offers a complete on-line help (a manual page in Unix, and help screens in Windows).

It will also provide the capability to "pipe" command results from one program to the other, following a common Unix idiom.

4.1.8 UML Metamodel Subsystem

This subsystem contains all the logic needed to create and manipulate UML diagrams and projects, its components (actors, classes, packages, relationships, etc) and is used by the command line and the GUI

4.1.9 Storage and Export Subsystem

This subsystem will be in charge of storing the representation of the UML Metamodel subsystem, used in a project or in a single diagram, as a platform-independent representation. To do this, the SQLite[5] library will be used. SQLite is a C library providing an embedded, lightweight, transactional and fast relational database engine, available in the three operating systems targeted by **Rem**, and offering advanced performance and security features.

4.2 Analysis Model

Given the importance of the UML Metamodel subsystem (it is used by both the GUI and the command line subsystems), this chapter provides its analysis model.

[5] http://sqlite.org/

4.2.1 UML Metamodel

Figure 4.4 shows the inheritance, containment and association relationships for the main classes of the **Rem** system.

To reduce coupling of the systems using the classes of this subsystem, factory methods will return instances (or collections thereof) of these to those requesting it.

This subsystem will also feature the highest number of unit tests, ensuring the highest possible quality.

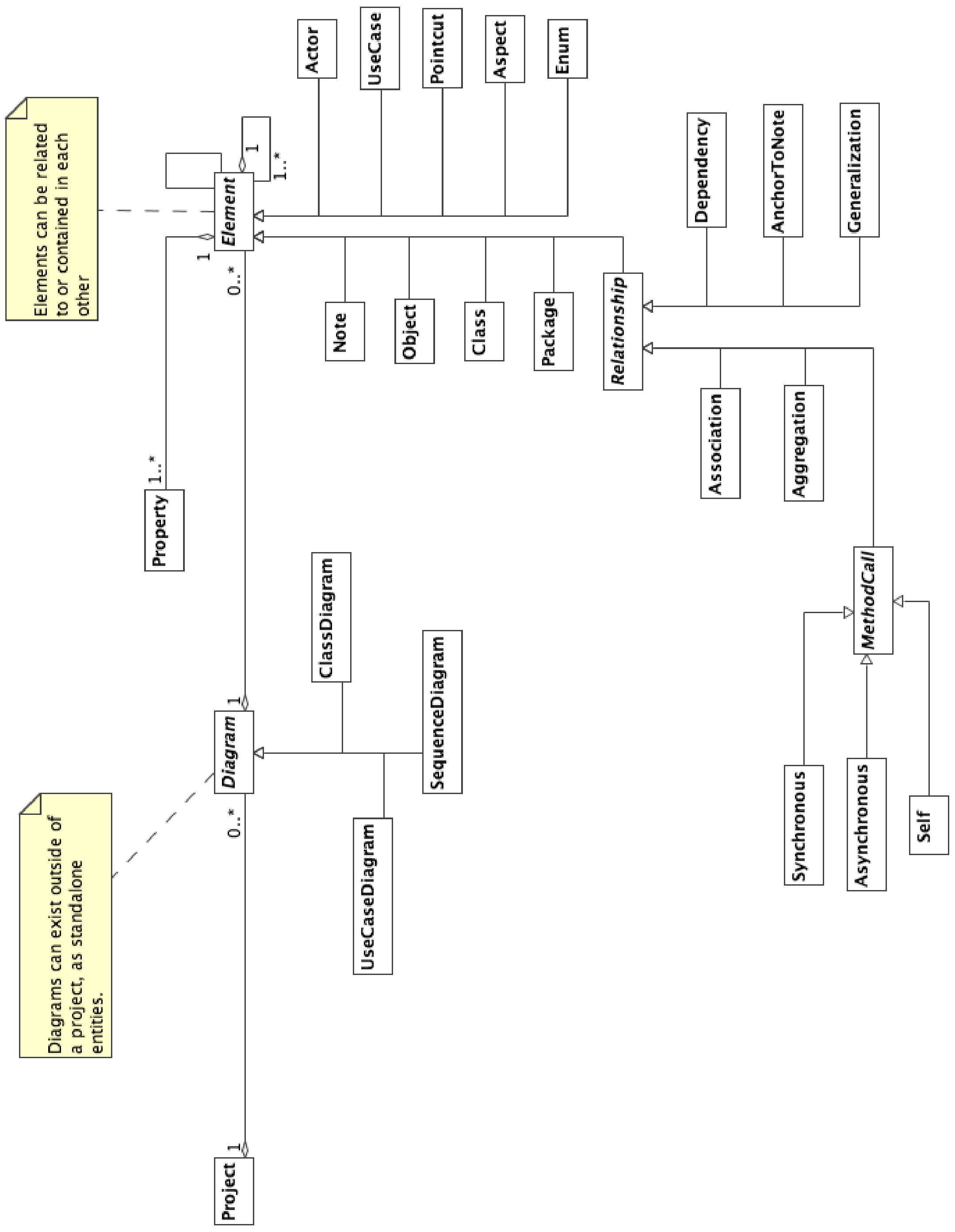


Figure 4.4: Class Diagram

Chapter 5

Methods and Realization

This chapter provides insight into the process of taking **Rem** from concept to reality, describing the pitfalls and design decisions taken during the course of its development.

5.1 Implementation

5.1.1 Development

Rem source code was mostly typed using Mac OS X 10.5 “Leopard” and its development tool, Xcode. The choice of the development platform was guided by several requirements:

Familiarity: the author has used this platform in his day-to-day tasks for the past 5 years, including the Xcode integrated development environment;

Stability and Reliability: Mac OS X is an exceptionally stable platform for most uses, and given the capability of C++ to manage hardware resources at a very low level (particularly memory and I/O) it is fundamental to use a reliable system, providing the process isolation required for a stable workflow.

As part of his development strategy, the author would fix incompatibilities and compilation issues in both Linux and Windows every week. The major part of this work, however, was due to the big differences between Windows and Unix-based systems such as Mac OS X and Linux; Windows required a higher degree of attention and adaptation of the source code in order to make it compile, link and work.

5.1.2 Cross-Platform Issues

The project suffered from several unexpected, rather complex cross-platform issues during the development of **Rem**; the following list provides details about these, and the ways found to address and solve them.

Library Availability: Most C++ libraries target a subset of the whole range of operating systems available, depending on the requirements of their respective teams and their target use cases. To find a stable, coherent and thorough set of libraries for **Rem**, then, was a major design decision.

However, even after having settled on using JUCE, POCO and SQLite, there were minor problems such as the availability of Poco as a "Universal Binary" for the Mac OS X platform. "Universal Binaries" are "fat" executable files containing native code running on both Intel and PowerPC architectures, both supported by Mac OS X, and which greatly simplifies application delivery; no matter which platform end-users have, the same binary distribution will run native code in it. The release of Poco available at the time of this writing only builds separate binaries for each platform; to achieve the simplified delivery of a Universal Binary version of **Rem**, the author found a pre-compiled version of Poco in a user forum on the web. This library helped achieve the goal of a Universal Binary Mac OS X distribution of **Rem**.

More details about this, including the address where the Universal Binary version of POCO was found, can be found in the Appendices at the end of this document.

Linking problems: Definitely, the most complex problems to solve in C++ development has been the correct definition of library dependencies, linker switches and parameters for each platform. These problems have been tackled by specifying them explicitly in the CMake file ("src/CMakeLists.txt" in the source code distribution), which clearly separates each platform and its particular dependencies, to avoid collisions between them and streamline the development process.

Another problem was caused on the Mac OS X platform, where Xcode 3.1 would only link the application to the dynamic version of the library, instead of the static version. Static linking is preferred, given the easier installation workflow and the reduced dependencies, and Xcode did not offer other solution rather than deleting the dynamic version of the library, which forced Xcode to perform a static link process.

Differences in library implementations across platforms: The same library has been found to have different behaviors in different platforms, which caused some headaches to the author.

For example, Poco::UUID class has a "createRandom()" method, supposed to generate a unique 32-bit identifier. This function is broken in the Windows version of Poco (where it throws an exception when executed), and the author had to use the standard library "tmpnam()" function instead[1].

Differences in program behavior: The Active Record implementation used in the "storage" namespace of **Rem** is based on the Poco::Any class, which allows developers to store any kind of variable, like most dynamic languages do. However, C++ being a strongly typed language, whenever an object of a certain type is stored in an variable of type Poco::Any, it must be explicitly cast to the correct type when the value is retrieved later. Otherwise, the runtime generates an exception, which is different in Windows and Unix systems; in the latter, it is a Poco::BadCastException, while in Windows is an std::bad_alloc one; of course, the code reflects this with compiler directives[2].

Naming collisions: Apple's C++ compiler can also be used to generate binaries using Objective-C, another object-oriented programming language derived from C. This language defines a special keyword "nil", which has a similar semantic behavior as the "NULL" keyword in C++. However, the Poco library uses a "nil()" method in the Poco::UUID class, and to allow the code to compile in Mac OS X, a special "#undef nil" directive had to be included in parts of the code[3].

