VICTORIA UNIVERSITY OF WELLINGTON
Te Whare Wananga o te Upoko o te Ika a Maui

School of Mathematical and Computing Sciences
Computer Science

A Lightweight Web-Based Case Tool for UML Class Diagrams

Daniel Mackay, James Noble, Robert Biddle

Technical Report CS-TR-02/21
September 2002

School of Mathematical and Computing Sciences
Victoria University
PO Box 600, Wellington
New Zealand

Tel: +64 4 463 5341
Fax: +64 4 463 5045
Email: Tech.Reports@mcs.vuw.ac.nz
http://www.mcs.vuw.ac.nz/research

VICTORIA UNIVERSITY OF WELLINGTON
Te Whare Wananga o te Upoko o te Ika a Maui

School of Mathematical and Computing Sciences
Computer Science

PO Box 600
Wellington
New Zealand

Tel: +64 4 463 5341, Fax: +64 4 463 5045
Email: Tech.Reports@mcs.vuw.ac.nz
http://www.mcs.vuw.ac.nz/research

A Lightweight Web-Based Case Tool for UML Class Diagrams

Daniel Mackay, James Noble, Robert Biddle

Technical Report CS-TR-02/21
September 2002

Abstract

Many existing UML tools are far too heavyweight for quickly producing a few simple models. These heavyweight UML tools usually contain large amounts of functionality, which can lead to usability problems and poor productivity. Lightweight web-based UML tools are designed to support quick and easy production of models. The use of the web has added benefits including portability and group collaboration. This report discusses lightweight groupware UML case tools and presents a lightweight web-based class diagram editor that requires no web browser extensions.

Publishing Information

A later version of this paper was included in the Proceedings of the 4th Australasian User Interface Conference, Adelaide, South Australia, February 2003; Conferences in Research and Practice in Information Technology, Vol 18.

Author Information

Dan Mackay was a BSc(Hons) student in Computer Science at Victoria University of Wellington in 2002. Robert Biddle and James Noble are members of the academic staff in the School of Mathematical and Computing Sciences at Victoria University of Wellington.

A Lightweight Web-Based Case Tool for UML Class Diagrams

Daniel Mackay **James Noble** **Robert Biddle**

School of Mathematics and Computer Science,
Victoria University of Wellington,
Wellington, New Zealand,
Email: kjx@mcs.vuw.ac.nz robert@mcs.vuw.ac.nz

Abstract

Many existing UML tools are far too heavyweight for quickly producing a few simple models. These heavyweight UML tools usually contain large amounts of functionality, which can lead to usability problems and poor productivity. Lightweight web-based UML tools are designed to support quick and easy production of models. The use of the web has added benefits including portability and group collaboration. This report discusses lightweight groupware UML case tools and presents a lightweight web-based class diagram editor that requires no web browser extensions.

Keywords: CASE Tools, Web Services, Groupware, UML.

1 Introduction

Nowadays the effort to produce large scale software can often result in the production of a "Big Ball of Mud"(Foote & Yoder 1999). Software engineers can always do with tools to lend them a helping hand, especially in the early stages of development. Computer Aided Software Engineering (CASE) tools can provide an excellent starting point for program modeling, but due to the overheads involved with some of the existing tools, case tools can be off-putting for smaller companies or projects.

A key part of the problem is that a lot of existing case tools, such as Rational Rose (Quatrani & Booch 1999), include a multitude of functionality to manipulate the diagrams produced. These *heavyweight* case tools are well suited to producing vastly complex UML (Unified Modeling Language) (Object Management Group 2000) diagrams but often require investing a considerable amount of time and effort into learning all the available functionality and how to apply it.

For example, consider a software engineer needing to quickly produce some simple class diagrams. Using a time consuming heavyweight tool would probably not be a viable option as it would be too slow to set up and produce the diagrams. This set up overhead plus the complexity of heavyweight case tools are some of the reasons why tools of this caliber are avoided by practicing engineers (Iivari 1996, Coplien 1994). Valuable case tools should not only support productive use but also collaboration from different locations. It would obviously be beneficial to be able to collaborate via tools whether at home, off-site, or even in a different country.

There are already web-based class diagram tools in use, however they use applets, plug-ins, or client side scripting. Although these class diagram tools support team collaboration, they restrict use of the tool to browsers supporting applets or plug-ins, and situations where such extensions may be used.

This paper first looks at solutions to this problem, and illustrates them with *NutCASE*, a simple lightweight case tool that supports manipulation of UML class diagrams. The major contribution is that this tool shows how such diagrams can be presented, maintained, and allow interaction, all without requiring any extensions to a web browser. We then discuss the technologies used during implementation, identify some related work, and present our conclusions.

