

Umbrello UML Modeller

**A Diagramming Programme for the Unified
Modelling Language Developed Using Bazaar
Methods**

BSc Honuors Dissertation, Final Report

**University of Stirling, Department of
Computing Science and Mathematics**

April 2003

Jonathan Riddell

Umbrello UML Modeller: A Diagramming Programme for the Unified Modelling Language Developed Using Bazaar Methods; BSc Honours Dissertation, Final Report; University of Stirling, Department of Computing Science and Mathematics; April 2003

by Jonathan Riddell

Copyright © 2002, 2003 by Jonathan Riddell

I am the lead developer and project manager for Umbrello UML Modeller, a computer programme for drawing diagrams of software. I am developing it using the bazaar method, a collaborative and open development style with contributions welcome from anyone. I took control of the programme from the original author in summer 2002.

As project manager I have encouraged and organised feedback and contributions to the programme and managed a growing group of regular contributors. I have successfully incorporated Umbrello into the larger KDE (K Desktop Environment) project, working with a number of KDE developers to ensure a smooth transition. Successful project management requires good handling of programming resources such as CVS and the bug tracker and communication resources such as the mailing lists and the website.

When I took over Umbrello many features had been added which had not been sufficiently debugged. There had been no stable release in almost a year and the programme crashed frequently. My initial development work focused on bringing the stability of the programme to a state where a new release could be made, this involved a great deal of debugging. Since then I have added new features and diagram types, while refactoring the internal code to make it much more easily maintainable in future.

Through my work on Umbrello I have created a thriving programme with an increasing number of real world users which will help to bring effective software engineering and modelling to the development of Free Software.

Permission is granted to copy, distribute and/or modify this document under the terms of the GNU Free Documentation Licence, Version 1.2 or any later version published by the Free Software Foundation; with no Invariant Sections, no Front-Cover Texts, and no Back-Cover Texts.

Acknowledgements

I would like to thank the following people for their help in making this project possible:

- Robert Clark for acting as project supervisor.
- Paul Hensgen, original author of the programme that is now Umbrello, for creating an excellent piece of software.
- Mike Quin for providing and supporting the student Unix Debian box.
- Perdita Stevens of Edinburgh University for advice on XMI.
- The other developers and contributors to Umbrello who have made the programme what it is today. Some of them are listed at <http://uml.sf.net/developers.php>.
- Finally the many KDE developers who have created the finest computer desktop available, and especially those who have helped and advised with Umbrello.

Table of Contents

1. Introduction.....	1
Background and Context.....	1
UML	1
Bazaar Development Method	2
Umbrello UML Modeller	3
Screenshots of Umbrello.....	4
Technologies Used.....	6
Scope and Objectives	6
Achievements	6
Overview of Dissertation	7
2. State of the Art	8
Argo UML.....	8
Ideogramic	9
Rational Rose	10
Together.....	11
General Diagram Tools: Dia, Kivio, Visio.....	12
Umbrello	13
3. Developing Umbrello UML Modeller	14
Rescuing the Programme	14
Porting to KDE 3.....	14
Tidying the Code.....	15
UML Diagram.....	16
Debugging	17
Features	18
Canvas Zoom and Resize.....	18
Undo and Redo	19
Parameterised Classes.....	19
Interfaces	19
Component Diagrams and Deployment Diagrams	19
Cut and Delete Selected.....	19
User Interface	20
Refactoring Classes	20
UMLAssociation	21
UMLObject.....	21
Data Classes.....	24
Use of Multiple Inheritance	25
Use of XMI	25
Compatibility with Solaris	26
4. Project Tools.....	27
Hosting on Sourceforge.net.....	27
CVS Revision Control.....	27
Bug and Feature Trackers	28
Website.....	28
Releases.....	28

Translations	29
Mailing Lists	29
IRC	30
Class Documentation	30
Handbook	30
Text Editor.....	30
5. Using The Bazaar Development Process	32
Developer Motivation.....	32
Umbrello Leadership.....	32
Programme Forks	33
Release Plan.....	33
Encouraging Participation	33
Publicity.....	34
Inclusion in KDE.....	34
Competition to Umbrello.....	35
Extendible API	35
Free Software.....	36
6. Conclusion	37
The Bazaar Method	37
Future Work	38
Colophon.....	39
References.....	40
Bibliography	42

Chapter 1. Introduction

Umbrello UML Modeller is a UML diagramming programme which I am developing using an open bazaar development method. It is written in C++ and uses the libraries available from KDE, the K Desktop Environment for Unix. My project consists of both developing the programme and managing the other developers and users.

Background and Context

UML

UML is The Unified Modelling Language[uml-specification], a specification for creating diagrams to model software systems. It was created as an open standard and based on best practice and experience from previous software modelling languages. The original authors of UML - Booch, Rumbaugh and Jacobson - had each previously independently devised their own software modelling techniques called OOSE, OMT and Booch. By 1997 they had created a unified language which became UML. UML is now standardised by the Object Management Group who develop a number of technology standards in the area of distributed computer technology, notably CORBA.

Modelling is instrumental in developing software of any significant complexity. It allows a scale of abstraction from the problem which is significantly higher than any programming language. It is used in the stages of development that bridge the requirements and the implementation. Using UML as a standard language ensures that all developers working on the system can understand the diagrams.

In theory UML is independent from the implementing language, although certain aspects of it can be considered to be more similar to Java and C++ than other languages. It is entirely based around the object orientated paradigm and unsuitable for other language styles.

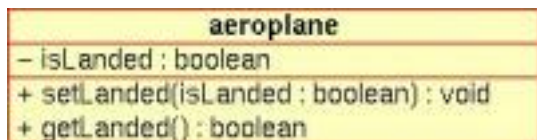
UML defines several "views", i.e. diagram types, onto a single model of the system and a model can have several different diagrams of the same type showing different parts of the system, different layers of abstraction of the system or areas of the system at various times during its lifecycle.

There are 9 diagram types in UML. They fall roughly into what is called the 4+1 categories of logical, process, development, physical (deployment) and scenario. Scenario is the +1 view, the use case diagrams created for this view are said to be laid over the top of the others because the use case view motivates the architectures in the other views while not being directly related to the implementation.

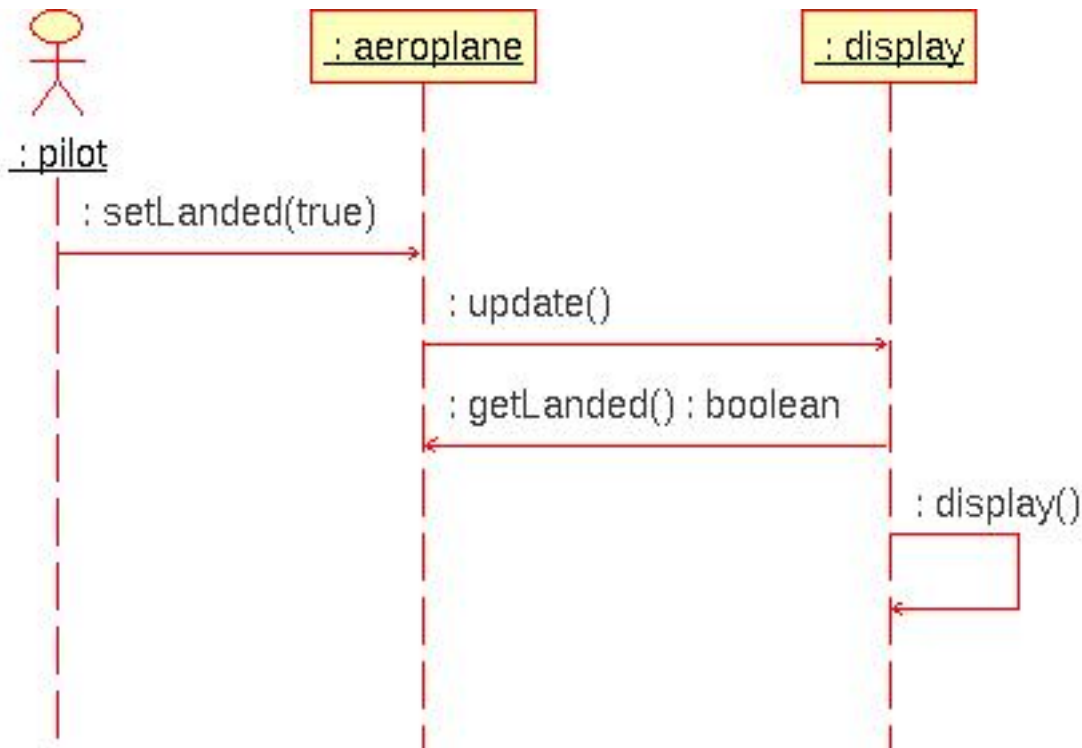
As an example of how UML is used, here are some diagrams from a previous project of mine:



A Use Case Diagram - the pilot is an outside user and is associated with a scenario which will occur as part of the program



A Class Diagram - a single class used in the program



A Sequence Diagram - models the method calls

These diagrams indicate how UML is flexible enough to be able to show different abstractions of the final programme. The use case diagram has no direct relationship to the final code but instead gives an idea of the interactions involved. The class diagram gives an outline of the classes involved while the sequence diagram goes right down to the level of showing which methods are called. In theory a program could be created entirely from UML without ever touching code but in practice implementation details mean this is not possible.

UML itself is defined in a higher level modelling language called The Meta Object Facility[mof]. MOF diagrams look a lot like UML although there are some inconsistencies between the two. MOF is also used for other modelling languages such as The Common Warehouse Metamodel.

There is an XML language called XML Metadata Interchange[xmi]. XMI is a container specification which can be used for arbitrary metadata models. Specifications of XMI exist for MOF and UML.

Bazaar Development Method

Umbrello is Free Software. This means it can be freely distributed (and sold) or modified by anyone who has a copy of it. The bazaar development method, which was first formally introduced in *The Cathedral and the Bazaar*[cathedral-bazaar], capitalises on these freedoms to allow controlled contributions from

any interested developer. Developers are motivated by the desire for a programme which they need and which is of a quality they find acceptable to use. Because developer motivation is for the end result (rather than because they're paid to do it) the final software is often of a higher quality than proprietary software. Software developed in this method is typically more secure and stable compared to proprietary software because of the larger number of developers - "Given enough eyeballs, all bugs are shallow"[cathedral-bazaar]. With enough people looking at the code, problems will be quickly recognised and fixed. It is also one of the few development methods to overcome the limitation cited in *The Mythical Man Month*[mythical-man-month] that adding more developers to a project will only make it later; because the developers work in their own time their learning of the system does not significantly impact on the work of more advanced developers. New developers can help each other and work at a pace with which they feel comfortable.