Support of the C++ standard: **Rem** makes heavy use of two "distinctive" features of C++: template metaprogramming and multiple inheritance. Compiler vendors often take some time in implementing features added to the C++ standard, and template metaprogramming is one of the features showing big differences in support among compilers. As Hutchings (2006) explains (and as shown in table 5.1.2),

[1] http://remproject.googlecode.com/svn/trunk/src/notifications/NewObjectAdded.cpp, lines 58 to 64
[2] http://remproject.googlecode.com/svn/trunk/src/tests/AnyPropertyMapTest.cpp, lines 145 to 157
[3] http://remproject.googlecode.com/svn/trunk/src/notifications/NewObjectAdded.cpp, line 41

name resolution rules is one of the strongest examples of compiler differences regarding template metaprogramming support:

> Q: Which rules do the various C++ implementations apply for name resolution in templates?
> A: I have divided implementations into three categories: CFront, those that resolve all names at the point of instantiation, like CFront did; intermediate, those that parse templates more fully, resolving some names at the point of definition and requiring disambiguation of others; and standard, those that use the standard rules. Note that there is a lot of variation among the "intermediate" implementations.

Implementation	**Versions and options**	**Name lookup rules**
Comeau C++	4.x, CFront mode	CFront
	4.x, relaxed mode; 4.0-4.2.43, strict mode	intermediate
	4.2.44-4.3.3, strict mode	standard
GNU C++ (g++)	2.8-3.3	intermediate
	3.4-4.1	standard
Metrowerks CodeWarrior	8-9, default	intermediate (?)
	8-9, -iso-templates	standard
Microsoft Visual C++	6.0	CFront
	7.0-8.0 (VS.NET 2002-2005)	intermediate

Table 5.1: Name resolution implementations in C++ compilers (Hutchings 2006)

Thankfully, as explained by Gentile (2002), Microsoft's support for C++ standards has been greatly enhanced since the release of Visual C++ .NET 2003:

> Many of Microsoft's core products have been and are built with C++, and the language has been central to Microsoft. Regrettably, Standard C++ has not been. The compiler has lagged behind the standard for most of the 90s, and the impression that many developers have is that Microsoft could care less about the standard. The good news is that with the new compiler, Microsoft has finally taken the standard very seriously, leading to this version being one of the most standards-compliant C++ compilers running today on any platform. This turnaround is nothing short of amazing.

5.2 Design Changes

The initial design of **Rem** is quite different from the final product. This is particularly visible in the "metamodel" namespace, which consists of only four classes:

- Project
- Diagram
- Element
- Member

This is due to the use of the Active Record pattern for storing instances as rows in SQLite database tables; the subclasses of "Diagrams", "Elements" and "Members" are differentiated in the respective tables thanks to entries in a "class" column.

Another strong difference between the final product and the initial design is the structure of the "storage" namespace. The final, stabilized version used by **Rem** features a multi-inheritance, template metaprogramming-based approach, inspired in the Ruby on Rails framework implementation of the Active Record pattern.

This difference was reflected in an article on the **Rem** project blog, as follows (Kosmaczewski 2008*a*):

> This new architecture replaces the "Curiously Recurring Template Pattern" used before, where ActiveRecord inherited (via template parameters) of the BelongsTo and HasMany classes:

```
template <class T
         , class P = NoParent
         , class C = NoChildren>
class ActiveRecord  : public P
                    , public C
{
public:
```

Listing 5.1: ActiveRecord class before refactoring

> Client classes used to have the following (ugly) syntax in their declarations; watch out for the brackets at the end of the declaration, and the duplication

of one of the parameters in the old HasMany declaration!

```
class Diagram : public ActiveRecord<Diagram
                        , BelongsTo<Project>
                        , HasMany<Element, Diagram> >
{
public:
```

Listing 5.2: ActiveRecord subclass before refactoring

...I refactored the whole family of classes, which now look like this...

```
template <class T>
class ActiveRecord : public virtual Persistable
{
```

Listing 5.3: ActiveRecord class after refactoring

And this means that now, client classes are defined in this (much more readable) way:

```
class Diagram : public ActiveRecord<Diagram>
              , public BelongsTo<Project>
              , public HasMany<Element>
{
    public:
```

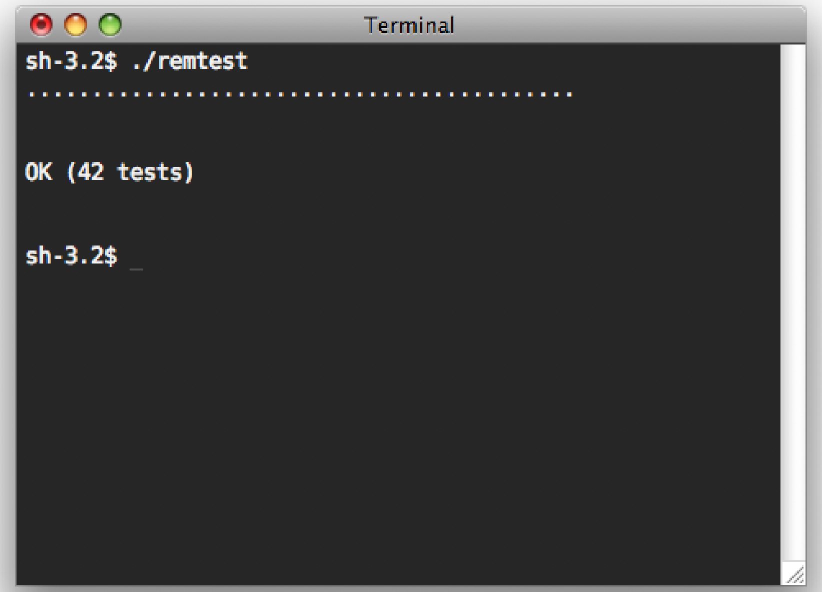
Listing 5.4: ActiveRecord subclass after refactoring

It is important to point out that the whole process of refactoring would not have been possible without a strong strategy of unit testing, described in the following section.

5.3 Testing

The whole process to bring **Rem** to reality was driven by a strong testing strategy: 42 different unit tests are available in the source code distribution, targeted towards the "storage" layer, based in rather complex C++ constructs, using multiple inheritance and template-based metaprogramming. These features were prone to be broken repeatedly, given the extreme volatility of the design, and unit tests proved to be essential as guarantees of the stability of the interfaces.

Given that the unit tests are portable, they have also unveiled potential portability problems, as well as implementation differences between platforms. Figure 5.1 shows a successful execution of the unit tests suite in the console.

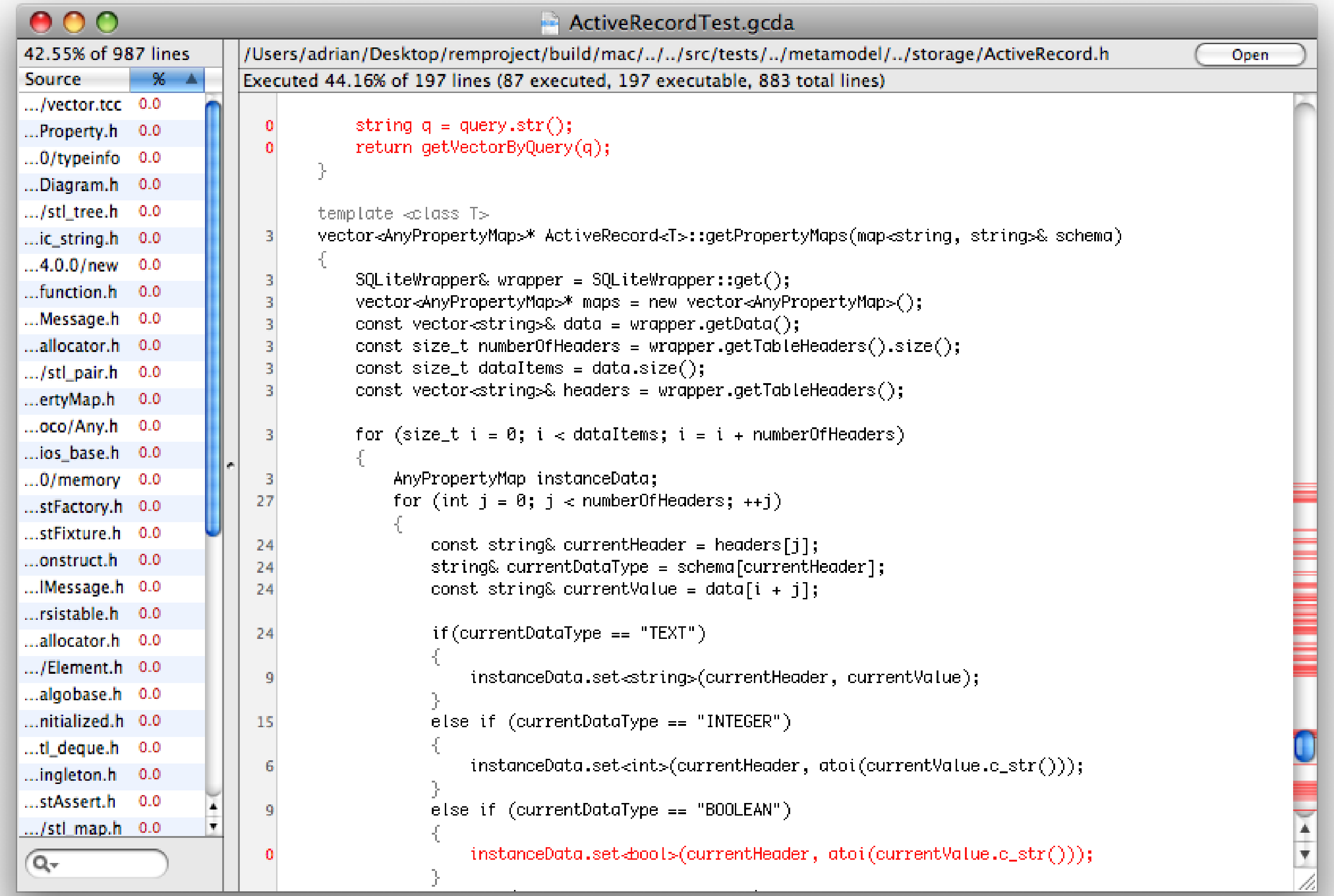


```
sh-3.2$ ./remtest
..........................................


OK (42 tests)


sh-3.2$ _
```