2 Lightweight Tools

In order to aid with the early stages of development the modeling techniques used should ideally be as easy and productive as pen and paper. We need to have tools consisting of the core functionality of similar heavyweight tools without the overhead that accompanies them. The use of the web for running these (and other tools) has been steadily increasing as a means of distributed communication. There are many benefits from this that we can leverage.

The key difference between heavyweight tools and lightweight tools is the amount of functionality they provide. Heavyweight tools offer a much functionality that relates to the problem domain, but the functionality can be overwhelming, and using them can be tiresome as you have to play around with specific details (whether you may want to or not) just to complete a simple task.

The key attribute that identifies a lightweight tool is *essential functionality*. Functionality is only included out of necessity, not for the want of having more bells and whistles. The functionality included should not restrict the users from carrying out common tasks, and should not clutter the user interface or present the user with uncountable options.

The Web offers many advantages as a communication medium, such as being widely distributed, supporting group collaboration, and allowing portable access from a variety of platforms.

By being distributed case tools are widely available to many people. By supporting group collaboration multiple people are able to have input to the design of software. Lastly, by being portable case tools are not only able to be viewed by a web browser on a PC but also by personal digital assistants and even cellular phones. These are the reasons why lightweight web-based services are successful, such as *hotmail.com* for e-mail service, and the Wiki Wiki Web *c2.com/cgi/wiki* for collaborative work. The services must be simplified, but the advantages are significant. These advantages also make the web an ex-

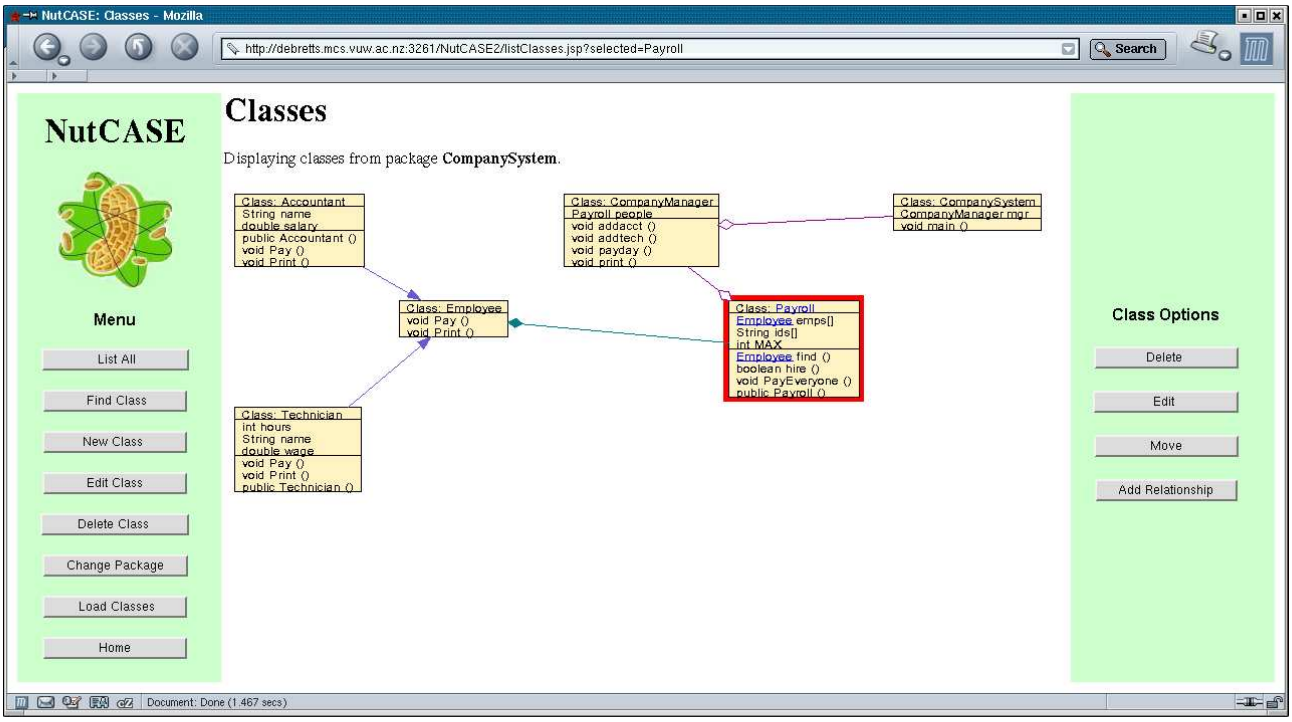

Figure 1: NutCASE: Displaying the Classes within a Package

cellent platform to support the use of lightweight case tools.