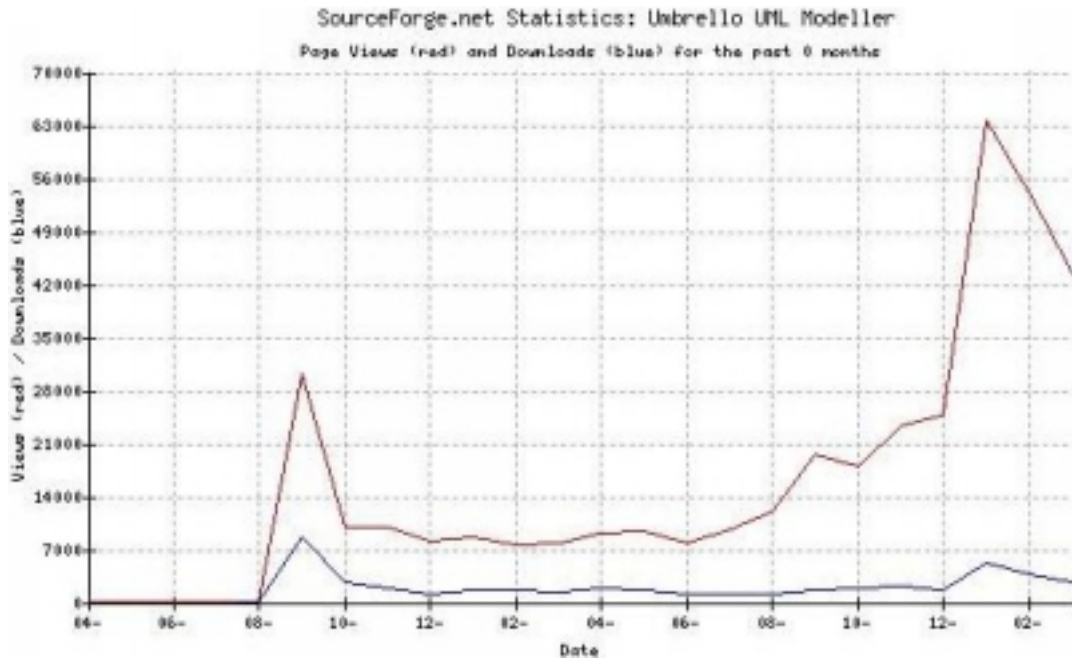
One of the first and most successful projects to use this method was Linux, a Unix-style operating system kernel, and there have been many other successes since then.

Umbrello UML Modeller

UML Modeller (as it was originally called) was written by a university student as part of a final year project. I joined its development in early 2001 at the time of its first public release when I found it lacked some features I required and created binary packages for it. Over the next year development on the programme was inconsistent: a number of features were added but unpolished and the file format was changed but development was not actively encouraged. The author would sometimes not be heard from for several months and there were no new releases. The original author left in the summer of 2002 and gave me full permissions over the project in September.

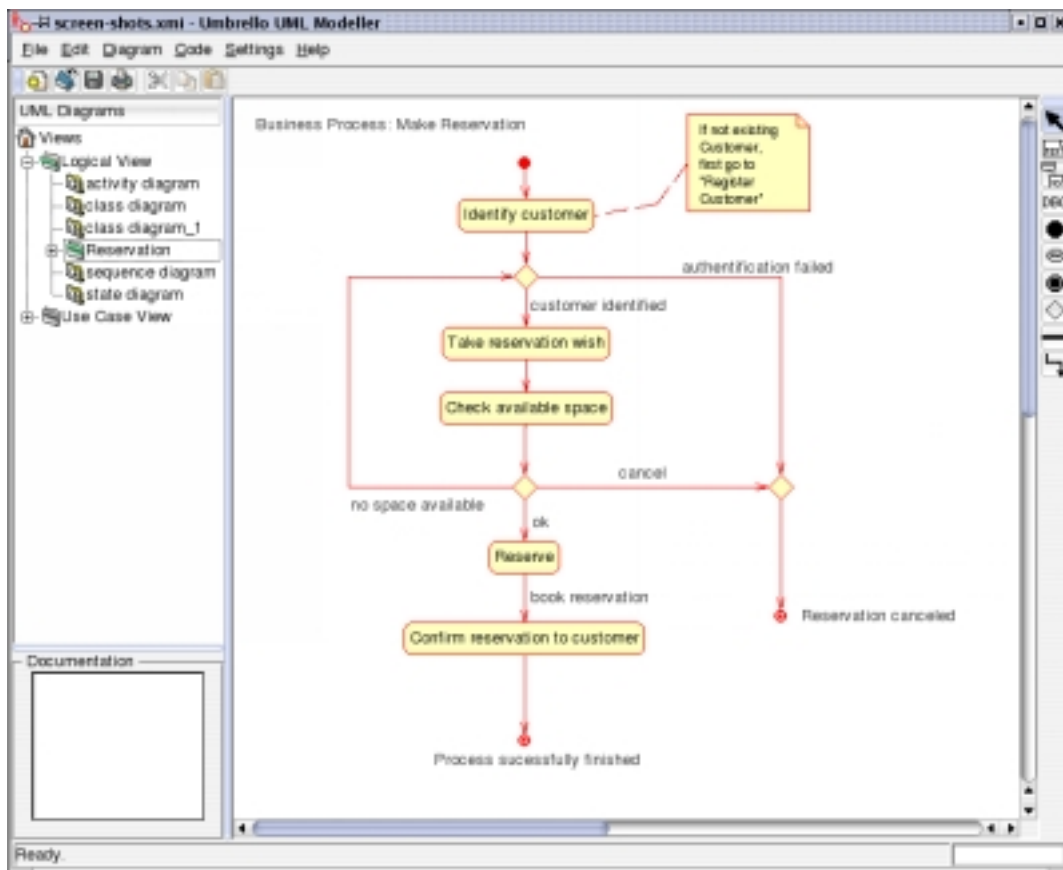
Umbrello is first and foremost a UML diagramming tool for developers. It is not a toy or a research vehicle. It is the most advanced Free Software tool in its class and in terms of features is beginning to compete with its commercial competition. It consists of over 65,000 lines of code including comments and is made up of almost 300 files/classes. It is currently ranked in the top 10% of Sourceforge.net projects for activity, even though most of the development now occurs outside of Sourceforge.net.

It is hard to judge the number of users but the latest version has had over 10,000 individual downloads and its inclusion into KDE will increase the number of users considerably. The number of active developers is almost as hard to tell but numbers at least half a dozen. The developers mailing list has about 450 postings a month.

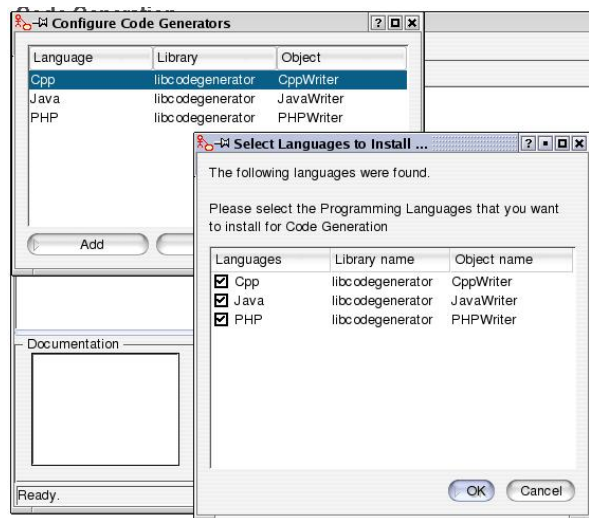


The two year statistics graph shows how interest in Umbrello has grown since it started in August 2001 and since I took over in September 2002. Spikes in September 2001 and January 2003 indicate stable releases.

Screenshots of Umbrello



Umbrello being used to create an activity diagram. The user interface consists of the current diagram, a tree view of the model elements and diagrams and a documentation box which displays the text documentation associated with the currently selected item.



Umbrello has a modular and dynamically extendable code generation feature.

Technologies Used

Umbrello uses a number of technologies and resources. It is written in C++ for Unix-style operating systems. The libraries used are almost all provided by KDE[kde] which itself is build on a cross platform widget set called Qt made by Trolltech[trolltech].

The programme is hosted at Sourceforge.net[sourceforge.net], a site which provides web space, mailing lists, bug trackers and other resources to Free Software projects. When Umbrello became part of KDE some of the resources used were moved over to those used by KDE.

Scope and Objectives

The objectives of my project are to create the best UML modelling programme available natively for Unix and to nurture a group of developers and contributing users who will ensure its continued development.

It is hoped that by releasing Umbrello as part of KDE there will be a greater awareness and use of UML by developers of KDE and other Unix applications. While open development of Free Software has had many successes it is also the case that many projects fail for a number of reasons including lack of programme architecture planning. By increasing the profile of UML, other developers will create visual models to aid in their own understanding of their problem and help new developers become familiar with their code bases.

As I am not the original author of Umbrello, and because it is being worked on by other developers (under my guidance), it is important to differentiate between my work and that of others. This is further complicated because all patches submitted to the project and reviewed are usually changed by myself. However the majority of the work done on the programme over the last year has been mine.

Achievements

I released Umbrello 1.1 on January 20th 2003, after several months of removing bugs and re-invigorating interest in the project. I integrated Umbrello into KDE shortly after and have added many new features since then.

New features include:

- Paramaterised classes
- An undo/redo function
- Re-write of the clipboard to use XMI, it now works with all widgets and diagram types
- Added component and deployment diagrams and associated widgets and model objects to complete the UML diagram family
- Refactored the association code to make associations part of the internal model rather than superficial widgets. This makes code generation much simpler.
- Refactored the code for internal objects to remove duplicated code while also bringing classes which had not been updated up to date.

Through promoting the programme and taking time to have conversations with KDE programmers I have encouraged several new developers to work on Umbrello. This should shortly bring new features such as a new code import API.

Overview of Dissertation

State of the Art introduces the current leading UML modelling programmes and explains their significance to this project. *Developing Umbrello UML Modeller* explains the technical details of my work including the features I have added and refactoring performed. The *Project Tools* chapter describes the technologies used for successful bazaar development. *Using The Bazaar Development Process* shows how I have leveraged the communications power of the Internet to create an active group of developers from around the world all working on Umbrello.

Chapter 2. State of the Art

Umbrello has a unique combination of features:

- It is Free Software
- Written in C++ so runs as a native program (not, for example, Java or Python)
- A clean and friendly user interface
- Written for one of the major desktop environments giving users a familiar interface

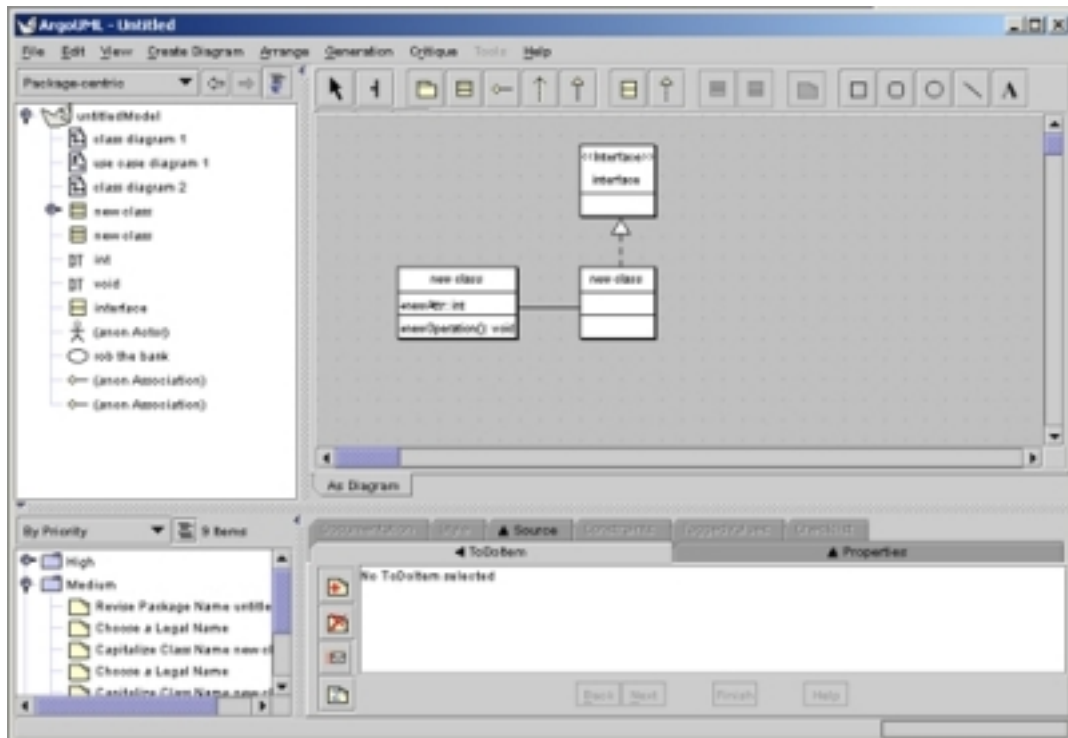
However there are several other significant UML and general diagramming programs available.