Figure 5.1: Test run in the console

Another important part of the testing strategy was the use of a code coverage tool, to ensure that every line of source code was tested at least by one test in the suite. Figure 5.2 shows how the code coverage information is shown using the **CoverStory**[4] tool.

[4]http://code.google.com/p/coverstory/



ActiveRecordTest.gcda

```
42.55% of 987 lines
Source          %
.../vector.tcc  0.0
...Property.h   0.0
...0/typeinfo   0.0
...Diagram.h    0.0
.../stl_tree.h  0.0
...ic_string.h  0.0
...4.0.0/new    0.0
...function.h   0.0
...Message.h    0.0
...allocator.h  0.0
.../stl_pair.h  0.0
...ertyMap.h    0.0
...oco/Any.h    0.0
...ios_base.h   0.0
...0/memory     0.0
...stFactory.h  0.0
...stFixture.h  0.0
...onstruct.h   0.0
...lMessage.h   0.0
...rsistable.h  0.0
...allocator.h  0.0
.../Element.h   0.0
...algobase.h   0.0
...nitialized.h 0.0
...tl_deque.h   0.0
...ingleton.h   0.0
...stAssert.h   0.0
.../stl_map.h   0.0
```

/Users/adrian/Desktop/remproject/build/mac/../../src/tests/../metamodel/../storage/ActiveRecord.h

Open

Executed 44.16% of 197 lines (87 executed, 197 executable, 883 total lines)

```
 0          string q = query.str();
 0          return getVectorByQuery(q);
        }

        template <class T>
 3      vector<AnyPropertyMap>* ActiveRecord<T>::getPropertyMaps(map<string, string>& schema)
        {
 3          SQLiteWrapper& wrapper = SQLiteWrapper::get();
 3          vector<AnyPropertyMap>* maps = new vector<AnyPropertyMap>();
 3          const vector<string>& data = wrapper.getData();
 3          const size_t numberOfHeaders = wrapper.getTableHeaders().size();
 3          const size_t dataItems = data.size();
 3          const vector<string>& headers = wrapper.getTableHeaders();

 3          for (size_t i = 0; i < dataItems; i = i + numberOfHeaders)
            {
 3              AnyPropertyMap instanceData;
27              for (int j = 0; j < numberOfHeaders; ++j)
                {
24                  const string& currentHeader = headers[j];
24                  string& currentDataType = schema[currentHeader];
24                  const string& currentValue = data[i + j];

24                  if(currentDataType == "TEXT")
                    {
 9                      instanceData.set<string>(currentHeader, currentValue);
                    }
15                  else if (currentDataType == "INTEGER")
                    {
 6                      instanceData.set<int>(currentHeader, atoi(currentValue.c_str()));
                    }
 9                  else if (currentDataType == "BOOLEAN")
                    {
 0                      instanceData.set<bool>(currentHeader, atoi(currentValue.c_str()));
                    }
```

Figure 5.2: Test coverage as shown by the CoverStory tool

5.4 Coding Conventions

During the creation of **Rem** the following set of coding conventions was used, generated using the Coding Standard Generator tool (Rosvall 2005) and adapted for the project:

- **Entity Naming**
  - Names shall begin with an upper case letter and words shall begin with an upper case letter.
  - Variables shall begin with a lower case letter.
  - Member variables shall be prefixed with "_".
  - Constants shall begin with an upper case letter and shall be upper case.
  - Functions shall begin with a lower case letter.
  - Macros shall be upper case.
- **Names**
  - Use sensible, descriptive names.
  - Only use english names.
  - Variables with a large scope shall have long names, variables with a small scope can have short names.
  - Use namespaces for identifiers declared in different modules
  - Use name prefixes for identifiers declared in different modules
- **Indentation and Spacing**
  - Braces shall follow "Exdented Style".
  - Braces shall be indented 4 columns to the right of the starting position of the enclosing statement or declaration.

    Example:

```
void f(int a)
{
    int i;
```

```
        if (a > 0)
        {
            i = a;
        }
        else
        {
            i = a;
        }
    }
```

Listing 5.5: Indentation and spacing guidelines

  - Function parameters shall be lined up with all parameters in the same line.
  - Loop and conditional statements shall always have brace enclosed sub-statements.
  - Braces without any contents may be placed on the same line.
  - Each statement shall be placed on a line on its own.
  - Declare each variable in a separate declaration.
  - For declaring pointers and reference the “*” and “&” shall be at the right side of the type name with no spaces between them.
  - All binary arithmetic, bitwise and assignment operators and the ternary conditional operator (?:) shall be
  - surrounded by spaces; the comma operator shall be followed by a space but not preceded; all other operators shall not be used with spaces.
  - Lines shall not exceed 78 characters.
  - Do not use tabs for indentations. All indentations should be done with four (4) space characters.

- **Comments**
  - Comments shall be written in english
  - Comments shall use the C-style.
  - Comments shall use the C++-style.

  - Use Doxygen style comments.
  - Multiple line comments shall be split in one comment per line, each having the /* and */ markers on the same line.
  - All comments shall be placed above the line the comment describes, indented identically.
  - Use #ifdef instead of /* ... */ to comment out blocks of code.
  - Every class shall have a comment that describes its purpose.
  - Every function shall have a comment that describes its purpose.
- **Files**
  - There shall only be one externally visible class defined in each header file.
  - There shall only be one externally visible function defined in each header file.
  - File name shall be treated as case sensitive.
  - C++ source files shall have extension ".cpp".
  - C++ header files shall have extension ".h".
  - Header files must have include guards.
  - The name of the macro used in the include guard shall have the same name as the file (excluding the extension) followed by the suffix "_H_".
  - Header files shall be self-contained
  - System header files shall be included with ¡¿ and project headers with "".
  - Put #include directives at the top of files.
  - Do not use absolute directory names in #include directives.
  - Use relative directory names in #include directives.
  - Each file must start with a copyright notice.
  - Each file must contain a revision marker.
- **Declarations**
  - Do not provide names of parameters in function declarations.

    - Use a typedef to define a pointer to a function.
    - Do not use exception specifications.
    - Declare inherited functions virtual.
    - Do not use global variables.
    - Prefer singleton objects to global variables.
    - Do not use global using declarations and using directives in headers.
    - Use specific using directives after the includes.
    - The parts of a class definition must be public, protected and private.
    - Declare class data private.
- **Statements**
    - Never use gotos.
    - All switch statements shall have a default label.
- **Other Typographical Issues**
    - Avoid macros if possible.
    - Do not use literal numbers other than 0 and 1.
    - Use prefix increment/decrement instead of postfix increment/decrement when the value of the variable is not used.
    - Write conditional expressions like: if (6 == errorNum) ...
    - Do not rely on implicit conversion to bool in conditions.
    - Use the new cast operators.

5.5 Communication

Another differentiation factor of **Rem** is the extensive use of open communication with the open-source community. Given the high number of open-source projects available, including those used in **Rem**, providing live feedback about the development process was a key element throughout the project.

This communication has taken place mainly through four channels:

The Home Page: http://remproject.org/ was updated regularly, with news about the advancement of the project. The website not only hosts a blog, allowing the author to provide updates about **Rem**, but also hosts the complete history of past releases[5]. Users interested in **Rem** can interact in this website, leaving messages in blog posts, downloading binaries and documentation for their platforms, and even getting support through the forums[6].

Twitter: **Rem**'s account, visible at http://twitter.com/remproject, provides a quick way to update interested users about new features, project updates and to interact with the software development community interested in **Rem**.

Google Code: The source code of **Rem** is publicly accessible to users at http://remproject.googlecode.com/, which not only hosts a Subversion server, but also a wiki (a special system which can be freely edited by users) and a ticketing support system, providing a space for end users to communicate bug reports and other problems.

Ohloh: The author has opened an account at http://www.ohloh.net/projects/rem, which helps open source projects to gather statistics about the project, its users and its developers, similar to the way in which social networks connect users in other areas. It also provides source code metrics, history about source control commits and even the geographical localization of users and developers.