3 NutCASE

To demonstrate the feasibility of this approach we designed a lightweight web-based class diagram editor called NutCASE. NutCASE offers an environment in which you can manipulate UML class diagrams via the web. The functionality that it provides includes creation, deletion, editing, and searching of classes. Along with these operations the classes are divided up into packages in order to display only the classes of interest at one time.

NutCASE attempts to provide the quick and productive approach of pen and paper techniques, yet excludes most of the disadvantages that arise from using them. For example diagrams are able to be manipulated and edited without having to redraw the entire model by hand. Classes are also laid out automatically so that if a class is added is will be added to the current diagram causing minimal disruption. Another benefit is the searching of classes. Compare wading through hundreds of pieces of paper containing class descriptions as opposed to simply requesting to view it.

3.1 User Interface

The interface of most interest is the main class diagram screen as show in Figure 1. This screen displays all classes belonging to the current package. On the left side of the screen is the main menu which is present on every screen. These functions include displaying all classes in the current package, finding a class (and also its package), adding a new class to the current package, editing any class, deleting any class, and changing the current package. NutCASE also includes a tool for reading in classes from a file.

The center of the screen in Figure 1 shows the actual class diagram, and signals which package the classes belong to. If a class has been selected with the mouse then it is highlighted with a red box and a class options menu sidebar appears on the right hand side of the screen. When a class is selected the non-primitive object types are highlighted in a similar fashion to that of a hyperlink. This is an affordance to signal that clicking on these hyperlinks will take you to a page containing the definition for the highlighted object (see Figure 2).

The class options sidebar only appears when one of the classes is currently selected. These options allow the user to manipulate the selected class. The user can either delete, edit, move it around on the screen, or add a relationship to one of the classes in the same package.

Once the user has chosen to edit a class they will come to the screen shown in Figure 2. The package to which it belongs to is shown near the top of the page, and can be changed by selecting one of the other packages from the drop down menu. The edit page combines three functionalities into one. Each existing attribute/method/relationship can either be altered or deleted altogether. Two empty spaces are provided so that the user may add new attributes/methods/relationships. This combination of functionality minimizes the number of interfaces needed and speeds up use at the same time. Adding a new relationship is done by selecting the type from one drop down menu and selecting the class from another drop down menu. Having these options means that only valid relationship types can be used and they can only point to new valid classes.

If any deletions have been selected a confirmation dialog will be presented to double check that each item selected for deletion really should be removed. After this dialog is finished (or after changes are submitted) the new version of the class is displayed.

The two figures in fact show most of the user interface for the tool. The interaction simply involves ammending the model and reviewing the diagram. There is also a screen that shows only the diagram, in a form suitable for copying to a document.