Argo UML

Like Umbrello, Argo[argo] started as an academic project and, like Umbrello, it is openly developed Free Software. It lists cognitive support as one of its differentiating features. In practice this means several innovative features inspired by theories on human cognition during design tasks:

- Critics - model design is continuously reviewed by plugin "critics", for example if a class inherited from two other classes a Java critic would display a warning that this is incompatible with Java.
- To Do List - Critics, UML diagram widgets and users can add items to categories within a To Do list which is continuously displayed. This mirrors the lists commonly used by software engineers when performing common tasks.
- Non-modal Wizards - wizards to guide developers through common tasks, which can be paused, stopped and resumed at any time to keep the user in control.
- Table view - diagrams can also be viewed as tables showing the relationships between diagram elements.

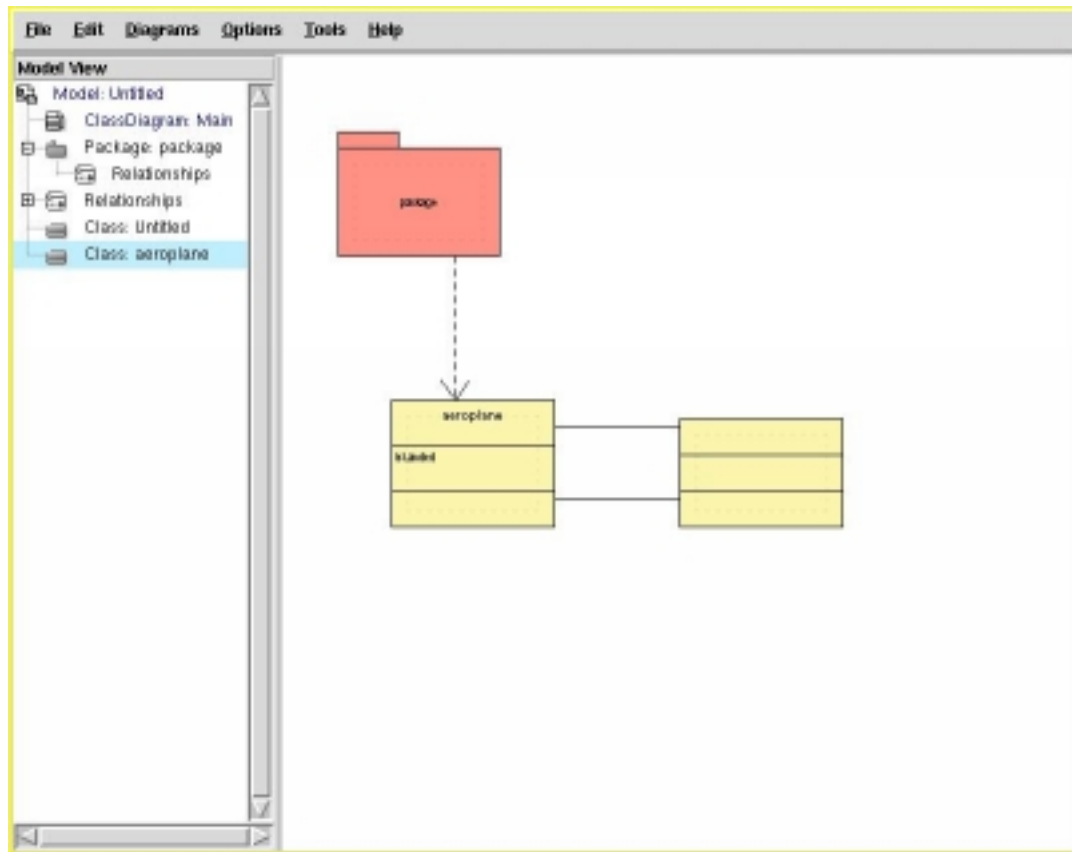
While Argo has many original features such as those above it has a poor user interface. Being written in Java it is slow, unresponsive and is inconsistent with the surrounding desktop. It is also not considered to be stable software, having not released a 1.0 version.



Argo UML

Ideogramic

Ideogramic produce a UML programme for GNU/Linux and Windows. It is not the most feature-filled of programmes but has one novel idea through its support for electronic whiteboards - widgets are created by drawing their outline on the canvas, associations similarly. This makes the user interface unintuitive without practice and even then its limitations are all too obvious. Unfortunately the whiteboard-friendly user interface is another example of a software patent which does not live up to the required non-obvious criteria.

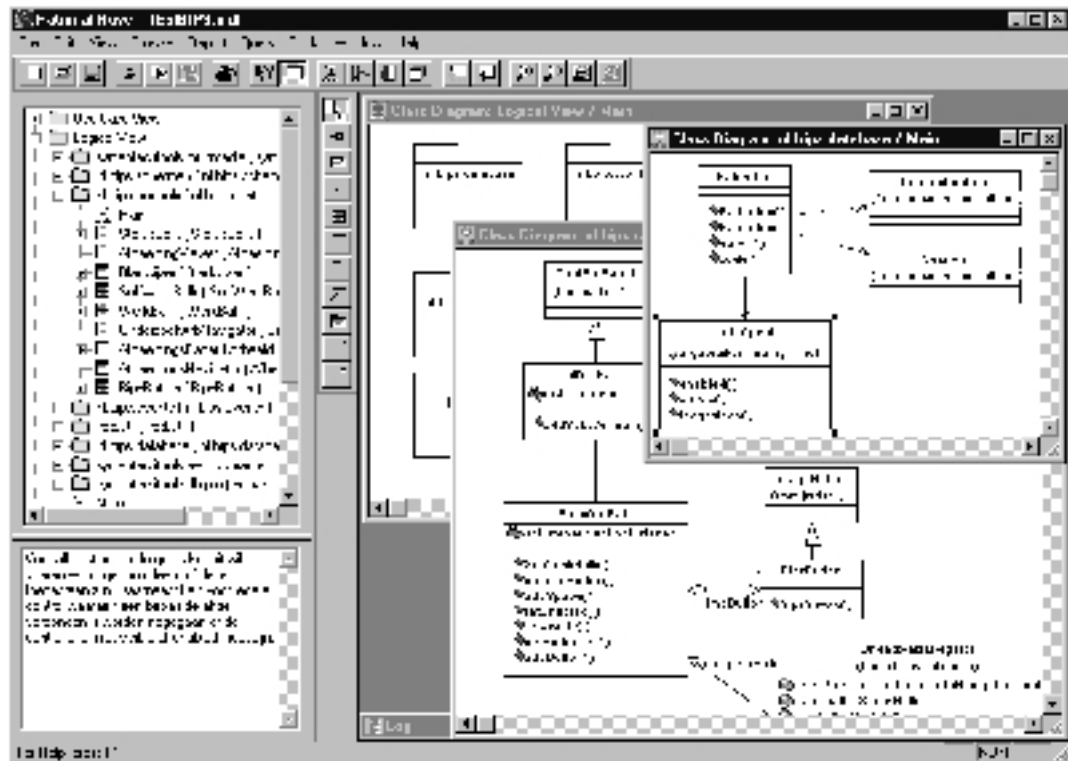


Ideogramic UML

Rational Rose

Rational Rose[rational] is often considered the definitive UML tool because the founders of the Rational company also include the creators of UML. Early versions of Umbrello described it as an equivalent of Rational Rose and screenshots indicate that it has a similar user interface layout.

Unfortunately I have been unable to find a working copy of Rational Rose and have not been able to evaluate the software.



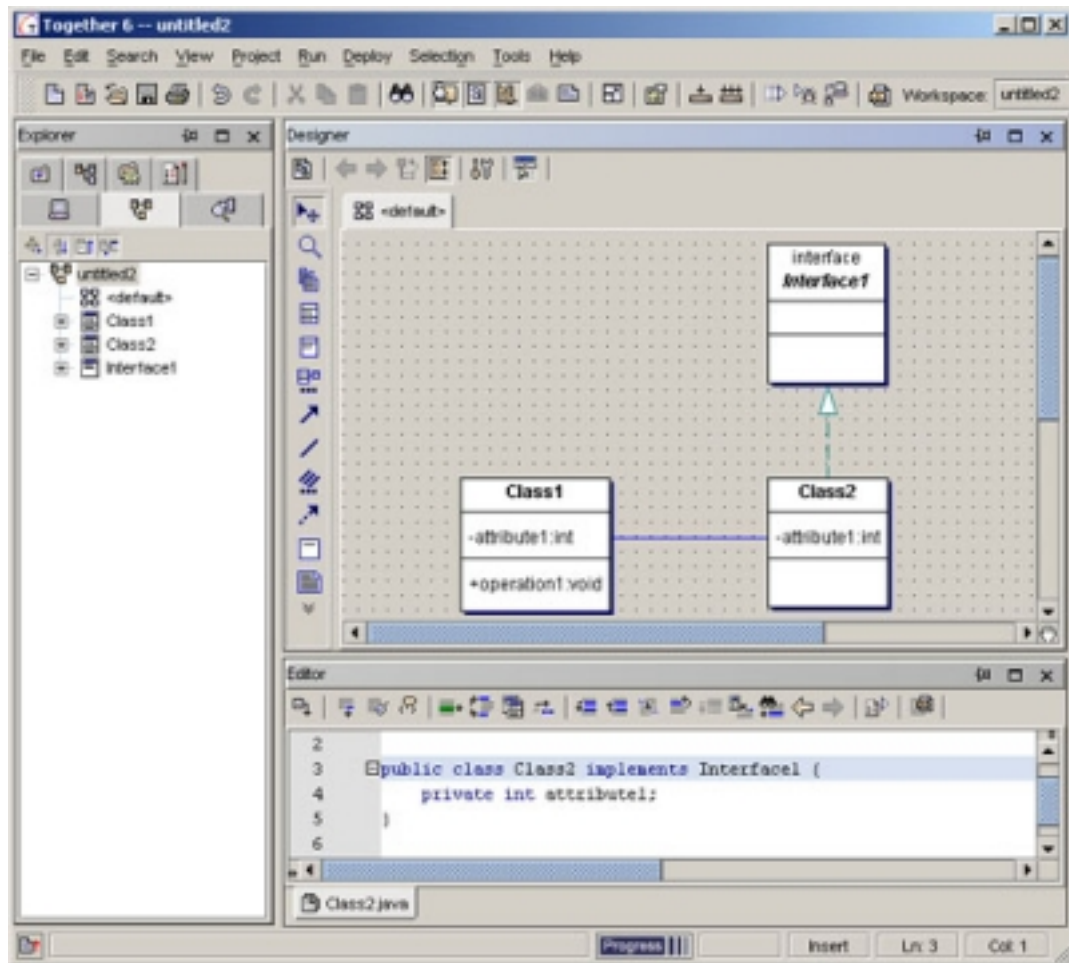
The layout of Umbrello's user interface is historically based on Rational Rose

Together

Together[together] considers itself to be a "model, build and deploy" platform because of its code synchronisation feature. It includes a text editor for program code as well as a UML diagram ability. Changes to the diagrams are in theory also made to changes in the code and changes to the code are reflected immediately in the diagrams. It also includes support for various specific Java language features such as Java Bean components.

The code and model synchronisation is an impressive feature. It reportedly has had to be re-written four times to reach its current quality. Unfortunately the feature is patented[software-patents], even though it is arguable that the idea truly is non-obvious. Like Argo, Together is written in Java and suffers from the same issues of unresponsiveness and interface inconsistency.

TogetherSoft has recently been bought by Borland, it will be interesting to see whether and how it will be integrated into Borland's existing tools.