[5]http://remproject.org/releases/
[6]http://remproject.org/forums/

Chapter 6

Results and Evaluation

This chapter contrasts the final outcome of the project with the initial requirements, highlighting strengths and weaknesses in four different areas:

1. Architecture
2. Extensibility
3. Cross-Platform Compatibility
4. Project Size Statistics

6.1 Architecture

This section highlights the most important aspects of the architecture of **Rem** and their impact in the future evolution of the project.

6.1.1 Model-View-Controller

The architecture of **Rem** is based in the Model-View-Controller pattern; this pattern states a clear separation between the graphical representation of an entity, and its internal structure. In the case of **Rem** the following elements compose this architecture:

Model: The classes of the “metamodel” namespace are **Rem**’s models, defining the relationships between projects, diagrams, and elements.

View: The classes of the “ui” namespace provide a visual representation of the model.

Controller: The singleton class controllers::FileController provides a unique point of communication between the model and the view; any change in the user interface is translated into model changes through this class, and whenever a file is opened, the controller notifies the user interface about the model changes.

This separation in layers allows **Rem** to be extended into a command-line utility in the future, which will provide power users with a strong tool to automate tasks and procedures related to UML diagrams, particularly code-generation tasks.

6.1.2 Reduction of Dependencies

The dependencies in C++ classes are done through #include directives, usually implemented through guards, to avoid multiple inclusion of the same file:

```
#ifndef NEWDIAGRAMADDED_H_
#include "../notifications/NewDiagramAdded.h"
#endif
```

Listing 6.1: #include directive in the ui::ProjectComponent class

However, this relationship has a major drawback: it can lengthen compilation cycles, since any change in a header file triggers a recompilation of all the dependent implementation modules (the “.cpp” files). In a large project this can be translated to an excessively long recompilation cycle, even for small changes.

To avoid this problem, the project uses a convention: since a pointer is simply an integer, the C++ standard allows to “forward” class declarations of types used as pointers or references, requiring the #include statement in the implementation file only:

```
class ProjectTabbedComponent; // Forward declaration

class ProjectComponent : public Component
```

```
{
private:
    ProjectTabbedComponent* _tabs;
```

Listing 6.2: Forward class declarations instead of #include statements

The use of this technique greatly reduces the time required for recompilation when changing interfaces on header files.

6.1.3 Usage of Abstract Base Classes

The structure of the application supports several points of extensibility, most importantly through the use of abstract base classes. Examples of these are

- ui::Figure
- ui::UMLDiagram
- ui::DiagramToolbar
- ui::LineFigure
- storage::Persistable

These classes are either abstract or provide virtual methods with default behaviors, which can be overridden by subclasses to provide specialized implementations whenever possible.

6.1.4 Observer Pattern

To reduce dependencies and coupling between components, the application uses the Observer design pattern, through the use of the Poco::NotificationCenter class, and several classes inheriting from Poco::Notification. To post notifications, clients use the singleton Poco::NotificationCenter class:

```
ExportDiagramAsPNG* notification = new ExportDiagramAsPNG();
NotificationCenter::defaultCenter().postNotification(notification);
```

Listing 6.3: Posting a notification using the Poco::NotificationCenter class

To be notified of a notification, any class can register an observer into the Notification-Center:

```
NObserver<ProjectComponent, NewProjectCreated>* observer;
observer = new NObserver<ProjectComponent, NewProjectCreated>(*this,
                        &ProjectComponent::handleNewProjectCreated));
```

Listing 6.4: Listening for notifications using observers

When the Poco::NotificationCenter receives a notification, it will call the corresponding method, passing the notification as parameter:

```
void handleNewProjectCreated(const AutoPtr<NewProjectCreated>& notif)
{
    // ...
}
```

Listing 6.5: Method handling a notification in the controllers::FileController class

The biggest advantage of this approach is that it reduces the number of direct dependencies between code fragments, thus reducing the time for recompilations and allowing several objects to be alerted at once of an event. In the case of user interfaces, this is useful to update parts of the screen (menus, buttons) depending on the user behavior, and allowing the system to evolve gracefully in the future without breaking existing interdependencies.

On the other hand, by their very nature, notifications hide these dependencies completely, and as such it is extremely complicated to have a picture of “who is responsible for what”, which affects the maintainability of the code in the long run. Another drawback is that the Poco::NotificationCenter does not offer hints about the order in which observers are called, and as such, code blocks handling notifications should not have any dependencies between them. Finally, it is important that these code blocks do not have side effects or complex behaviors, or even fire notifications themselves, since this added complexity makes code maintainability even harder.

6.2 Extensibility

As of version 1.0, **Rem** can only be used to create use-case diagrams. However, the architecture of the software allows it to be extended to the other diagram types following these steps:

Subclass the ui::UMLDiagram class: All diagrams in **Rem** are subclasses of the ui::UMLDiagram abstract base class, which provides "hooks" so that inheritors can save and retrieve their state from the file. When this is done, the ui::ProjectComponent class can be extended to add new instances of this class, similarly as how it's done currently:

```
UseCaseDiagram* addUseCaseDiagram(const string& uniqueId)
{
    int index = _tabs->getNumTabs();
    UseCaseDiagram* diagram = new UseCaseDiagram(uniqueId);
    DiagramComponent* diagramComponent;
    diagramComponent = new DiagramComponent(diagram, index);
    _tabs->addTab(String("Use Case Diagram"),
        Colours::white, diagramComponent, true);
    _tabs->setCurrentTabIndex(index);
    return diagram;
}
```

Listing 6.6: Creation of new diagrams in the ui::ProjectComponent class

New diagrams subclasses must implement three pure virtual methods:

1. virtual void addFigure(const AutoPtr<NewFigureAdded >&) = 0;
2. virtual void populateFrom(Diagram*) = 0;
3. virtual DiagramToolbar* createToolbar() = 0;

Subclass the ui::DiagramToolbar class: As shown above, the ui::ProjectComponent class "wraps" subclasses of the ui::UMLDiagram class into a ui::DiagramComponent instance, which provides scrolling capabilities. When this happens, the ui::DiagramComponent requests a toolbar to the ui::UMLDiagram; as such, there should be a subclass of the ui::DiagramToolbar class for every type of diagram, providing the buttons required to operate in that particular kind of diagram. This is done through the pure virtual ui::UMLDiagram::createToolbar() method:

```
DiagramComponent::DiagramComponent(UMLDiagram* diagram, const int index)
: Component()
, _index(index)
, _viewport(new Viewport())
, _diagram(diagram)
, _toolbar(diagram->createToolbar())
, _isActive(false)
```

Listing 6.7: Fragment of the ui::DiagramComponent constructor

Creation of Figures: All objects that can be dropped into a diagram are subclasses of the ui::Figure abstract base class; this requires subclasses to implement three pure virtual methods:

1. virtual void drawFigure(Path&) const = 0;
2. virtual void updateSpecificProperties() = 0;
3. virtual void setSpecificProperties() = 0;

Creation of suitable buttons for the toolbar: For each new ui::Figure subclass, it is required a new Button subclass for the corresponding ui::DiagramToolbar.

Creation of ad-hoc lines or arrows: To adhere to the UML standard, arrow shapes and style change depending on the meaning of the relationship between entities. All lines of the **Rem** project are implemented as subclasses of the base class ui::LineFigure, which provides a virtual drawLine() method, which can be overridden by subclasses to provide custom renderings.

Add new notifications: To add new behaviors to **Rem** it might be useful to add new notifications to the “notification” namespace. Notifications are subclasses of the Poco::Notification class, and can convey extra information, such as pointers or numbers, indicating multiple observers at once about a particular event.

As explained previously, this reduces coupling between components using the Observer pattern, making the software easier to extend.

Given the above steps, **Rem** could virtually be extended to support any kind of diagrams, with several advanced features such as resizing, “drag and drop”, multiple line types, and multiple selection support.

6.3 Features

Given the short deadline and the complex requirements, **Rem** currently does not support features such as:

- Class diagrams;
- Sequence diagrams;
- Aspect-Oriented constructs;
- Code generation for MDA tasks.

However, thanks to the extension points highlighted above, future evolutions of **Rem** are made possible, including the support for the lacking features enumerated above. In particular, to provide the support for MDA tasks, the next step would be to provide class diagrams, so that software engineers can translate into working code the designs of families of classes, into any programming language, as required.

6.4 Cross-Platform Compatibility

This section will provide an overview of the current cross-platform capabilities of **Rem**, both from the point of view of the software developer and the user.

6.4.1 Compilation

Developers of **Rem** are not limited to a single operating system; thanks to the C++ ISO standard, and tools like CMake and Doxygen, the future evolution (maintenance and new features) of the application is completely platform-independent, and it could be possible to port the application to other operating systems, particularly Solaris.

However, supporting multiple platforms has the drawback of increasing the need and complexity of quality management. Whenever a new version of **Rem** is released, the system must be tested in all supported operating systems, including unit testing execution and user-based testing as well, to ensure that the new release does not include regression bugs or other functional problems.