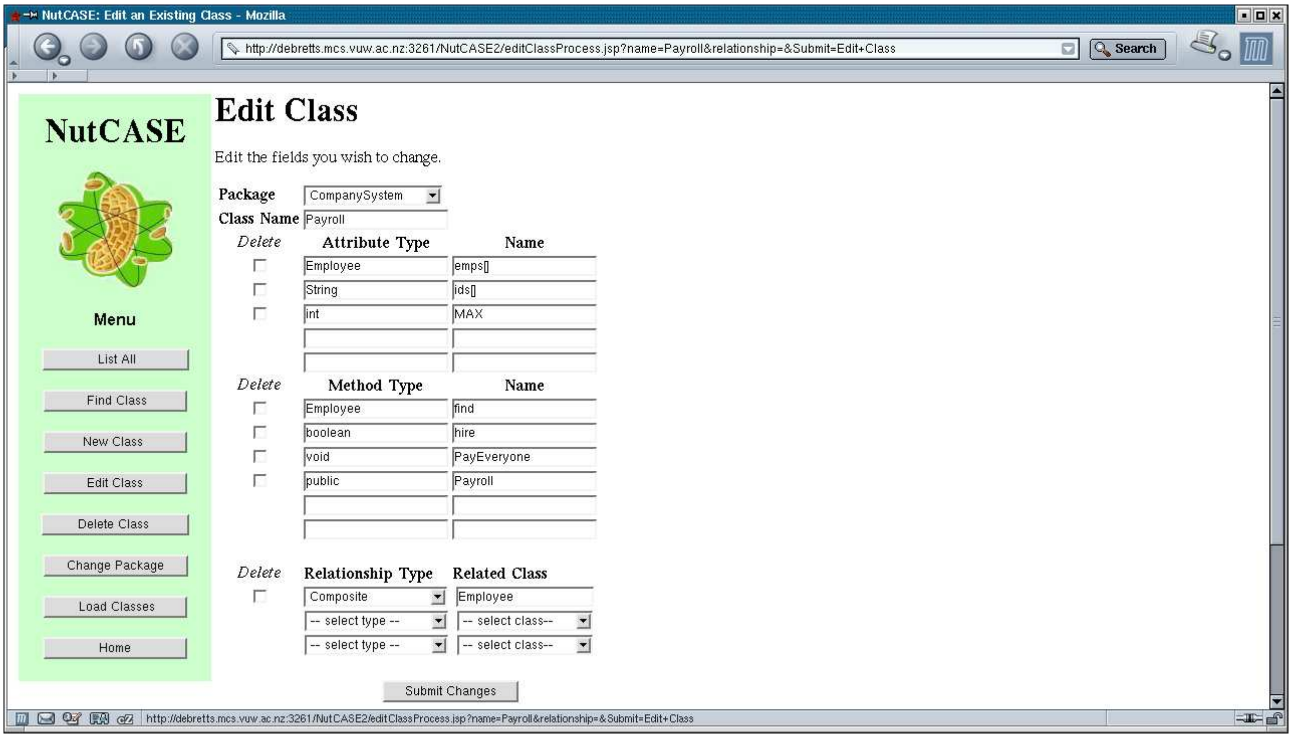

Figure 2: NutCASE: The Interface for Editing a Class

3.2 Automatic Class Generation

In order to decrease the time taken to rapidly produce class diagrams we introduced the notion of automatic class generation. Usually when creating a class diagram using user defined types, one would have to create a definition for the class type either before or after it is used (assuming you even remember to do so). When creating a class in NutCASE the types entered are checked for existence. If the type is not already defined as a class and is not a primitive type, then a skeleton class is generated to represent this new type. Once the original class has been created, any new class types that have just been created are then able to be edited to add more detail.

4 Implementation

There are a number of technologies behind NutCASE such as HTML, JSP(Java ServerPages), JavaBeans, and the MySQL database management system. JSP is a server side scripting language that processes the requests by calling functions defined in the JavaBeans. In response to the request the JavaBeans query/update the MySQL database and the server sends back the HTML to the browser. The native scripting language for JSP pages is based on the Java programming language. Because all JSP pages are compiled into Java Servlets, JSP pages have all of the benefits of Java technology, including robust memory management and security.

4.1 Server Side Storage

One disadvantage of using JSP is that the requests have to be sent away processed and sent back, however the only information that needs to be sent to and from the server is the requests and responses. A crucial advantage of this is that the only technology needed to use NutCASE is a device that displays HTML. Server side storage relies on the server doing the processing, and the browser need only display the results from its requests.

The speed of requests is therefore dependent on the internet connection, as opposed to the client processor (e.g. applets). Structuring web-based services greatly increases their portability making them widely available to access and use. Use of applets on the other hand restricts users to only those having compatible browsers, and to situations where such extentions may be installed and used.

By storing the implementation of NutCASE on the server we are separating the content from the presentation. This isolation means that the implementation can be maintained independently of the HTML, and likewise the layout of the information displayed can be altered without affecting the implementation.

4.2 GIF Generation

Whenever classes are to be displayed they are done so as a GIF (Graphics Interchange Format) image. Due to the nature of GIFs images with flat colour (e.g. blank space) can be compressed, and large images can be interlaced by the browser when loaded. This means the image can be loaded at a poor resolution to get an overview then detail can be rendered into the image. Producing these class diagram GIFs is done by querying the MySQL database, retrieving the class information needed and drawing pictures of this class information in a Java image. This image is then converted into a GIF using a purpose built Java class. The name of the file is embedded within the HTML so the browser simply displays the image, and the server takes care of the image generation.

The other type of image we considered using was SVG (*Scalable Vector Graphics*). SVG uses an XML based specification to define images (Duignan et al. 2002). To change an image being displayed the SVG needs to simply be altered for the changes to take affect. This type of functionality would suit intensive diagram manipulations. However, SVG requires a plug-in to be downloaded and needs to be accompanied by a hefty amount of client side scripting to manipulate the images. Having to use client side scripting would defeat the purpose of using server side storage and greatly reduce portability.

4.3 Image Maps

All the class diagram interactions in NutCASE use client side image maps. The image maps, combined with server side storage, perform the same tasks that are accomplished with the use of applets and plug-ins in other case tools. Each time a GIF is generated a meta-file is also generated containing layout information about the class(s). This meta-information is used to create an image map which is embedded within HTML. When NutCASE changes state, the picture is re-generated and a new image map is dynamically embedded within the HTML.