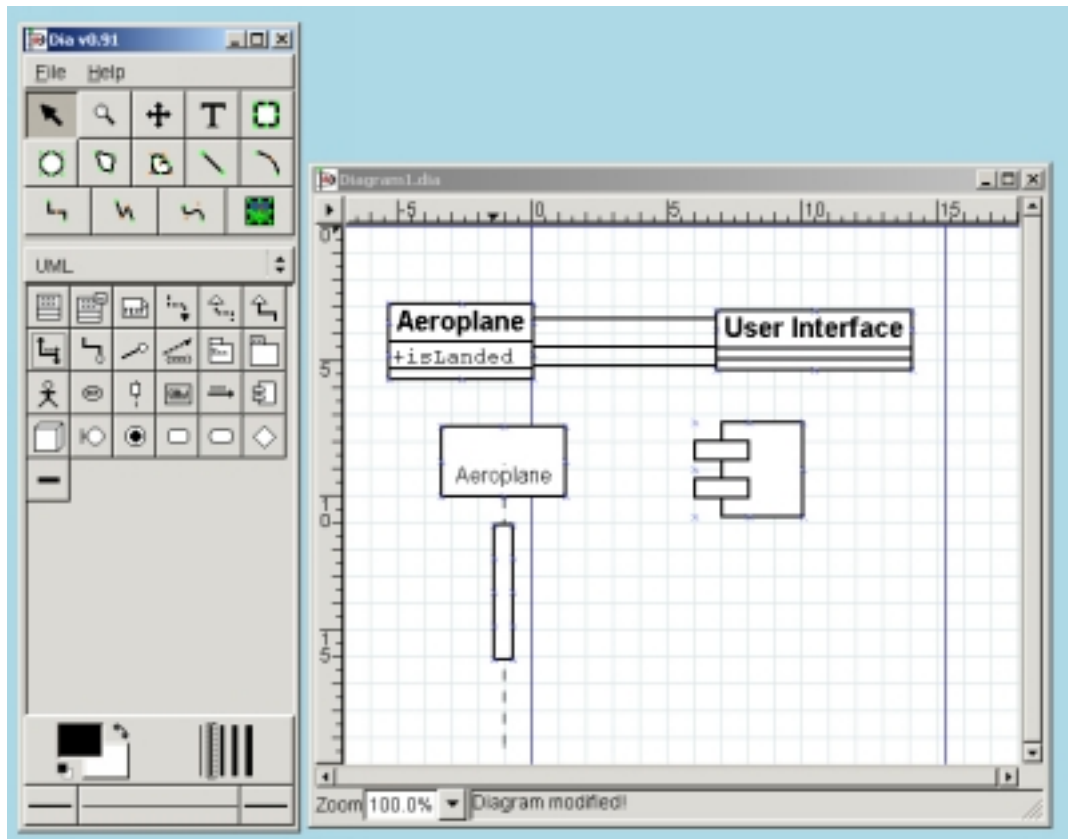
Together

General Diagram Tools: Dia, Kivio, Visio

There are a number of general diagramming programmes which include stencils for numerous types of diagram from computer networks to office layout. They typically also include stencils for UML. Programmes in this category include:

- Dia[[dia](#)] - a mature Free Software tool for Unix written for the GNOME desktop environment.
- Kivio[[kivio](#)] - part of the KOffice suite, a Free Software tool with proprietary stencils. It is buggy and unreliable to use.
- Visio[[visio](#)] - a proprietary tool for Windows by Microsoft.

The disadvantage of using these tools rather than a UML specific tool is the lack of enforcement of UML rules so sequence diagram objects can appear next to use cases for example. There is also no internal model so classes cannot appear in more than one diagram (or at least changes to a class in one diagram will not change its representation in any other diagram).



Dia, while an excellent general diagramming tool, lacks UML specific features such as enforcing a diagram type, code export/import and XMI.

Umbrello

Umbrello's development depends very little on that of its competitors. The programme is not yet at a stage where it contains many unique features, rather the concentration has been on getting a powerful UML modelling programme while retaining its friendly user interface. The possibility of being inspired but then restricted by software patents is another reason why I have tended to pay little attention to other UML programmes. Umbrello is well positioned as a unique programme which serves a market that was previously unfulfilled.

Chapter 3. Developing Umbrello UML Modeller

Umbrello UML Modeller is written in C++ for Unix-style operating systems. C++ is the most widely used programming language for desktop applications, its object orientated basis making it suitable for graphical user interface programming. It compiles to native code making it much more efficient than alternatives such as Java. Unfortunately its heritage of being backwards compatible with C means it is a complex language with many features and oddities.

Unix style operating systems have traditionally been excellent for servers and technically competent users but the high prices and expensive hardware have prevented it being used on the desktop. However the rise of the Free Software movement and the freely available Unixes, GNU/Linux and the BSD variants, which work on cheap PC hardware have allowed it to be used by non-enterprise users. As the demand for a user-friendly desktop and applications increased several projects sprung up to create them. The original and most successful is KDE (the K Desktop Environment, a play on words of CDE, K originally stood for Kool but that was quickly and quietly dropped). KDE provides a large set of libraries, an advanced component system called KParts, inter-process communication technology called DCOP, a large number of applications and all the integration that would be expected of a desktop environment including drag-and-drop, unicode clipboard, configuration tools, a taskbar with application menus and a desktop background.

KDE itself is built upon Qt[qt], a commercial cross-platform library providing primarily GUI widget functions but also classes for files, sockets, data structures, XML parsing and much more. Qt extends the C++ language to add an inter-object communication mechanism called signals and slots. An object's slots are methods which are called by emitting a connected signal from an object. Slots and signals can be arbitrarily connected at run time. It allows for independence between classes which is useful in patterns such as Model View Controller.

Like most Free Unix applications, KDE programmes use GNU Autoconf and Automake to simplify the compile process and to ensure it works on as many platforms as possible. Autoconf and automake are complex programmes with many configuration details, however the power of KDE is that most of the hard work has been done for you. A script called `kapptemplate[kapptemplate]` generates the autoconf and automake files given some simple parameters. KDE also makes use of GNU Libtool which takes care of the differences between compilers and linkers for shared object files (dynamically loaded libraries). Libtool uses plain text files to keep track of the dependencies required by each library.

Rescuing the Programme

The original author abruptly stopped his work on Umbrello earlier than had been expected when his computer spontaneously exploded in the middle of a commit of the code. I received a tearful e-mail from an internet cafe about the incident. This left the code base in a state where it did not compile. The commit had been a large one which contained many different features. I could have reverted the code to the version from the day before but this would have meant the loss of all the features which were to be added. I spent two days going through the code working out which features had been completely committed and which had been only partially committed so had to be removed by hand or reverted back to a previous revision.

This surprise start to the project got me familiar with the code base and taught me that incremental commits of features, rather than committing several days work at one time, is very good practice.

Porting to KDE 3

My first task in development was porting the current code base to KDE 3 which had been recently released. While the step from KDE 1 to KDE 2 was a complete re-write of the environment, KDE 3 was simply a chance to break binary compatibility with KDE 2. There were relatively few differences to the APIs and these were sufficiently well documented that it was a straightforward job to track down the required changes. Umbrello was originally written for KDE 2. The major part of converting to KDE 3 was updating the autoconf and automake configuration files to make them compatible with the latest KDE version.

I immediately created and announced packages for the KDE 3 version of Umbrello, which was the first sign of activity from the project in some time and effectively restarted the project.

Tidying the Code

The code itself was in a messy state when I took over, the indentation and naming schemes were inconsistent, having been worked on by several different people. I used a programme called `astyle` [`astyle`] to create a constant indentation and brackets policy. I chose to use tabs for all indentation, which is equivalent to 8 spaces. This large indentation makes it instantly clear where blocks begin and end. It also shows up areas with too many nested blocks, an indication of poor design.

Some attempt had been made in the code to use Hungarian notation naming. This is a syntax where variables and classes are prefixed with an indication of their type. So `m_bIsInstance` is a boolean member variable and `CAssociationWidget` is a class. There is a balance to be made between informative use of Hungarian notation and its distractive value. I removed the occasional uses of `C` to prefix a class name; it is obvious to the programmer what is a class especially since class names always begin with an upper case letter. I kept the Hungarian notation for member variables, which are declared far away in the code from where they are used, but removed it for method variables where the declaration should be no more than a few lines above its use.

Several enums are used in the code to keep track of widget and object types, and these were stored in a class called simply `Uml`. Namespaces are a relatively new feature to C++ and traditionally are not supported by all compilers, however I found no problems and received no complaints on replacing the class with a namespace. The lack of use of namespaces in C++ seems to come from a reliance on the kludged solutions used prior to their introduction than on any real lack of support for them.

There are bad practices throughout the code which I did not change because it would have taken too much time. Many methods use variables with single character names, or un-descriptive names like `temp`. The `->` dereference operator is often surrounded by spaces, despite being a very tightly coupled operator. C++ allows for method code to be included in header files rather than in the normal code files. This is supposed to help with optimisation but I find the feature frustrating because it makes methods hard to find and requires a complete recompilation of the programme when there are any changes to the methods. I removed some of the worst offending inline methods, those which contain more than trivial code.

I never did create a formal style document dictating guidelines for the layout of code. People's preferences vary and it is better to let them code in the style that they find easiest and most efficient to working. Almost all submissions though have followed my preferred style since the advantages of a constant code layout are obvious to anyone who has worked on code with other developers. However I have received complaints that the problems listed above and the lack of a formal coding style make the code base harder to get to grips with. Given that, and the amount of time I spend during coding sessions

reformatting code to remove the problems listed above and make it more readable, I wish I had spent more time initially on this apparently superficial problem.

Flex is a computer language lexical analyser which is used by Umbrello to create the code importer. I was receiving a number of compilation problems from users who, it turned out, did not have Flex installed. By adding a further check in the autoconf files machines without Flex will now give out a helpful error message.

I received a patch for cleaning up the #include header file lines: removing headers which had been required but were no longer needed by the file and adding ones which were required but brought in by other includes (something which should not be relied upon). Unfortunately this patch removed a number of headers which were required for the files. Although it worked fine for the submitter it would no longer compile on my setup which shows the importance of testing configuration changes on a number of different systems. The only parts of this patch which I chose to accept were the replacements of a number of C includes replaced with C++ equivalents.

The compiler outputs warnings for several potential problems such as unused variables and implicit casting. In the case of implicit casting the required fix is usually just to add an explicit cast. Unused variables are a common occurrence with overridden methods, it is possible to just comment out the variable name in the method header to prevent a compiler warning. While most compiler warnings are unlikely ever to be issues which could cause a problem, it is good practice to remove any that occur so I spent some time going through the code to remove all the warnings.

UML Diagram

Surprisingly for a UML tool there were no UML diagrams of the code behind Umbrello. I imported the classes from the source into a model and created the class diagram by manually checking each file for inherited and associated classes. I included only the core classes which provide the main functionality of storing and drawing class diagrams. Sections of the program which are clearly separated, such as code import, export and the clipboard were not included. This makes the diagram easier to read, and at only a few classes each these sections can be understood without the need for a diagram. The class diagram[umbrello-class-diagram] has been extremely useful in understanding the structure of the code and when working with the code to know which classes are aware of which other classes' attributes and operations.

Creating the diagram also gave me a chance to actually use Umbrello for a real world use. This highlighted some user interface issues and I discovered some small technical bugs as well.

Debugging

Most of my initial work on Umbrello was debugging the features which had been added since the last stable release but not yet properly tested. Bugs can involve any area of the code and each one requires detective work to find the relevant code, understand it and work out why the error occurs.