6.4.2 Execution

End users have a double benefit from cross-platformity: not only they can choose to run the application in whichever operating system they use, they can also exchange files seamlessly between those, since SQLite files are completely platform independent, as explained in previous chapters. The sample file provided with the source code of **Rem** can be used to prove that a file created in one platform can be read and written in other operating systems without any problem.

6.5 Project Statistics

The figures in table 6.5 were gathered using the open source **ohcount**[1] source code line count utility.

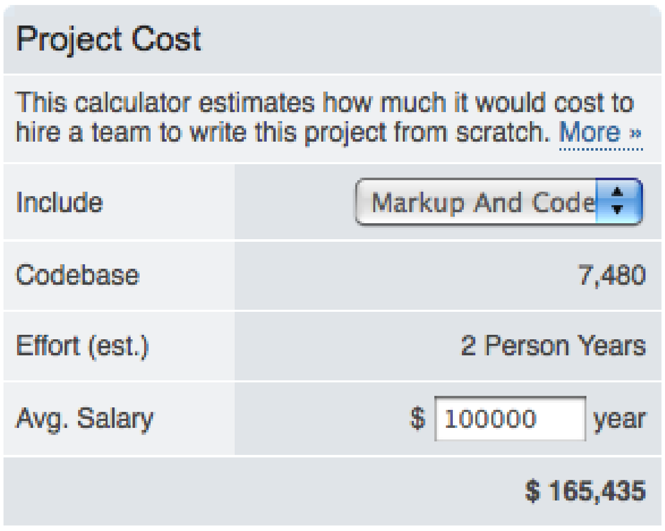
Namespace	Files	Code	Comment	Comment %	Blank	Total
controllers	2	403	190	32.0%	93	686
metamodel	8	316	459	59.2%	112	887
notifications	28	542	1327	71.0%	234	2103
storage	13	1463	1448	49.7%	408	3319
tests	17	1417	910	39.1%	460	2787
ui	46	3159	3596	53.2%	942	7697
utility	2	132	194	59.5%	42	368
Total	**116**	**7432**	**8124**	**52.2%**	**2291**	**17847**

Table 6.1: Source code statistics

The Ohloh website, mentioned in section 5.5, also offers a calculation of the effort required to create a project similar to **Rem**, shown in figure 6.1.

As explained by Luckey (2006) in the Ohloh website,

[1] http://labs.ohloh.net/ohcount



Project Cost

This calculator estimates how much it would cost to hire a team to write this project from scratch. More »

Include	Markup And Code
Codebase	7,480
Effort (est.)	2 Person Years
Avg. Salary	$ 100000 year
	$ 165,435

Figure 6.1: COCOMO I estimation by Ohloh (Luckey 2006)

> We use a software costing model called COCOMO. There are several variations of this model, each with different precisions. We base our calculations on the simplest form of these models (the "Basic COCOMO" model). We currently lack the information required to produce the more advanced models.
>
> For those familiar with the details of the model, we are using coefficient values of a=2.4 and b=1.05.

As Boehm (1998) explains in the COCOMO II manual, the approach used by Ohloh uses the first version of the COCOMO model, where the "a" and "b" factors mentioned above correspond to an "organic" approach to software development:

> The data analysis on the original COCOMO indicated that its projects exhibited net diseconomies of scale. The projects factored into three classes or modes of software development (Organic, Semidetached, and Embedded), whose exponents B were 1.05, 1.12, and 1.20, respectively. The distinguishing factors of these modes were basically environmental: Embedded-mode projects were more unprecedented, requiring more communication overhead and complex integration; and less flexible, requiring more communications overhead and extra effort to resolve issues within tight schedule, budget, interface, and performance constraints.

These factors were replaced in the COCOMO II model by the Precedentedness (PREC) and Development Flexibility (FLEX) factors, which describe in greater detail the complexity of the software development task.

Chapter 7

Conclusions

This chapter will provide some final thoughts about **Rem**, as well as a discussion of its legacy, in terms of strengths and weaknesses.

7.1 The Project

It must be said that the whole project had a scope much bigger than what could be done during the course of this degree project; the author acknowledges that this project has been one of the most complex programming tasks he has ever tackled by himself.

There were different dimensions of complexity in it, all of which made it a challenging but rewarding and thoroughly enjoyable project:

- The cross-platform nature of the project;
- The use of advanced features of C++, such as templates and multiple inheritance;
- The design and usability of a complex user interface;

7.2 Strengths and Weaknesses

Regarding the initial objectives, the project has succeeded in providing a stable, documented, and cross-platform foundation to build upon. As described in section 6.2, the project can be extended to not only support new UML diagram types, but to support virtually any type of diagramming requirement.

There is another positive outcome of this project, which is the implementation of the Active Record pattern; the classes in the storage namespace are highly reusable outside of **Rem**, providing software developers with a useful C++ library to use in other projects.

There are, however, several weaknesses (both technical and functional) which could become critical blocking issues in the future evolution of **Rem**:

The aggressive use of notifications: However interesting in terms of decoupling and reduced dependencies, notifications are currently used pervasively throughout the application, even when the Juce framework provides a simpler, one-to-one event dispatching mechanism.

Limited testing: **Rem** only provides unit tests for the lower layers of the application, namely the controllers, metamodel, utility and storage namespaces, but not for the classes used in the GUI, nor in the notifications namespace. It is fundamental, to ensure a smooth and easier maintenance in the future, to expand the testing strategy, and provide quality management measures for these elements of code as well.

The lack of import / export to the XMI standard: Although XMI export was one of the key deliverables mentioned in the introduction, there is not such feature in version 1.0 of **Rem**. This is a real threat to the interoperability of **Rem** with other UML diagramming applications, and should be addressed as soon as possible.

The rigidity of the Active Record pattern implementation: As explained in sections 2.3.3 and 5.2, **Rem** bundles an implementation of the Active Record pattern heavily inspired in the Ruby on Rails framework. This approach is based in multiple inheritance, which currently avoids the possibility of having a class implementing more than one “HasMany” or “BelongsTo” relationships, given that this would lead to name collisions currently unavoidable.

A possible solution to this problem would be to use template methods in these two template classes, each specifying the parent / child relationship explicitly, but this might mean rewriting and retesting the library, almost from scratch.

7.3 The Future

As stated in the introduction, one of the biggest successes of **Rem** would be to become a tool used both in academia as well as in the industry, being not only a useful tool but also a learning instrument. **Rem** has been designed to be extensible, portable and extremely simple to understand and use.

The author strongly hopes that other students and software engineers will find this tool useful, interesting and flexible for their study or day-to-day work.

Appendices

Source Code

The source code of **Rem**, together with its complete history, is freely available with a Subversion client:

```
$ svn checkout http://remproject.googlecode.com/svn/trunk/ remproject-read-only
```

Alternatively, users can also “export” a non-working copy, which does not allow to commit changes back to the repository:

```
$ svn export http://remproject.googlecode.com/svn/trunk/ remproject-read-only
```

It can also be browsed online, including its history, from
http://code.google.com/p/remproject/source/browse/

A zip file with version 1.0 of the **Rem** source code can be downloaded from
http://remproject.org/releases/1.0.0/Rem-1.0.0-src.zip

Binaries

The complete history of binaries for **Rem** is available from the project home page:
http://remproject.org/releases/

The latest version is always available at:
http://remproject.org/downloads/

The binaries of version 1.0 of **Rem** can be downloaded from the following locations:
Windows: http://remproject.org/releases/1.0.0/Rem-1.0.0-Win32.exe
Linux: http://remproject.org/releases/1.0.0/Rem-1.0.0-Linux.tar.gz

Mac: http://remproject.org/releases/1.0.0/Rem-1.0.0-Tiger.dmg and http://remproject.org/releases/1.0.0/Rem-1.0.0-Tiger-Installer.dmg.

Source Code Documentation

The source code documentation extracted using Doxygen is freely available from:
http://remproject.org/releases/1.0.0/Rem-1.0.0-doc.chm as a CHM (Compiled HTML) file, which can be viewed natively in Windows systems;
http://remproject.org/releases/1.0.0/Rem-1.0.0-doc.pdf as a PDF (Portable Document Format) file, and
http://remproject.org/releases/1.0.0/Rem-1.0.0-doc.html.zip as a zip file with the HTML version.

Build Instructions

This section provides the necessary instructions to build **Rem** directly from source code, in each of the three supported operating systems.

Microsoft Windows XP SP 2

To build **Rem** in Windows XP Service Pack 2 follow these instructions:

1) Install Microsoft Visual C++ 2008 Express Edition

Free download from here:
http://www.microsoft.com/express/download/

It is recommended to download the “Offline Install DVD”, available at the bottom of the page. You only need to install the C++ IDE and libraries from that DVD.

2) Download and install QuickTime for Windows

JUCE depends on the QuickTime libraries for displaying media files. You can download and install QuickTime from this address:
http://www.apple.com/quicktime/download/

3) Install the ASIO SDK

Juce requires this SDK to be available. Follow the instructions in this forum post:
http://www.rawmaterialsoftware.com/juceforum/viewtopic.php?p=12107#12107

The ASIO drivers can be downloaded from here:
http://www.steinberg.net/324+M52087573ab0.html

In that page, follow the link at the bottom, agree to the end-user license agreement and fill the form to access the download link. Install the libraries in "C:/ASIOSDK2/".