The GIF generation and the image map are the keys to the ability to present class diagrams and allow interaction without any browser extensions. They allow interactive images that are compatible with almost any web browser and without any special setup.

4.4 Class Layout

Each time a class diagram is to be generated all classes belonging to the current package are loaded into a layout manager. As each class is loaded the dimensions of the class are determined. After all the classes have been loaded the maximum width and height over all the classes are determined and the display area on the screen is divided up into a grid containing cells equal to this maximum width and height. The classes are then arranged in the grid in an alternating fashion (like black squares on a chess board) to reduce clutter, yet still making use of available space. This layout information is then written to the meta-file.

4.5 Reverse Engineering

NutCASE has rudimentary support for reverse engineering. Classes can be specified in a file and loaded into the database with a click of a button. As it stands the file has to be in the correct format to be read, but this is planned to be extended to read in raw Java source code classes or packages. These capabilties will also allow the tool to be useful for documenting existing software.

5 Related Tools

There are a number of other similar lightweight web-based Case tools that have we have been working on, including a sequence diagram tool and a use case tool.

One of the first lightweight web-based tools we built was UKase (Biddle et al. 2002). UKase was developed to support essential use case models. It is written in Perl script and CGI (Common Gateway Interface) script and uses HTML for displaying the information in tables. It was originally intended to be used as a recording tool, but was soon found useful for the development of the use cases themselves. For organisational purposes the use cases can be grouped either by a package, or by actor. As well as organising use cases UKase can also list them in a number of ways including by name, priority, actor, or package. However, unlike NutCASE, UKase originally had no support for any kind of graphical diagram editing.

Seek is a lightweight web-based tool that supports manipulation of UML sequence diagrams (Khaled et al. 2002). Its user interface is comprised of HTML image hyperlinks and forms. As with NutCASE, any computation is done server side using JSP and JavaBeans, so browsers need only display HTML to use the tool. While Seek does provide a graphical display it is quite constrained based on the heavy use of HTML tables.

All these tools are functional and have all been used in real development, but they are still research prototypes and subject to further work. One area of our ongoing work is integrating such tools to allow simple interoperability. We have conducted usability studies on all the tools, including NutCASE, as part of our development process. The studies confirm that the tools do work as we intend, and also allow us to improve the tools as we continue our research..

6 Conclusion

In this paper we have presented NutCASE, a lightweight web-based UML class diagram editor. NutCASE is lightweight because of the inclusion of essential functionality without excessive extra functionality its device independent implementation. We expect the restricted set of functionality offered will lead to easier learning of the tool and quicker production of actual class diagrams.

Due to the lightweight implementation and nature of the web, NutCASE supports group collaboration of class diagrams. Because it is a web service using server side storage, NutCASE is highly portable and is capable of being used on any device able to browse HTML, without needing to download or run applets. We believe NutCASE is the first web-based tool to provide editing of UML diagrams that can be controlled by the user without using plug-ins or applets.

As proof of concept designing NutCASE encourages use of lightweight case tools for all the above reasons. It offers improvements over then pen and paper modeling techniques, and draws designers to use such UML modeling techniques whereas before the process might have been too lengthy or complex with more heavyweight tools.

References

Biddle, R., Noble, J. & Tempero, E. (2002), Lightweight Web-Based Tools for Usage-Centered and Object-Oriented Design.

Coplien, J. (1994), A Generative Development-Process Pattern Language. *in* Pattern Languages of Program Design.

Duignan, M., Biddle, R., & Tempero, E. (2003), Evaluating Scalable Vector Graphics for use in Software Visualisation, *in* Proceedings of The Australasian Symposium on Information Visualisation, Adelaide, 2003. Conferences in Research and Practice in Information Technology, Vol 24. Tim Pattison and Bruce Thomas, Eds.

Foote, B. & Yoder, J. (1999), Big Ball of Mud *in* Fouth Conference on Pattern Languages of Program Design.

Iivari, J. (1996), Why are CASE tools not used *in* Vol. 39, No. 10, Communications of the ACM 1996.

Khaled, R., Mackay, D., Biddle, R. & Noble, J. (2002), A Lightweight Web-Based Case Tool for Sequence Diagrams, *in* 'Proceedings SIGCHI-NZ Symposium on Computer-Human Interaction', Hamilton, New Zealand, pp. 55-60.

Object Management Group. (2000) *Unified Modeling Language (UML) 1.3 specification*. Object Management Group.

Quatrani, T. & Booch, G. (1999) *Visual Modeling with Rational Rose 2000 and UML*. Addison-Wesley.