One vital tool for debugging is the GNU Debugger, gdb. It allows backtraces which show the exact line where an error occurred and how that method was reached. It can also be used to step through a programme line by line displaying the values of any variables, however I usually find it just as easy to send lines to standard output showing only the variable in which I am interested.

While debugging it is very tempting to refactor the code to remove any duplications. Doing this is a balance between removing excess code and ensuring that you don't create another bug by unknowingly changing the behaviour.

A large number of bugs are crashes caused by using uninitialised variables. Gdb usually tracks these down quite easily but it can then be hard to work out why the variable was not initialised. It typically turns out to be a route through the code which was not anticipated when it was first written. To prevent this it is important that any class initialises its member variables as soon as possible after construction. After fixing this type of bug I usually create or bring up to date an *init()* method which initialises all the class variables.

A new tool called *Valgrind*[valgrind] has recently been released to check all reads and writes of memory for use of uninitialised variables, reading/writing after memory has been freed and memory leaks. Valgrind will not catch all memory problems because it can only detect those which occur during a running occurrence of the programme. To find all problems you would have to run the programme through all possible states, a practical impossibility in a code base of this size. So while I was busy detecting user reported bugs, one of the other developers ran Valgrind over Umbrello. It did not pick up any of the reported problems but it did pick up many unreported ones and it is likely that this solved many problems before they were ever noticed.

Many of the errors which have been fixed are interface inconsistencies. Menu items with different names for the same operation, buttons in the wrong order or unclear text. Again, refactoring of code to remove duplications, usually just creating a method of one or two lines, can prevent inconsistencies occurring.

Features

I have added several new features to Umbrello since I took over control of the programme.

Canvas Zoom and Resize

Zoom and resize ability for the diagram canvas was a much requested feature. The fixed size canvas that Umbrello had was a severe limitation for creating diagrams of any substantial size. Being able to zoom in and out is important for users to be able to get an overview of their diagram. I was therefore not surprised when these two features were the first to be submitted after the end of the feature freeze for the 1.1 version. The patch used affine matrix transformations for zoom. Using the patch as a starting point I gave the code the ability to save to and load from the file and removed the need for the user to manually resize the canvas by automatically resizing it as needed after anything was moved. The affine matrix transformations caused some problems with the placing of new widgets and right click menus, the positioning for these had to be changed to go through the matrices as well. Toolbar buttons and

Properties dialogue widgets were added. This update has been one of the main reasons why people use the unreleased version of Umbrello over the released version.

Undo and Redo

When I was first designing the undo and redo code I thought it would require a mechanism to save and reload every possible change to the user's model and diagrams. This seemed like a nightmare to implement until I realised it would be possible to just save the whole model using the same code used to save it to the file. This required some changes to the save and load code to allow it to work on arbitrary buffers, not just files. Any changes to the model are saved to an undo stack. When the undo function is called, the top item is loaded from the stack and the current model is saved to the redo function. The undo and redo stacks have limits on them to prevent them taking up excessive amount of memory. Loading a whole model afresh on each undo or redo does cause some flickering as the old model is discarded and the new one loaded, I would like to work on the loading mechanism to cause a smoother redraw but this is currently a low priority.

Parameterised Classes

A parameterised class is a fancy name for what C++ calls a template class. I had been getting calls to allow this ability for classes as it is useful for data structures. When it was announced that Java 1.5 would have them added to the language[`java-generics`] I received even more requests for the feature. Technically templates are a lot like attributes and operations, they are all components of classes. I implemented them by essentially copying the code for attributes and adapting it to the requirements of templates. The amount of duplicated code convinced me that I needed to refactor attributes, operations and templates to make them share much more of their code.

Interfaces

Interfaces are essentially abstract classes (which means their methods are not implemented) without any attributes. In UML they can be drawn either as boxes, in the same way as classes, or as circles. Implementing interfaces required a lot of copying and duplicating existing code from the classes code, in the same way as templates copied code from attributes. While the interfaces code worked perfectly I was unhappy with the amount of duplicated code and made it a priority to refactor this area.

Component Diagrams and Deployment Diagrams

As a UML programme I felt it important that Umbrello implemented as much of UML as possible. The two diagram types specified by UML and missing from Umbrello were component diagrams and deployment diagrams. Implementing these required new widgets - components and nodes - and adding new items to the tree view. I broke the file format when adding to the list view. It is saved out with the rest of the data and the enumerations used to identify the type of each tree view item have to be carefully edited to prevent old documents loading tree view items as incorrect types.

Cut and Delete Selected

One of the most obvious missing features was cut. The programme had a (buggy) copy and paste feature but no ability to cut. Cut is simply a combination of copy followed by deleting the selection.

Unfortunately there was no delete function either so I had to define this too as a Qt slot method which iterated over the list of selected widgets deleting each one. Because of the different copy types (widgets, tree list view objects) the cut method had to detect the correct items to delete. The cut method is a good example of how breaking up a function can result in reusable code for other functions.

User Interface

Umbrello has a clean and friendly user interface which is quick to get used to. It integrates well with the rest of KDE because it uses all the standard KDE programming interfaces.

I made several small improvements to the user interface where inconsistencies existed or improvements could be made.

- I replaced the splitters which were used to define the main elements of the application with *KDockWidgets*. This powerful class allows widgets which can be attached to the edges of the application window and arbitrarily moved by users. Their positions are saved and reloaded automatically. It results in simpler code inside Umbrello and more advanced functionality for the users.
- The attribute and operation dialogue boxes required a right click to perform any functions. I added buttons and refactored the code to allow it to work with the buttons. This makes the dialogues much more intuitive.
- A couple of dialogue boxes had their *OK* and *Cancel* buttons the wrong way around. The KDE user interface guidelines dictate that *OK* is to the left of *Cancel*, which is the same as Windows and the opposite of GNOME or MacOS. I had never had to think about this before because KDE's dialogue classes are advanced enough that you never have to code the buttons yourself. The code for dialogues was going out of its way to reimplement functionality and add bugs in the process. I ported the dialogues to the correct classes and they now display the standard button ordering.
- I manually implemented tab ordering and setting the focus for dialogues where this did not successfully happen automatically. The libraries usually guess correctly the tab ordering and initial focus but some dialogues needed it explicitly coded. This is important for users who prefer not to have to swap between their mouse and keyboard.
- KDE contains a powerful widget theming ability for users who like their eye candy. A theme called *scheck[scheck]* contains no good looking graphics but instead has the clever ability to check widgets for missing or incorrect keyboard accelerators and capitalising errors. Any errors it finds are highlighted in a suitable colour. Using this tool I was able to ensure that Umbrello conformed to KDE's user interface guidelines.
- UML contains several different association types for showing the relationship between two objects. Realisation and generalisation are two similar association types which are never used between the same two object types. I decided to merge the toolbar buttons for this function and implement a method which detected what type of association should be created depending on which type of objects are being associated. This is an example of the help which can be given to the users through a UML specific programme compared to a general diagramming one.

Refactoring Classes

Refactoring is restructuring code to simplify and enhance its internal abilities while retaining the same outward functionality. In practice it can mean creating new, usually private, methods to remove potentially duplicated code or adding new classes when the current structure of the code has become difficult to maintain.

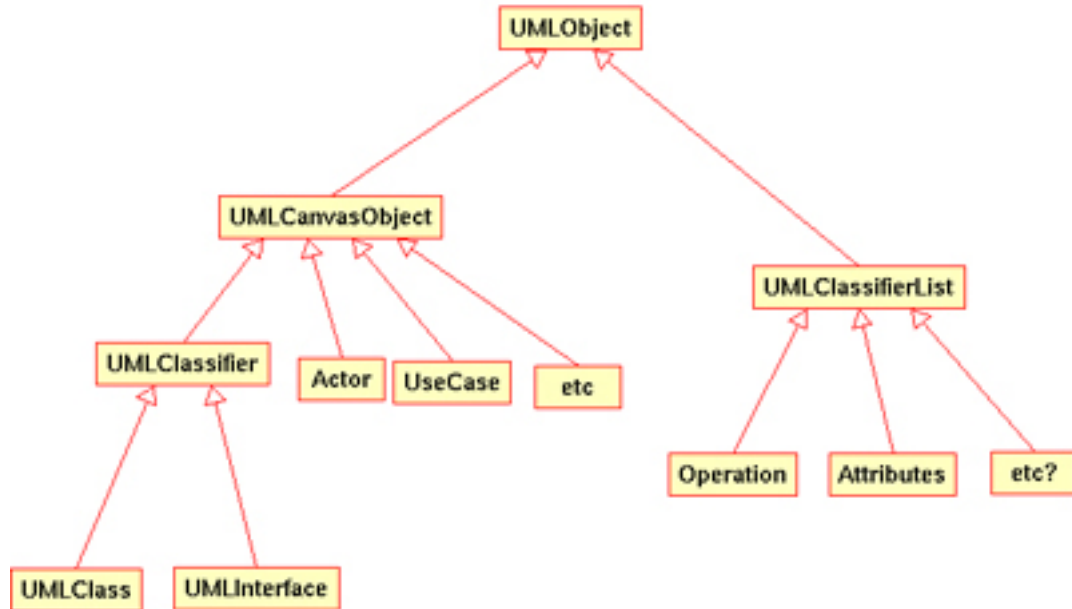
UMLAssociation

Where appropriate Umbrello stores items as a combination of internal UML objects which can be seen in the tree view and displayed widgets which can be seen on the diagrams. This allows the same objects to be represented in more than one diagram. Associations were not treated this way and were only represented as diagram widgets. This caused code generation to be much more complex than it had to be as all the diagrams had to be searched for relevant associations.

Adding a new class *UMLAssociation* to move the associations into the internal UML model required a change in the file format and associated code to successfully import old version files. It also provided the opportunity to add more functionality to associations which now allow for roles, constraints and documentation at each association end. There were significant changes required to each of the other UML objects to hold a list of their associations and several bugs were introduced, not all of which have yet been fixed (such as saving break points). However it greatly simplified the code export routines which no longer have to search through the diagrams looking for associations.

UMLObject

UMLObject is the super-class used for items held in the internal UML model. This includes items which will be displayed as widgets such as classes and activities, classifier items such as attributes and operations, and following the refactoring of *UMLAssociation*, associations. This is a diverse group of functions for one class and I found that the derived classes often shared a lot of code. To review what was the existing situation I used Umbrello to create a class diagram of the existing layout.



New UMLObject design

The refactoring of the widget related UML objects went smoothly. In one move I was able to remove large sections of duplicated code, update classes which did not have the new association code and reduce dependence throughout the programme on checking enum values, methods which check for canvas objects can now simply check if the object is a *UMLCanvasObject*. Another developer volunteered to create the *UMLClassifier* class to sit between *UMLCanvasObject* and the *UMLInterface* and *UMLConcept* classes (*UMLConcept* was also renamed to *UMLClass* to reduce confusion). It will now be a trivial matter to add Datatypes and other UML classifiers.

I had less luck with refactoring the code for attributed, operations and templates. My hope was to create a generic framework which could easily be extended to other classifier properties such as exceptions or slots which are also allowed by UML. The framework would also allow for stereotypes to be mixed in with the lists, an occasionally requested feature. Unfortunately the classes involved share little code which is not already shared in *UMLObject*, they are simply not the same classes on any lower level. Add the possibility of stereotypes to the requirements and there is no shared functionality at all. The code is also heavily reliant on these classes being direct children classes of *UMLObject*, with many hard coded calls to *parent()*. In the end I had to abandon my work on this. It seems likely that any general framework to handle more arbitrary classifier lists will have to work with *UMLObject* rather than a more specific class.

Data Classes

One of the stranger and more problematic elements of the design of Umbrello is the separation of the widget classes into two, one half derived from *UMLWidget* which contain the code to draw the widget on the canvas, and the other half derived from *UMLWidgetData* which stores the properties of the widget. The theory of this is a separation between display and model but in practice all properties are accessed through the *UMLWidget* anyway and Umbrello already has a separation between display and model, the UML objects exist quite separately from the widgets.