4) Checkout and build POCO

POCO is bundled with the file "lib/poco/Foundation/Foundation_vs80.sln" which can be opened with Visual C++ 2008 Express Edition (you will be asked to upgrade it to the Visual Studio 2008 format). Right click on the project and select "Build", in each of the four available configurations. The generated libraries will be generated in "lib/poco/lib" and the DLLs will be installed in "lib/poco/bin".

5) Build JUCE

JUCE is bundled with a Visual Studio solution, that can be used to build the library. Open the "lib/juce/build/win32/vc8/JUCE.sln" file (you will be asked to upgrade it to the Visual Studio 2008 format) and right-click on the JUCE project inside. Go to "Configuration Properties - C/C++ - General" and modify the the "Additional Include Directories" to the value: "C:/Program Files/QuickTime";"C:/ASIOSDK2/common". Right-click on the project and select "Build", in both Release and Debug configurations.

6) Install CppUnit

Download
http://downloads.sourceforge.net/cppunit/cppunit-1.12.1.tar.gz
and unzip it into “C:/cppunit-1.12.1”. Open the “C:/cppunit-1.12.0/src/CppUnitLibraries.sln” file and build the “cppunit” project. The resulting libraries are stored at “C:/cppunit-1.12.0/lib”.

7) Install SQLite

Download the precompiled DLL for Windows here:
http://www.sqlite.org/sqlitedll-3_5_8.zip
Unzip it and install the files “sqlite3.dll” and “sqlite3.def” in “lib/sqlite”.

8) Generate the DEF from the SQLite DLL

In order to use the DLL from within Rem, you need to generate a DEF file out of it. To do this, follow the instructions at:
http://support.microsoft.com/kb/131313

1. cd lib/sqlite
2. LIB /DEF:sqlite.def

This will generate the “sqlite3.exp” file required by the Rem Visual Studio solution.

9) Install CMake 2.4

There is an installer available here:
http://www.cmake.org/files/v2.4/cmake-2.4.8-win32-x86.exe

Do not use CMake 2.6 (the latest version available at the time of this writing) since there is a regression bug that affects the command-line script that builds Rem from scratch:
http://www.cmake.org/Bug/view.php?id=7222

10) Install NSIS 2.37

NSIS (Nullsoft Scriptable Install System) is used by CMake to generate installers for Windows. You can download it from here:
http://prdownloads.sourceforge.net/nsis/nsis-2.37-setup.exe?download

11) Build Rem in Visual Studio

Open the “build/windows/Rem.sln” file; select the “Release” configuration, click on the “Rem” project on the solution tree, and select “Build Rem” on the “Build” menu.

12) Build Rem using CMake

Use the “build/windows/build.bat” file, which generates an NMake makefile from the CMake file, and then uses NMake (available with Visual C++ 2008 Express Edition) to build the application and the installer from the command line.

Kubuntu Linux 7.10

To build **Rem** in Kubuntu Linux 7.10 “Gutsy Gibbon” follow these instructions:

References

Much of the explanations for building JUCE in Kubuntu are taken from this entry in **Rem** author’s blog:
http://kosmaczewski.net/2007/11/16/building-juce-on-kubuntu-710/

Note about processor architectures

JUCE cannot run on Linux on PowerPC processors, given that it relies on assembler code for dealing with endianness issues, as shown here:
http://www.koders.com/c/fid26F230513834417D1CC7BE6FDF5CE455DA49BE09.aspx
This rules out the possibility of running Rem on PowerPC systems, like G3, G4 and G5

Macs running Linux. For the moment, Rem can only be built with Linux running on processors supporting the i386 architecture.

1) Lua

Make sure that you have the Lua programming language installed; you can use your favorite package manager to install it, or you can grab the source files from
http://www.lua.org/

2) SQLite 3 and CppUnit

Using Synaptic manager or any other package manager, install the “libsqlite3” and “libcppunit-dev” libraries.

3) Libraries required by JUCE

Make sure that you have the following libraries installed in your Kubuntu installation using Synaptic or any other package manager, as specified in this JUCE forum post:
http://www.rawmaterialsoftware.com/juceforum/viewtopic.php?t=1312

- libx11-dev
- libasound2-dev
- libfreetype6-dev
- libxinerama-dev
- libglu1-mesa-dev
- libglut3-dev (with its dependency freeglut3-dev too)

4) Libraries required by POCO

Install “libssl-dev” using Synaptic or any other package manager.

5) Download premake from Sourceforge

You can download premake
http://premake.sourceforge.net/
from this link:
http://prdownloads.sf.net/premake/premake-linux-3.4.tar.gz

Unzip the file and install the binary where you want (typically /usr/bin). You have to do this manually, since premake is not available through the Synaptic package manager, in any repository.

You can also build it from source, downloading and extracting the following file:
http://prdownloads.sf.net/premake/premake-src-3.6.zip

6) Install CMake 2.6

Download the source files for CMake and install it following the instructions:
http://www.cmake.org/files/v2.6/cmake-2.6.0.tar.gz

7) Build JUCE

- “cd” into the lib/juce/build/linux folder.
- Run “sh runpremake” which will use premake and Lua to create a makefile
- Run “make” (which is equal to “make CONFIG=Debug”) or “make CONFIG=Release” to build the library; a couple of minutes later you'll have a “juce/bin/libjuce_debug.a” and a “juce/bin/libjuce.a” library files ready to use.

8) Build POCO

- “cd” into the lib/poco folder.
- Type “chmod 755 configure” to make the “configure” script executable.
- Type “chmod 755 build/script/*” to make all build scripts executable.
- Type “configure”, “make” and “sudo make install” to build and install the POCO library. This operation might take several minutes.

9) Build Rem

The “build.sh” shell script in this folder shows how to build Rem and the distribution package using CMake. The resulting binaries will be placed in a “bin” subfolder at the root of the current distribution of Rem.

Mac OS X 10.5 “Leopard”

To build **Rem** in Mac OS X 10.5 “Leopard” (PowerPC or Intel) follow these instructions.

1) Install Xcode

Install the Xcode developer tools (bundled with Mac OS X). Rem has been successfully built with both Xcode 3.0 (bundled with Leopard) and 3.1 (bundled with the iPhone developer tools).

2) CMake 2.6

- Download CMake 2.6 from
  http://www.cmake.org/files/v2.6/cmake-2.6.0-Darwin-universal.dmg
- Open the DMG file, and execute the installer in the disk image.
- When prompted to install the command-line utilities in /usr/bin, answer “YES”

3) CppUnit 1.12.1

Install CppUnit in the usual Unix paths, with the following commands:

```
$ mkdir  /Desktop/cppunit/
$ cd  /Desktop/cppunit/
$ curl http://switch.dl.sourceforge.net/sourceforge/cppunit/cppunit-1.12.1.tar.gz
> cppunit-1.12.1.tar.gz
$ tar xvfpz cppunit-1.12.1.tar.gz
$ cd cppunit-1.12.1
```

To build Universal Binaries of the CppUnit library, follow these instructions taken from the CppUnit wiki,
http://cppunit.sourceforge.net/cgi-bin/moin.cgi/BuildingCppUnit1#head-606831052dc6f25163a5f79aec04

When building on an Intel machine:

```
$ ./configure --disable-dependency-tracking CXXFLAGS=‘‘-arch ppc -arch i386
-gdwarf-2 -O2’’
$ make AM_LDFLAGS=‘‘-XCClinker -arch -XCClinker ppc -XCClinker -arch -XCClinker
i386’’
$ sudo make install
```

When building on a PowerPC machine (with Universal SDK installed):

```
$ ./configure --disable-dependency-tracking CXXFLAGS=‘‘-isysroot
/Developer/SDKs/MacOSX10.5.sdk -arch ppc -arch i386 -gdwarf-2 -O2’’
$ make AM_LDFLAGS=‘‘-XCClinker -arch -XCClinker ppc -XCClinker -arch -XCClinker
i386 -XCClinker -isysroot -XCClinker /Developer/SDKs/MacOSX10.5.sdk’’
$ sudo make install
```

This last step will install the headers in /usr/local/include/cppunit and the libraries in /usr/local/lib.

4) Fix an incompatibility in the 10.4u SDK

Type the following command in Terminal.app

```
$ sudo ln -s /Developer/SDKs/MacOSX10.4u.sdk/usr/lib/crt1.o
/Developer/SDKs/MacOSX10.4u.sdk/usr/lib/crt1.10.5.o
```

5) Get a Universal Binary version of POCO

The POCO libraries included in the Rem source distribution (as an “svn:externals”) is not prepared to be built as a Universal Binary. You can use it to create platform-specific (i.e., only PowerPC or Intel) versions of Rem.

To download a precompiled version of POCO as Universal Binary, go to this forum post: http://www.openframeworks.cc/forum/viewtopic.php?p=2371#2371 There is a link to download the lib there:

http://openframeworks.cc/files/poco/poco-universal.zip Unzip the file and place it in the “lib” subfolder of this project, together with the “juce” folder.

6) Build JUCE 1.45

Go to lib/juce/build/macosx and open the Juce.xcodeproj file with Xcode.

1. In Xcode 2.5 the library compiles out of the box.
2. In Xcode 3 (Leopard-only):
   (a) Select the “Juce” target and open the “Info” dialog box, “Build” tab.
   (b) Remove the dependencies to GCC 3 and to the Mac OS X 10.2 SDK. Instead, specify GCC 4.0.
   (c) In the “Base SDK Path” entry specify “$(DEVELOPER_SDK_DIR)/MacOSX10.4u.sdk” for Release and Debug
   (d) In the “Architectures” entry specify “$(NATIVE_ARCH)” for Debug, and “ppc i386” for Release.
3. Build the solution using the “Build” button in the toolbar, in both Debug and Release modes.

7) Install Doxygen

Install Doxygen using MacPorts:

```
$ sudo port install doxygen
```

8) Build Rem using Xcode

Open build/mac/rem.xcodeproj and click the “Build” button on the toolbar, either in Release or Debug mode. The solution should compile without problems now.

The Xcode project is configured to build a Universal Binary of Rem and remtest only in “Release” mode. In “Debug” configuration, the binaries built will only work in the same architecture in which Xcode is running (as specified by the “$(NATIVE_ARCH)” configuration value).

9) Build Rem using CMake

The “build.sh” shell script in this folder shows how to use CMake, CPack and CTest to build the application from the command line.

10) Build the documentation with Doxygen

Type the following command at the root of the project (where the Doxyfile file resides):

```
$ /opt/local/bin/doxygen
```

Note about PDF output with Doxygen

If you want to create a PDF output using Doxygen, you should install first the MacTex distribution, freely available from http://www.tug.org/mactex/

Download the disk image file from http://mirror.ctan.org/systems/mac/mactex/MacTeX.dmg

Open the image file and execute the installer inside. This will install all the required tools to generate PDF files from the Doxygen documentation.

Note about dynamic libraries and Xcode

Xcode has a feature (bug?) when linking executables to static libraries: if both the dynamic and static version of the same library are available (which usually is the case, for example with CppUnit), Xcode will link to the dynamic version, and there is no configuration possible to change this behavior. The only possible solution to get a statically-linked executable is to delete the dynamic library, which forces Xcode to link to the static version. This way, the deployment of the application is easier (even if the resulting binary is obviously larger)

Bibliography

Alexandrescu, A., Meyers, S. & Vlissides, J.; *"Modern C++ Design: Applied Generic and Design Patterns (C++ in Depth)"*, Addison Wesley, 2001, ISBN 0-201-70431-5

Ambler, S. W.; *"Introduction to the Diagrams of UML 2.0"*, 2007 [Internet] http://www.agilemodeling.com/essays/umlDiagrams.htm (Accessed July 12th 2008)

Angelidis, A.; *"How do I write a template method of a template class?"*, 2005 [Internet] http://www.cs.otago.ac.nz/postgrads/alexis/tutorial/node44.html (Accessed July 15th, 2008)

Apple; *"Developing with Core Data"*, 2006 [Internet] http://developer.apple.com/macosx/coredata.html (Accessed August 26th, 2008)

Beck, K. & Cunningham, W.; *"Using Pattern Languages for Object-Oriented Program"*, 1987 [Internet] http://c2.com/doc/oopsla87.html (Accessed August 26th, 2008)

Bell, D.; *"UML basics: An introduction to the Unified Modeling Language"*, 2003*a* [Internet] http://www.ibm.com/developerworks/rational/library/769.html (Accessed July 12th 2008)

Bell, D.; (2003*b*), *"UML Basics: Part III: The class diagram"*, *The Rational Edge* [Internet] http://download.boulder.ibm.com/ibmdl/pub/software/dw/rationaledge/nov03/t_modelinguml_db.pdf (Accessed July 12th 2008)

Bell, D.; *"UML basics: The class diagram"*, 2004*a* [Internet] http://www.ibm.com/developerworks/rational/library/content/RationalEdge/sep04/bell/ (Accessed July 12th, 2008)

Bell, D.; *"UML's Sequence Diagram"*, 2004*b* [Internet] http://www.ibm.com/developerworks/rational/library/3101.html (Accessed July 12th 2008)

Bennett, S.; *"Object Oriented Systems Analysis/Design"*, Gardners Books, 2005, ISBN 0-077-11000-5

Boehm, D. B.; (1998), *"COCOMO II Model Definition Manual"*, The University of Southern California, chapter Using COCOMO II, p. 20 [Internet] ftp://ftp.usc.edu/pub/soft_engineering/COCOMOII/cocomo99.0/modelman.pdf (Accessed August 27th, 2008)

Booch, G.; *"Object-Oriented Analysis and Design with Applications"*, Addison-Wesley Professional, 1993, ISBN 0-805-35340-2

Booch, G., Jacobson, I. & Rumbaugh, J.; *"Unified Modeling Language User Guide (Object Technology S.)"*, Addison Wesley, 1998, ISBN 0-201-57168-4

Brooks, F. P.; *"The Mythical Man Month and Other Essays on Software Engineering"*, Addison Wesley, 1995, ISBN 0-201-83595-9

Cline, M.; *"C++ FAQ Lite - multiple and virtual inheritance - [25.9] Where in a hierarchy should I use virtual inheritance?"*, 2006 [Internet] http://www.parashift.com/c++-faq-lite/multiple-inheritance.html#faq-25.9 (Accessed July 29th, 2008)

Coplien, J. O.; (1995), *"Curiously Recurring Template Patterns"*, *C++ Report* **7**(2), 24–27 [Internet] http://portal.acm.org/citation.cfm?id=229229 (Accessed July 27th, 2008)

Cruz-Neira, C.; *"VR Juggler Portable Runtime (VPR) Home Page"*, 2008 [Internet] http://www.vrjuggler.org/ (Accessed July 13th, 2008)

Dawes, B., Abrahams, D. & Rivera, R.; *"Boost C++ Libraries Home Page"*, 2008 [Internet] http://www.boost.org/ (Accessed July 13th, 2008)

Dean Wampler, P.; *"The Role of Aspect-Oriented Programming in OMG's Model-Driven Architecture"*, 2003 [Internet] http://www.aspectprogramming.com/papers/AOP27th,2008 (Accessed

Deitel, H. & Deitel, P.; *"C++ How to Program (5th Edition) (How to Program)"*, Prentice Hall, 2005, ISBN 0-131-85757-6

Distler, T.; *"C++ Portable Runtime Evaluation"*, 2007 [Internet] http://tdistler.com/2007/11/01/c-portable-runtime-evaluation (Accessed July 13th, 2008)

Duerner, M., Maekitalo, T. & Indrayanto, A.; *"Platinum C++ Framework"*, 2007 [Internet] http://pt-framework.sourceforge.net/index.html (Accessed July 13th, 2008)

Duffy, D. J.; *“Financial Instrument Pricing Using C++ (The Wiley Finance Series)”*, John Wiley & Sons, 2004, ISBN 0-470-85509-6

Eckel, B.; *“Strong Typing vs. Strong Testing”*, 2003 [Internet] http://mindview.net/WebLog/log-0025 (Accessed July 16th, 2008)

Fowler, M.; *“Patterns of Enterprise Application Architecture”*, Addison-Wesley, 2002*a*, ISBN 978-0321127426 [Internet] http://martinfowler.com/books.html (Accessed August 26th, 2008)

Fowler, M.; *“Patterns of Enterprise Application Architecture: Active Record”*, 2002*b* [Internet] http://martinfowler.com/eaaCatalog/activeRecord.html (Accessed July 15th, 2008)

Fowler, M., Beck, K., Brant, J., Opdyke, W. & Roberts, D.; *“Refactoring: Improving the Design of Existing Code”*, Addison-Wesley Professional, 1999, ISBN 0-201-48567-2

Freeman, E., Freeman, E. & Bates, B.; *“Head First Design Patterns”*, O’Reilly, 2004, ISBN 0-596-00712-4

FSF; *“Autoconf Home Page”*, 2008 [Internet] http://www.gnu.org/software/autoconf/ (Accessed July 14th, 2008)

Gamma, E., Helm, R., Johnson, R. & Vissides, J.; *“Design patterns : elements of reusable object-oriented software”*, Addison Wesley, 1995, ISBN 0-201-63361-2

Gentile, S.; *“What’s New in Visual C++ .NET 2003”*, 2002 [Internet] http://www.ondotnet.com/pub/a/dotnet/2002/11/18/everettcpp.html (Accessed August 25th, 2008)

Glass, R. L.; *“Facts and Fallacies of Software Engineering”*, Addison Wesley, 2002, ISBN 0-321-11742-5

Graham, P.; *“Hackers and Painters: Essays on the Art of Programming”*, O’Reilly UK, 2004, ISBN 0-596-00662-4

Gunderloy, M.; *“Coder to Developer: Tools and Strategies for Delivering Your Software”*, Sybex International, 2004, ISBN 0-782-14327-X