The *UMLWidgetData* class and its derivatives complicated the design of Umbrello significantly, but I decided not to get rid of them. The changes would have been too much effort for little programmer gain, and no end user advantage. It would almost certainly have introduced bugs to the programme and it would have been difficult to keep file compatibility. There is a trade off between creating a clean internal design and actually creating the features that users will see and in this case I considered that with the time constraints it would be hard to justify.

Use of Multiple Inheritance

Umbrello makes only one use of multiple inheritance, a practice usually condemned by software engineers. *UMLWidget* inherits both from *QCanvasRectangle* which draws the widget on the canvas to be displayed to the user, and *QObject* which is the base class of most of Qt's classes.

The purpose of this dual inheritance is not to implement the functionality of both classes but to allow for the language extensions used by Qt but only allowed in classes derived from *QObject*. *UMLWidget* makes use of Qt's extensions for mouse and other events. It seems strange that *QCanvasRectangle* is not derived from *QObject*, it is one of the few classes in the library not to. Presumably this is for optimisation reasons.

Use of XMI

The *XML Metadata Interchange* language is an XML language framework which can be used to store data from object based metamodels such as UML. It is a complex and rapidly developing language made more complex by the mix of versions of XMI with versions of UML and the lack of documentation available. In theory XMI allows for the sharing of UML models between modelling programmes, however XMI does not include visual information such as the positioning of widgets, nor does it allow for documentation. All programmes which use XMI must therefore extend it and in practice there is little portability of XMI files between programmes. Umbrello uses XMI as a basis for its file format.

I created an example file[umbrello-example-file] for Umbrello which contains all the objects, widgets and attributes possible. This is used as a test file to ensure compatibility in future versions. I submitted this file to the XMI mailing list[xmi-mailing-list] following a request for example implementation XMI files. The feedback from this confirmed that Umbrello's file format is essentially incompatible with other XMI implementations and some improvements were suggested. However it would be hard to implement these suggestions without breaking backwards compatibility of the file format and it would be hard to see the justification when other UML programmes produce XMI which is just as incompatible.

To make the file import code more compliant I added a check for the XMI metamodel being UML. Unfortunately this broke backwards compatibility of the format because an old alpha version of

Umbrello did not save the metamodel to the file. Ensuring compatibility of file formats is a difficult task but it is especially difficult with openly developed software where pre-release versions are always available and people may come to rely on them.

By the time I had added new diagrams and objects to Umbrello, and refactored the association code the clipboard code had become out of date and contained many bugs. The clipboard is based on the original file format which used *serialise()* methods to write out the state of all classes to a data stream. These methods are hard to maintain (impossibly so if keeping backwards compatibility) and essentially duplicate the functionality of the more advanced XMI load and save code. I decided it would be easier and allow for simpler maintenance to re-write the clipboard to use XMI data rather than serialised data. This is more complex than the undo/redo code as it is not saving the whole file out at any time, the programme has to know which parts to save and requires custom loading code for each type of data which can be copied, however I was able to reuse some of the existing algorithms and methods. The result works well and should continue to work with minimal maintenance as more features are added to the programme.

Compatibility with Solaris

While I have been unable to test Umbrello on Solaris myself, there have been a couple of users who tried to compile it and failed. The compilation error was with a C library function *setenv()* which doesn't exist in Solaris. Fortunately this problem has already been solved by the KDE developers who provide an alternative *setenv()* in a compatibility library so it just required a test in the autoconf file for the *setenv()* function. Umbrello now successfully compiles on Solaris.

Chapter 4. Project Tools

Hosting on Sourceforge.net

Umbrello was, and largely still is, hosted on Sourceforge.net[sourceforge.net]. Sourceforge hosts Free Software projects and provides a collection of tools to enable and enhance collaborative development. The tools include revision control, bug and feature trackers, web site hosting, mailing lists and file releases. There are individual equivalents to all of these tools and Sourceforge makes use of many of these external projects but brings them under a unified web interface with a single login and point of administration. Many projects which are organised in the same style as Umbrello set up their own servers to host the projects but using Sourceforge I have been able to concentrate on managing the development of Umbrello rather than configuring management tools. Sourceforge's main downside is that it is a victim of its own success. It is relied upon by so many Free Software projects that if it were to lose funding or experience technical failure it would prevent development of a lot of programmes.

When Umbrello became part of KDE I moved the bug/feature trackers and CVS repository to KDE's facilities. This brings a large body of KDE developers into contact with Umbrello's code.

CVS Revision Control

CVS (Concurrent Versions System) is a fundamental tool used by almost all openly developed programmes. It is a client/server based revision control tool which stores files in a repository on the server. The files are checked out by developers to their local machines where they are edited, then the changes are committed back to the repository. It allows multiple developers to work on the same files at the same time by intelligently merging commits into the repository or, the other way, updates from the repository into local files a developer has already edited.

As a revision controller it allows previous versions of files to be checked out and any changes can be easily reverted. This is useful for tracing bugs, by going through previous versions it is possible to track down what commit introduced the bug.

Shortly after I took control of Umbrello I setup CVS to send information on any commits made on the repository to the developer mailing list. This has proved invaluable in tracking the work done by developers. Previously any commits had to be either announced manually on the mailing list or caught by chance while viewing the web CVS interface. I also imposed a rule that all commits must have a descriptive comment (previously most commits had no comment at all). The CVS e-mails complement the discussion e-mails to the mailing list and ensures everyone knows what has been happening to the code base.

One other feature I made use of in CVS is revision tagging. Whenever I make a release of Umbrello the files are tagged with that version number (for example UMBRELLO_1_1_BETA_1_TAG). The tag can then be used to refer to a release if someone wants to get the source of it out of CVS, or more usually to compare one or more files from that version with the current CVS.

A limitation of CVS is that commits of several files are not atomic: if a commit fails half way through uploading some files the repository will be in an inconsistent state, which is what had happened when I first became administrator for Umbrello.

diff and *patch* are two vital tools for sharing modifications with others. *diff* extracts the difference between two files in an easily human and computer readable format and *patch* merges a diff file with the original file. CVS makes use of these programmes when comparing local files to those held in a repository and it can also be used to create correctly configured diff files.

Bug and Feature Trackers

While bugs and features being worked on are always discussed on the developers mailing list it is also important to keep a formal list of them. The trackers do this and allow a priority number to be given to an entry, as well as comments and patches to be attached to each entry.

I have made an effort to keep the bug and feature trackers up to date by opening and closing entries whenever a new bug is reported or one is fixed. Anyone can add a new entry to the trackers and many bugs and features are added directly to them as a quicker alternative to having to subscribe to the mailing lists. As with CVS, I set up the trackers to send changes to the developer mailing list to ensure everyone is informed of any changes.

Two other trackers are offered by Sourceforge, patches and support. I have not found them to be very useful because they offer little over what can be done with mailing lists. The feature to assign tracker items to certain developers has also been little used possibly because it signifies a definite commitment to do the job which they may not be able to fulfil.

Website

Umbrello already had a useful website[umbrello-website] when I became project manager, but I tidied it up to make it XHTML compliant (and thus accessible to any web browser) and have added some features to it which enhances its usefulness to people interested in the project.

- A "Current Releases" box on the front page gives a quick notification of the latest stable and development versions.
- A Developer's Resources page with links to useful information such as the generated documentation from the programme sources, the daily CVS tarball produced by Sourceforge and comments and patches submitted by developers.
- The Frequently Asked Questions page has helpful hints to developers and users. I generally put any topic which occurs more than once on the mailing lists onto this page.
- The download page contains links to all the latest official and third party packages as well as instructions on checking out the sources from CVS, the complementary install page has instructions for installing including help sent in from users who have solved some problems.
- The front page contains a general description of the programme and UML and the project news to let browsers quickly familiarise themselves with Umbrello.

I grant all developers access to the website, but in practice few changes have been made except by myself. Despite this the website is an essential communications tool both with the public and other developers.

Releases

One of the important rules of open development is to "release often and release early". This ensure that developers and contributing users are up to date with the latest features and bugs. When I took control of Umbrello there had only been one unofficial release (made by me) in almost a year. My first task was to package and release the current code in CVS as a beta version. Since then I have tried to make releases more frequent than once a month.

I also set up a script, run nightly, which checks out the latest sources from CVS and packages it as a source release. This is available from the website along with the official releases. Unlike an official release, a nightly build has no guarantee of not including major bugs or even compiling at all.

Binary packages are made of official releases so most users will not have to compile the programme themselves. One of the problems with Unix-like operating systems is the enormous number of (often just slightly) incompatible configurations. I make packages for the configurations I have access to but other packages are contributed by users and developers. Requesting packages is one way to gently introduce people into contributing to Umbrello.

The releases are numbered to indicate their stability. The first release after I took over was 1.1-beta1. After another beta I made 1.1-rc1, a release candidate. The theory of a release candidate is that it will becomes the final release without any changes, although in practice this rarely happens. Finally 1.1 was released In January 2003. The larger number of users that the promise of a stable release brought to the code meant a number of bugs were found, including one which prevented diagrams from being printed. I considered this serious enough to quickly create a bug-fix release numbered 1.1.1. The current code in KDE's CVS is numbered 1.2-alpha to indicate that it has too many bugs to be of real world use.

Translations

A German Umbrello user kindly translated the programme into German which threw up a number of issues. KDE has a function `i18n()` which is put around any string that should be translated. When a translation was attempted it became clear that there were many strings with which did not use that function. I wrote a Perl script to search for strings which needed translated[fish-untranslated] but had to be careful to filter out strings which should not be changed such as XML tags, image files, debugging messages and `#include` lines. I also needed to edit the automake files to allow for translation of the "tip of the day" file by setting it to run a programme which prepares the file to be translated.

Mailing Lists

I have three mailing lists for Umbrello, one for developers, one for users and a low traffic list for announcements. The developers list, `uml-devel`, is by far the most active and is where most of the discussion about the programme occurs. It is also the target of the mailings generated by commits to CVS and changes to the bugs database. Threads on this list can be anything from proposing a psuedo-metamodel that plugs in at the reflective layer to wishing the other developers a happy new year. `Uml-user` is a lower traffic list and mainly consists of people having problems compiling the programme. These are usually general trivial problems with their Unix install which would prevent any KDE programme from being compiled. The users list can often venture into the topics of the developers list due to the technical nature of the programme but the list is worth keeping to prevent trivial topics

flooding the developer's list. The `uml-announce` list is a very low volume list used to announce new releases.

IRC

Internet chat is used by many projects to allow for an interaction not possible through e-mail. Trivial problems can be asked about and solved almost instantly while more lengthy ideas can be discussed at length bouncing ideas off each other. I proposed an IRC channel for Umbrello which would be used for discussing development ideas and helping users but there was little interest. There are simply not enough developers to justify an IRC channel and the volume of traffic on the developers list is small enough for people to follow without having to catch up on the day's news. However I did make extensive use of IRC for talking to KDE developers, asking questions where I could not find the answers in documentation or mailing list archives, and especially for discussing the technical details of the integration of Umbrello into KDE.

Class Documentation

Like an increasing number of programming platforms, KDE allows for the ability to document code through specified comments above the class, attribute and method declarations. There is a programme, *kdoc*, which produces formatted HTML from these comments and I put the output from this on the developers resources page on the website[`umbrello-kdoc-output`]. It has proved useful for class reference but since the documentation is already directly in the code I mostly just read it direct from there. Just as useful is the error messages *kdoc* creates for missing documentation which help to enforce discipline among programmers to always document their code. Whenever I am working on a method or class which has not been documented I ensure that I complete the documentation immediately.

Handbook

A *Handbook* help file is a standard feature of KDE programmes. The handbook was badly outdated and included little that was of help to most users so I put out a request for a new handbook which should contain an introduction to UML and to Umbrello and a kind volunteer spent several days creating it. The handbooks are written in Docbook, a standard XML language for technical documentation. The submitted new handbook did not validate as correct XML so I used HTML Tidy[`html-tidy`] in XML mode to fix most of the problems and spent a day going through the document by hand fixing grammar and spelling and XML. Documentation is a fairly boring part of development, but its value to users is immense.

Text Editor

I use GNU Emacs[`gnu-emacs`] as a text editor while developing. Although its 30 year history gives it a user interface which is hard to pick up, its features are unrivalled, and unlike pure GUI text editors it works on a text console meaning files can be edited from any computer on the internet. KDE provides extremely useful scripts for Emacs which provide it with features I have come to depend upon such as

variable name tab completion, jumping between equivalent methods in headers and body files and inserting debugging lines.

KDE does include an excellent integrated development environment called KDevelop[kdevelop]. I have tended to shy away from using KDevelop and IDEs in general because I find it is just as easy to use the individual tool as it is to use them in an integrated environment. KDevelop would be yet another tool that I would have to learn to use but offers little advantages compared to working without it. However, many other Umbrello developers successfully use KDevelop and other environments and it is important to keep the programme neutral of development environments to allow each developer to use the tools which they find most comfortable.

Chapter 5. Using The Bazaar Development Process

Free Software is as old as computers which can run software, but the rise of propriety software in the seventies and eighties left it as virtually the only option for desktop software until a few years ago. The recent reversal in trends has been almost entirely due to the massive increase in Internet access which has allowed for a new software development model, the bazaar method. This open, collaborative method allows anyone to participate with leadership largely based on merit. It is not chaos as peer pressure and a project leader or leaders ensure a focused project but, because this is Free Software, if anyone is unhappy with the result they are free to work on the problems themselves. The communication possibilities created by the Internet mean it can be worked upon by people with a genuine interest in the project regardless of their geographical location. The result is software with a reputation for stability and security but also increasingly for user-friendly and innovative features.

Developer Motivation

Openly developed software depends on users contributing back to the programme. Contributions can be large or small, technical or non-technical. They include bug reports, fixes, packages, translations, promotion, documentation, feature requests and whole new features. Helping a project such as Umbrello takes time of course, so why do so many people do it? The answer in almost all cases is that the developer is "scratching an itch", they have a need for a programme which performs a specific function and if one doesn't exist the easiest way to get what they want is to create it themselves. Umbrello was created because the author needed a UML Modelling programme for Unix and found that no suitable programmes existed. Because the author had no financial interest in the programme he made it publicly available at no cost and soon others with a similar need were contributing. Most developers work unpaid on these projects but many work on them as part of their professional work and a few are employed full time. There is also a significant contribution from university students such as myself. My experience from Umbrello is that most of our developers contribute partly as a tool for their professional work and partly for their own satisfaction. I have also received help from KDE developers who are employed on KDE full time.

Umbrello Leadership

I was granted leadership of Umbrello from its original author. He had become leader through the merit that he had written it but Umbrello was now becoming an openly developed programme so, like all major decisions, the leadership had to be broadly agreed by the developers. There is no vote in these decisions but it is obvious that a consensus has been reached and any dissent would be clear. Sourceforge offers different levels of permissions to developers and the formal sign of the Umbrello leader is simply that I have full permissions and can grant permissions to others.

As the leader I do most of the development work but also organise and criticise the work of others. This involves asking for specific contributions and commenting on proposals for features or implementation details. I review all the code which goes into Umbrello either as it is sent to me as patches or once it has been committed to CVS, it almost always requires some tidying up. I also do the day to day maintenance of removing spam from mailing lists, marking bugs as invalid or duplicates and answering user's queries.

It is also the leader's job to occasionally refuse contributions. It can be hard to reject a patch when someone has obviously worked hard on it, fortunately there are few cases when this is necessary and usually a bad patch can be worked on to be improved. In the worse of cases other developers will generally offer an opinion on why the given patch would be a bad idea. Most unsuitable patches are the result of insufficient discussion before implementing the idea, which can be my fault as much as the fault of the person who implemented it.

Programme Forks

The position of leader is one of a benevolent dictator, my word is final and any decisions are ultimately made by me. It is benevolent because the freely available and modifiable nature of the source code means anyone can take the code base and start up a rival project if enough bad decisions are made. In reality few such forks have occurred in Free Software programmes, the GNU Emacs and XEmacs split[gnu-emacs-xemacs] probably being the most famous. When forks do occur they can usually be considered branches, one branch used for more adventurous and possibly unstable features, as happened with Samba[samba-tng]. It is the possibility of these forks though which keeps the developers on their toes.

Release Plan

One of my responsibilities as project administrator is to make releases. Although I have never made a release plan listing exact features to be implemented and bugs to be fixed by a certain date I have always had a rough idea of what a release should include. For example 1.1-beta1 was the code that was available when I took over the leadership. 1.1-beta2 included code to read old binary files which was important for not losing long-time users who thought their files were unreadable. 1.1-rc1 was in theory the code which would become 1.1, and the target for 1.1 was a bug free release with the same features as version 1.1-beta1.

Many collaborative projects create a timetable with exact dates for releases and features to be implemented by then. I felt that the development work and number of bugs in Umbrello was too unreliable for this but I did announce a feature freeze as soon as I became Umbrello leader. Due to the number of bugs this was a fairly long feature freeze and, with people so pleased to see development of the programme taking off again, a number of new features did creep in which predictably caused more bugs. It can be difficult to say no to a new feature which is obviously a benefit, but I found a gentle e-mail to the developer concerned indicating the importance of bug free code at this stage in the development would usually cause the bugs to be fixed quickly.

Not all of the GNU/Linux distributions make packages for Umbrello and it is important for us to provide binary packages for as many as possible. From the help requests I get sent it is surprising, for such a technical audience, how many people have trouble compiling a programme. Providing packages is also an easy way to get people involved in the development of the programme.

Encouraging Participation

It is important to encourage and remind users that they can and should contribute back to Umbrello. I always accompany any announcement of a new version with a request for feedback and a statement that any help would be appreciated. Whenever a new feature is added or a bug fixed I make a point of thanking the developer for their help.

I have a policy of offering CVS write access to anyone who has shown that they can contribute to the project. This usually means submitting a patch or two to the developers list. Before I give new developers access to CVS I ask for a brief description of themselves and why they want to work on Umbrello which I put on the website. This is mainly to give some context to a name and helps the developers build familiarity with each other.

Publicity

Publicity is as important to openly developed programmes as it is to commercial programmes. Without publicity we would have few users and that means fewer developers. Umbrello used to not have a name other than UML Modeller which caused confusion and was too generic for some packagers. I organised a poll on the website offering a number of choices and found a large consensus towards 'Umbrello'. The poll and rename was also a signal to users that the programme was going to get moving again.

The website is the most useful publicity tool, it allows users to get an overview and description of the programme before they try it. It includes a page of screenshots showing different functions of Umbrello which offers potential users a quick way to get a feel for what the programme can do. I also added an online version of the Umbrello handbook which contains a detailed description of not just Umbrello but also UML to help convert those who are not accustomed to object modelling.

Umbrello releases are announced on freshmeat[freshmeat-umbrello] and apps.kde.com[kde-apps-umbrello]. The statistics from these sites show that Umbrello has had several thousand downloads through them. I did not send announcements of Umbrello releases to paper based publications but a German developer did and there was an article about it in c't magazine[ct-umbrello-article] so perhaps I should have made more effort to contact English language publications.

Inclusion in KDE

KDE is the most popular and (arguably) technically accomplished desktop for Unix. It has made Unix user-friendly enough to be used by any computer user and the large number of applications available for the platform mean users need never know they are using anything other than KDE. KDE is not controlled by any one organisation or company, its direction is solely dictated by its developers. Unlike most bazaar developed projects it has no formal leadership or group of core developers probably because it is such a large project that nobody could manage all of it. Instead each of the applications has its own organisational method and it is considered polite to pay special respect to long standing contributors and those who do a lot of work for KDE (which usually means people who are employed to work on KDE).

The idea of including Umbrello in KDE had been around for some time but it was only after the 1.1 release that Umbrello was mature enough and the release cycles coincided with a recent release of KDE. I suggested the idea on KDE's developers IRC channel to get a feel for what KDE developers would think about including Umbrello. The idea was universally accepted and I followed it up with a more formal proposal to the kde-core-devel mailing list. There was some concern that as the bug tracker was hosted on Sourceforge it would not be a proper part of KDE but I replied that I fully intended to use the KDE bug tracker to ensure that Umbrello was a proper KDE citizen.

KDE developers, like Umbrello developers, need an account on KDE's CVS. Fortunately I already had an account from previous work I had done with KDE. The integration of Umbrello's code base into the kde module kdesdk required some changes to the build files and, because KDE does not allow casting of

strings, a few changes to the actual code. I packaged the result, and had it checked over by a KDE developer who had shown interest in helping, before committing it to KDE's CVS repository. With hundreds of KDE developers dependent upon a clean compile of the source it was vital to ensure there would be no problems for any configuration. The speed of IRC came into its own here as several experienced KDE developers and myself fixed possible build issues such as outdated Makefiles, reliance on C functions and the problems with dynamically loadable libraries used for code generators. I moved the items in the Sourceforge bugs and features trackers over to KDE's bug tracker by hand which offered an opportunity to remove some outdated requests and reports. Finally the KDE translation teams were informed of the new programme and I made the current Umbrello translations available for them to add to their KDE translations.

Integration into KDE has already brought us a number of new developers and users. The next release of KDE, which will include Umbrello, will bring UML modelling to thousands of Unix desktops for the first time.

Competition to Umbrello

While I stated earlier that Umbrello is not heavily influenced by its more commercial competition, there are a few spin off projects from Umbrello which I have decided not to develop for personally but which I offer support and some publicity from Umbrello.

xmi2code[xmi2code] generates programming code from Umbrello's XMI based file format. Umbrello generates programming code from its own internal objects, a method I prefer for its comparative simplicity, as there is no need to write file parsing code twice and no dependence on file format compatibility.

xmi2html[xmi2html] is a script which uses XSLT to turn Umbrello files into HTML documentation. This is an occasionally requested feature of Umbrello but one which I have avoided because the generated code already includes any documentation in a format specific to that code. Most languages and platforms have tools to convert this to HTML or other formats already, as I have done with Umbrello and kdoc, and this is the preferred method for most developers.

Most significantly there is an *Umbrello 2*[umbrello2]. It is a completely new programme written from scratch by an Umbrello developer who felt the current code base was difficult to maintain and extend. It is based entirely around MOF and XMI. Currently it does not include a user interface or any code which of more than academic interest. I disagree that a complete rewrite is the best way to improve Umbrello and have found that with refactoring and tidying the code is perfectly maintainable, moreover the current code for *Umbrello 2* is almost as large and complex as Umbrello's code without any end user functionality. However if *Umbrello 2* does turn into a functioning programme it will encompass much more than UML and be a unique, general and extendible object orientated modelling base.

While it is a shame that the developer time that has gone into these projects has not gone into Umbrello it is important to remember that Free Software is about choice. I would much rather developers who disagreed with some part of the direction of Umbrello's development did so within the project rather than start a completely separate project. By keeping them generally supported by Umbrello, if these offerings do clearly become superior to what Umbrello already offers then they can be easily integrated.

Extendible API

One of the areas of Umbrello which has seen a lot of development, especially from outside contributors,

is the code generation feature. I have had code generators sent to me for 11 different languages including C++ and Java, and even JavaScript, XML Schema and SQL. While some of these are obscure for use with object orientated modelling if someone has taken the time and effort to code them it means they must get used.