Hansson, D.; *“Class ActiveRecord::Base Reference”*, 2008 [Internet] http://ar.rubyonrails.com/classes/ActiveRecord/Base.html (Accessed July 15th, 2008)

Hipp, D. R., Kennedy, D. & Harrelson, S.; *"About SQLite"*, 2008*a* [Internet] http://sqlite.org/about.html (Accessed July 13th, 2008)

Hipp, D. R., Kennedy, D. & Harrelson, S.; *"Appropriate Uses for SQLite - Application File Format"*, 2008*b* [Internet] http://www.sqlite.org/whentouse.html#appfileformat (Accessed July 13th, 2008)

Hutchings, B.; *"C++ Templates FAQ"*, 2006 [Internet] http://womble.decadentplace.org.uk/c++/template-faq.html (Accessed August 25th, 2008)

Informatics, A.; *"POCO C++ Libraries Home Page"*, 2008 [Internet] http://www.appinf.com/poco/info/index.html (Accessed July 13th, 2008)

Jacobson, I.; (2003), *"Use Cases and Aspects – Working Seamlessly Together"*, *Journal of Object. Technology* **2**(4), 7–28. [Internet] http://www.jot.fm/issues/issue_2003_07/column1.pdf (Accessed July 27th, 2008)

Jewell, D.; *"Giving some Juce to cross-platform tools"*, 2006 [Internet] http://www.theregister.co.uk/2006/12/18/juce_cross_platform/ (Accessed August 25th, 2008)

Johnson, E.; *"C++ - The Forgotten Trojan Horse"*, 2004 [Internet] http://ejohnson.blogs.com/software/2004/11/i_find_c_intere.html (Accessed July 15th, 2008)

Johnson, J. & Nielsen, J.; *"GUI Bloopers: Don'ts and Do's for Software Developers and Web Designers (Morgan Kaufmann Series in Interactive Technologies)"*, Morgan Kaufmann, 2000, ISBN 1-558-60582-7

Josuttis, N. M. & Vandevoorde, D.; *"Templates and Inheritance Interacting in C++: The Curiously Recurring Template Pattern (CRTP)"*, 2003 [Internet] http://www.informit.com/articles/article.aspx?p=31473&seqNum=3 (Accessed July 15th, 2008)

Kalev, D.; *"Target 32- and 64-bit Platforms Together with a Few Simple Datatype Changes"*, 2007 [Internet] http://www.devx.com/cplus/Article/27510/1954 (Accessed July 14th, 2008)

Kiczales, G.; *"Common Misconceptions"*, 2004 [Internet] http://www.ddj.com/showArticle.jhtml?articleID=184415113 (Accessed July 27th, 2008)

Kiczales, G. & Hilsdale, E.; *"Aspect-Oriented Programming with AspectJ"*, 2003 [Internet] http://www.ccs.neu.edu/research/demeter/course/w03/lectures/lecAspectJ-w03.ppt (Accessed July 27th, 2008)

Kiczales, G., Lamping, J., Mendhekar, A., Maeda, C., Lopes, C. V., Loingtier, J.-M. & Irwin, J.; (1997), *"Aspect-Oriented Programming"*, *European Conference on Object-Oriented Programming (ECOOP)* [Internet] http://www.parc.com/research/projects/aspectj/downloads/ECOOP1997-AOP.pdf (Accessed July 29th, 2008)

Kitware; *"CMake Cross Platform Make Home Page"*, 2008 [Internet] http://cmake.org/ (Accessed July 14th, 2008)

Kosmaczewski, A.; *"ActiveRecord and Unit Tests"*, 2008*a* [Internet] http://remproject.org/2008/05/26/activerecord-and-unit-tests/ (Accessed July 27th, 2008)

Kosmaczewski, A.; *"Rem Project: Subversion Log, revision 104"*, 2008*b* [Internet] http://code.google.com/p/remproject/source/detail?r=104 (Accessed July 16th, 2008)

Lewis, K.; *"Correct Endian Issues with Hex Constants Used as Byte Arrays"*, 2008 [Internet] http://softwarecommunity.intel.com/Wiki/DevelopforCoreprocessor/300.htm (Accessed July 14th, 2008)

Luckey, R.; *"Codebase Cost"*, 2006 [Internet] http://www.ohloh.net/wiki/project_codebase_cost (Accessed August 27th, 2008)

Melikyan, H.; *"Portable Types (PTypes) Home Page"*, 2008 [Internet] http://www.melikyan.com/ptypes/ (Accessed July 13th, 2008)

Meyers, S.; *"Effective C++ : 55 Specific Ways to Improve Your Programs and Designs"*, Addison-Wesley Professional, 2005, ISBN 0-321-33487-6

Ng, P.-W. & Jacobson, I.; *"Aspect-Oriented Software Development with Use Cases"*, Addison Wesley, 2005, ISBN 0-321-26888-1

Obiltschnig, G.; *"http://www.appinf.com/download/FPIssues.pdf"*, 2006 [Internet] http://www.appinf.com/download/FPIssues.pdf (Accessed July 14th, 2008)

O'Regan, G.; *"Introduction to Aspect-Oriented Programming"*, 2004 [Internet] http://www.onjava.com/pub/a/onjava/2004/01/14/aop.html (Accessed July 27th, 2008)

Perkins, J.; *"premake Home Page"*, 2008 [Internet] http://premake.sourceforge.net/ (Accessed July 14th, 2008)

Raymond, E.; *"The Cathedral and the Bazaar"*, OReilly, 2001, ISBN 0-596-00108-8

Rice, D.; *"Geekonomics: The Real Cost of Insecure Software"*, Addison Wesley, 2007, ISBN 0-321-47789-8

Richardson, J. & Gwaltney, W.; *"Ship it! A Practical Guide to Successful Software Projects"*, Pragmatic Bookshelf, 2005, ISBN 0-974-51404-7

Rosvall, S.; *"Coding Standard Generator"*, 2005 [Internet] SvenRosvall (Accessed August 27th, 2008)

Schaerli, N., Ducasse, S., Nierstrasz, O. & Black, A.; *"Traits: Composable Units of Behavior"*, 2003 , Technical report, Software Composition Group, University of Bern, Switzerland [Internet] http://web.cecs.pdx.edu/~black/publications/TR_CSE_02-012.pdf (Accessed July 29th, 2008)

Schmidt, D. C.; *"Adaptive Communication Environment (ACE) Home Page"*, 2007 [Internet] http://www.cs.wustl.edu/%7Eschmidt/ACE.html (Accessed July 13th, 2008)

Spolsky, J.; *"Daily Builds Are Your Friend"*, 2001 [Internet] http://www.joelonsoftware.com/articles/fog0000000023.html (Accessed July 16th, 2008)

Stein, D., Hanenberg, S. & Unland, R.; (2002), *"A UML-based Aspect-Oriented Design Notation For AspectJ"*, *in* "AOSD '02: Proceedings of the 1st international conference on Aspect-oriented software development", ACM, New York, NY, USA, pp. 106–112

Stephens, D. R., Diggins, C., Turkanis, J. & Cogswell, J.; *"C++ Cookbook (Cookbooks (O'Reilly))"*, O'Reilly Media, Inc., 2005, ISBN 0-596-00761-2

Stokes, J.; *"Ars Technica: RISC vs CISC in the Post RISC Era."*, 1999 [Internet] http://arstechnica.com/cpu/4q99/risc-cisc/rvc-1.html (Accessed July 14th, 2008)

Stroustrup, B.; (1987), *"Multiple Inheritance for C++"*, *in* "Proceedings of the Spring 1987 European Unix Users Group Conference", Helsinki [Internet] http://citeseer.ist.psu.edu/cache/papers/cs/15957/http:zSzzSzwww.cs.colorado.eduzSz~diwanzSz5535-00zSzmi.pdf/stroustrup99multiple.pdf (Accessed July 29th, 2008)

Stroustrup, B.; *"C++ Programming Styles and Libraries"*, 2002 [Internet] http://www.research.att.com/~bs/style_and_libraries.pdf (Accessed July 13th, 2008)

Stuart, J. A., Dascalu, S. M. & Jr., F. C. H.; *"Towards A Unified Approach for Cross-Platform Software Development"*, 2005 [Internet] http://www.cse.unr.edu/~dascalus/IASSE2005_JS.pdf (Accessed July 14th, 2008)

Tidwell, J.; *"Designing Interfaces : Patterns for Effective Interaction Design"*, O'Reilly Media, Inc., 2005, ISBN 0-596-00803-1

Tiobe; *"TIOBE Programming Community Index for July 2008"*, 2008 [Internet] http://www.tiobe.com/index.php/content/paperinfo/tpci/index.html (Accessed July 13th, 2008)

Warmer, J., Bast, W., Pinkley, D., Herrera, M. & Kleppe, A.; *"MDA Explained: The Model Driven Architecture: Practice and Promise"*, Addison Wesley, 2003, ISBN 0-321-19442-X

Weber, S.; *"The Success of Open Source"*, Harvard University Press, 2004, ISBN 0-674-01292-5

Weirich, J.; *"Rake Home Page"*, 2006 [Internet] http://rake.rubyforge.org/ (Accessed July 14th, 2008)

Index