The reason for the large number of contributions by developers who have often never worked on the programme before is the extendible and clear programming interface for code generators. To create a code generator it is only necessary to understand the CodeGenerator class[codegenerator], no understanding of the internal functioning of Umbrello is required. The API creates an easy way for programmers to contribute to the programme and is particularly suitable for the code generation function because of the large number of programming languages which could be required but which neither I nor the other regular contributors use.

By contrast the code import function, a more complex but similar programming concept, has had no work done on it since it was first introduced with only one language, C++, supported. I have recently been in discussions with developers about creating a defined programming interface for this feature modelled along the code export one. This would allow the success of code export to be shared by enabling developers new to Umbrello to easily extend code import to their own preferred programming languages.

The C++ code import parser is largely taken from KDevelop and effectively represents a fork in their code. It is an example of how Free Software allows for effective code reuse but improvements in the equivalent KDevelop feature have not been ported into Umbrello. Creating a code import API would have the added bonus of allowing Umbrello to link to the KDevelop import library which would prevent two parallel branches of the same code.

Free Software

What has allowed for the bazaar development model is the removal of artificial restrictions on our use of software brought about by copyrights. Umbrello uses a copying license called the GNU General Purpose License[gnu-gpl] which uses copyright to reverse its normal restrictions and instead allow the copying of the software only on condition that the freedoms to modify and share it are ensured. Copyright has been around for so long that few people consider how, with increases in technology, it now restricts individual freedoms rather than those of commercial publishers. The effect is particularly noticeable in the software industry where copyright has created monopolies that restrict innovation and competition to the detriment of society. Free Software does not prevent commercial software development, Umbrello is sold as part of commercial GNU/Linux distributions and some of its developers contribute to Umbrello as part of their professional work.

Software, both Free and proprietary, is further threatened by software patents, which restrict the very ideas and algorithms used. Software patents are currently illegal in Europe but many thousands are nevertheless granted every year and a proposed EU directive[eu-swpat-directive] will likely soon make them legal. Software patents are something most programmers and educated users object to. Umbrello is probably potentially threatened by dozens of patents, it would be impossible to search for them all, but the Togethersoft patent[software-patents] which restricts round trip code engineering between diagrams and programming code is particularly worrying.

The advent of Digital Restrictions Management technologies now legally protected in the US and due to come into Scots law under a recent EU directive will further threaten what programmers and users can achieve with computers. *Trusted Computing* schemes such as Microsoft's *Palladium* will prevent users from running Free Software programmes such as Umbrello on their own computers.

Chapter 6. Conclusion

Over the course of this project I have taken a stagnant and bug-ridden programme and engineered it to become a useful and unique application with a vibrant group of developers which will ensure it continues to improve and support the increasing number of users. The unglamorous but vital initial technical work of removing the accumulation of bugs gave me time to familiarise myself with the tools and social leadership required for the project, and allowed the creation of a stable release which users could depend upon. My successful integration of Umbrello into KDE will ensure that it is no longer a fringe programme but will be distributed and treated as a central part of modern Free Software desktop operating systems. The new features I have added, such as multiple undo and a functioning clipboard, as well as the implementation of all UML diagram types, brings the programme up to a quality required for industry use. The promotional work I have done has brought Umbrello and UML modelling to a larger audience and it is now included in commercial GNU/Linux distributions from Mandrake[mandrake] and SuSE[suse].

Free Software development traditionally has paid little heed to software engineering techniques used elsewhere in the computing industry. Umbrello is starting to bring UML modelling to Free Software development, initially by documenting existing object structures of programmes such as Quanta the HTML editor[quanta] and Karbon the vector drawing programme. It is likely that by expanding and manipulating these diagrams the programmers will find, as I have done with my work on Umbrello, that UML gives a powerful abstraction from details which greatly simplifies high level programme design. Umbrello is now also used by a number of commercial developers, including NASA and Nextphere[nextphere], who allow programmers to contribute to Umbrello as part of their employment.

The Bazaar Method

Bazaar development works well for a programme such as Umbrello. It is a technical programme with technical users who are all familiar with software development. An indication of how well suited it is to the method comes from the number of people who are working on it as part of their job. This is an encouraging development and it would be excellent to see this practice spread to less technical programmes. In the other direction efforts could be made to encourage participation from users who are less technical but nether-the-less motivated by a desire to better the tools they have. One way to do this might be to improve the bug and feature reporting facilities, the user-interface of which can be hard to use.

A leader with good communication skills and a vision for the general course of the programme is an essential requirement for bazaar development. One of the obvious problems with the bazaar method is that it takes place over the Internet which gives limited communication possibilities compared to face to face development. As the leader I made sure to build familiarity within the developers by insisting on a personal paragraph for each developer on the website. I also made sure to use the available communications mechanisms as effectively as possible by keeping the news section of the website updated, and ensuring every e-mail and bug entry is responded to. It is also important that the leader is more than an administrator but also takes an active role in development. Since the organisation works on merit the leader or core developers must merit their position to hold sway with the other developers.

With hindsight I think I would have made it clearer what stage of the development cycle we were at, perhaps a box on the front page of the website to indicate whether the development focus was on debugging or on new features. The biggest problem I have found with supporting Umbrello is solving

problems caused by users who have differently configured computers. Fortunately the bazaar method is perfect for helping with the multitude of possibilities allowed for by Unix, and somebody who has come across the same problem will almost always be able to offer help.

Future Work

The future for Umbrello seems bright with development continuing at a rapid pace. Already, during the writing of this report, there has been a class refactoring assistant added to allow easy manipulation of operations and attributes between classes, and there has been much discussion on an improved code exporter which would stop changes in the files being overwritten and allow for roundtrip software engineering. Umbrello's user base now numbers several thousand and there are about 200 people subscribed to the mailing lists. Between the mailing lists, bug tracker, IRC and being contacted directly I now receive about half a dozen requests for support and congratulations for the programme a day. As part of KDE, Umbrello will now follow the KDE release schedule, including its translation and feature freezes, which will ensure it retains its current level of stability. I hope to take an active part in these and future developments.

Colophon

This document was created with Docbook 3.1 and formatted using sgm12x, jade and docbook-stylesheets. The accompanying presentation was created in XHTML using CSS 2 in presentation mode and Opera in full-screen mode to display as a presentation. The project diary was made in HTML and signed using Gnu Privacy Guard (GPG).

The permanent URL for this document is <http://jridell.org/programs/umbrello/html/>.

The permanent URL for the diary is <http://jridell.org/programs/umbrello/diary.html>.

The permanent URL for the presentation is <http://jridell.org/programs/umbrello/presentation.html>.

This document and the presentation may be freely copied under the terms of the GNU Free Documentation License. The diary may be freely copied, in whole or in part, only if unmodified.

References

- [argo] *Argo UML*, <http://ftp.ics.uci.edu/pub/eden/papers/conferences/1999/coset/CoSET99.pdf>.
- [astyle] *astyle*, <http://astyle.sf.net/>.
- [cathedral-bazaar] Eric Raymond, *The Cathedral and the Bazaar*,
<http://www.tuxedo.org/~esr/writings/cathedral-bazaar/>.
- [ct-umbrello-article] *C't Umbrello Article*, c't magazine, 04/2003, Page 37.
- [dia] *Dia*, <http://www.lysator.liu.se/~alla/dia/>.
- [fish-untranslated] *Fish Untranslated*, http://uml.sf.net/developers/fish_untranslated.pl.
- [freshmeat-umbrello] *Freshmeat Umbrello page*, <http://freshmeat.net/projects/uml/>.
- [gnu-gpl] *GNU GPL*, <http://www.gnu.org/copyleft/gpl.html>.
- [gnu-emacs] *GNU Emacs*, <http://www.gnu.org/software/emacs/>.
- [gnu-emacs-xemacs] *GNU Emacs Vs XEmacs*,
<http://www.xemacs.org/About/XEmacsVsGNUemacs.html>.
- [codegenerator] *CodeGenerator Class*, <http://uml.sf.net/kdoc/CodeGenerator.html>.
- [eu-swpat-directive] *EU Directive on Patentability of Computer-implemented Inventions*,
http://europa.eu.int/comm/internal_market/en/indprop/comp/.
- [html-tidy] *HTML Tidy*, <http://www.w3.org/People/Raggett/tidy/>.
- [java-generics] *Java Generics*,
<http://developer.java.sun.com/developer/technicalArticles/releases/generics/>.
- [kapptemplate] *Kapptemplate*, <http://home.earthlink.net/~granroth/kapptemplate/>.
- [kde] *KDE*, <http://www.kde.org/>.
- [kdevelop] *KDevelop*, <http://www.kdevelop.org/>.
- [kde-apps-umbrello] *KDE apps Umbrello page*, <http://apps.kde.com/nf/2/info/id/1264>.
- [kivio] *Kivio*, <http://www.thekompany.com/projects/kivio/>.
- [mandrake] *Mandrake*, <http://www.mandrake.com/>.
- [mof] *The Meta Object Facility*, <http://www.omg.org/technology/cwm/>.
- [mythical-man-month] Frederick Brooks, 0201835959, Addison-Wesley Pub Co, *The Mythical Man-Month*.

- [nextphere] *Nextphere*, <http://www.nextphere.com/>.
- [qt] *Qt*, <http://www.trolltech.com/>.
- [quanta] *Quanta*, <http://quanta.sf.net/>.
- [rational] *Rational Rose*, <http://www.rational.com/>.
- [samba-tng] *Samba TNG*, <http://us1.samba.org/samba/tng.html>.
- [scheck] *scheck*, <http://webcvs.kde.org/cgi-bin/cvsweb.cgi/kdesdk/scheck/>.
- [software-patents] *Searching the EU and UK Patent Archives*, <http://jriddell.org/patents.html>.
- [sourceforge.net] *Sourceforge.net*, <http://sourceforge.net>.
- [suse] *SuSE*, <http://www.suse.co.uk/>.
- [together] *Together*, <http://www.togethersoft.com/>.
- [trolltech] *Trolltech*, <http://www.trolltech.com/>.
- [umbrello2] *Umbrello 2*, <http://cvs.sf.net/cgi-bin/viewcvs.cgi/uml/umbrello2/>.
- [umbrello-class-diagram] *Umbrello Class Diagram*, <http://uml.sf.net/uml-modeller-class-diagram.png>.
- [umbrello-example-file] *Umbrello Example File*, <http://uml.sf.net/developers/umbrello-example-file.xmi>.
- [umbrello-kdoc-output] *Umbrello Kdoc Output*, <http://uml.sf.net/kdoc/>.
- [umbrello-website] *Umbrello Website*, <http://uml.sf.net/>.
- [uml-specification] *UML 1.4 Specification*, <http://www.omg.org/technology/documents/formal/uml.htm>.
- [valgrind] *Valgrind*, <http://developer.kde.org/~sewardj/>.
- [visio] *Visio*, <http://www.microsoft.com/office/visio/default.asp>.
- [xmi-mailing-list] *XMI Mailing List*, <http://xml.coverpages.org/xmiMailingListAnn.html>.
- [xmi] *XML Metadata Interchange*, <http://www.omg.org/technology/documents/formal/xmi.htm>.
- [xmi2code] *xmi2code*, <http://xmi2code.sf.net/>.
- [xmi2html] *xmi2html*, <http://uml.sf.net/developers/xmi2html/>.

Bibliography

[mastering-xmi] Timothy Grose, Gary Doney, and Stephen Brodsky, 0471384291, John Wiley & Sons,
Mastering XMI.

[uml-user-guide] Grady Booch, Ivar Jacobson, James Rumbaugh, 0201571684, Addison-Wesley Pub Co,
The Unified Modelling Language User Guide.