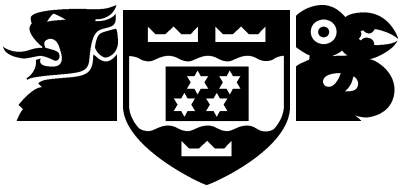
VICTORIA UNIVERSITY OF WELLINGTON
Te Whare Wananga o te Upoko o te Ika a Maui



School of Mathematical and Computing Sciences
Computer Science

A Lightweight Web-Based Case Tool for Sequence Diagrams

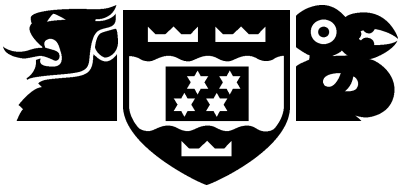
Rilla Khaled, Dan Mackay, Robert Biddle, James Noble, and Ewan Tempero

Technical Report CS-TR-02/10
May 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 Sequence Diagrams

Rilla Khaled, Dan Mackay, Robert Biddle, James Noble, and Ewan Tempero

Technical Report CS-TR-02/10
May 2002

Abstract

This paper presents a simple lightweight Case tool to support manipulation of UML sequence diagrams. The tool is intended to have high usability to facilitate communication and discussion in the early part of the design process, where sequence diagrams are frequently used to explore and explain object interaction. The tool is web-based to maximise accessibility and use by teams, and uses a simple yet effective image tiling technology to allow manipulation of the diagrams from any web browser.

Publishing Information

This paper was presented at the Computer-Human-Interaction Conference (CHINZ) in Hamilton, New Zealand, 2002.

Author Information

Rilla Khaled and Dan Mackay are graduate students in the School of Mathematical and Computing Sciences and the School of Information Management at Victoria University of Wellington. Robert Biddle and James Noble are members of the academic staff at Victoria University of Wellington. Ewan Tempero is a member of the academic staff of the Department of Computer Science at the University of Auckland.

A Lightweight Web-Based Case Tool for Sequence Diagrams

Rilla Khaled, Dan Mackay, Robert Biddle, James Noble
School of Mathematical and Computing Sciences
Victoria University of Wellington
Wellington, New Zealand
{rkhaled,dmackay,robert,kjx}@mcs.vuw.ac.nz

Ewan Tempero
Department of Computer Science
University of Auckland
Auckland, New Zealand
ewan@cs.auckland.ac.nz

ABSTRACT

This paper presents a simple lightweight Case tool to support manipulation of UML sequence diagrams. The tool is intended to have high usability to facilitate communication and discussion in the early part of the design process, where sequence diagrams are frequently used to explore and explain object interaction. The tool is web-based to maximise accessibility and use by teams, and uses a simple yet effective image tiling technology to allow manipulation of the diagrams from any web browser.

Keywords

Case tools, world-wide-web, user-interfaces

INTRODUCTION

The promise of computer assisted software engineering tools (CASE tools) is to provide support for software development processes. One problem that can arise in providing this support is that processes and tools may become so complex that they become difficult to use. Such poor usability of Case tools has been found to be a reason that such tools are less frequently used than had originally been hoped [5].

This paper presents *Seek*, a simple lightweight Case tool that supports manipulation of sequence diagrams in the Unified Modeling Language (UML) [8] notation. A sample screen is shown in figure 1. The tool is intended to facilitate the early part of the design process, where sequence diagrams are frequently used to explore and explain object interaction. The tool is web-based to maximise accessibility and use by teams, and uses a simple yet effective image tiling technology to allow manipulation of the diagrams from any web browser.

The rest of the paper is as follows. In the next section we review the background for the development of the tool. In the third section we discuss the design of the tool and how it is used. We then discuss some initial development concerns about the tool, and the results of an evaluation. Finally, we present our conclusions.

BACKGROUND

Heavyweight Tools

Existing tools that support the creation of UML sequence diagrams, such as Rational Rose[9], tend to be heavyweight: they offer a multitude of functionalities that support the drawing of any sequence of events the user may want to represent. However, precisely because so much functionality is supported, it takes users some time to acquaint themselves with how the tools work.

While a heavyweight case tool is desirable from the perspective of being able to represent complex system interactions, and therefore complex UML diagrams, there are many users with the intention of representing simple system interactions only. In these cases, the overhead taken to learn how to use the tool is largely wasted, as they will never fully make use of the tool's available functionalities. Even when heavyweight tools have been learned, they frequently present the user with so much choice that they remain difficult to use. In particular, these tools often allow freedom to create complex diagrams that may be correct but are seldom of interest. In supporting such complexities, tools often entrust the responsibility of checking the diagrams for errors and conformance to UML standards to the user.

The lack of constraints therefore means that diagram creation is both more difficult than usually necessary, and also more likely to lead to invalid diagrams. This situation is especially bad for novice developers or analysts, or other stakeholders. We have frequently seen heavyweight tools require inappropriately arduous work, and result in diagrams that represent bizarre and impossible sequences of events (see figure 3).

Earlier Work

In earlier work of ours, we developed another lightweight sequence diagram case tool, Seeker, that was designed especially to be highly usable and responsive [2]. The original intended use of the tool was for rapid documentation of design walk-throughs, so that use cases that were being role-played could be rapidly and efficiently recorded.

This tool featured constraints that made the most common diagram structures very easy to specify, and the tool used colouring to show regular structure and to highlight discontinuities. We also found this led to uses in teaching and learning. For example, it represents the concepts of composition and collaboration by depicting how objects work together in a step-by-step manner.

An issue that came up repeatedly during the development of Seeker was the need to carefully manage useful extra functionality while maintaining high usability.

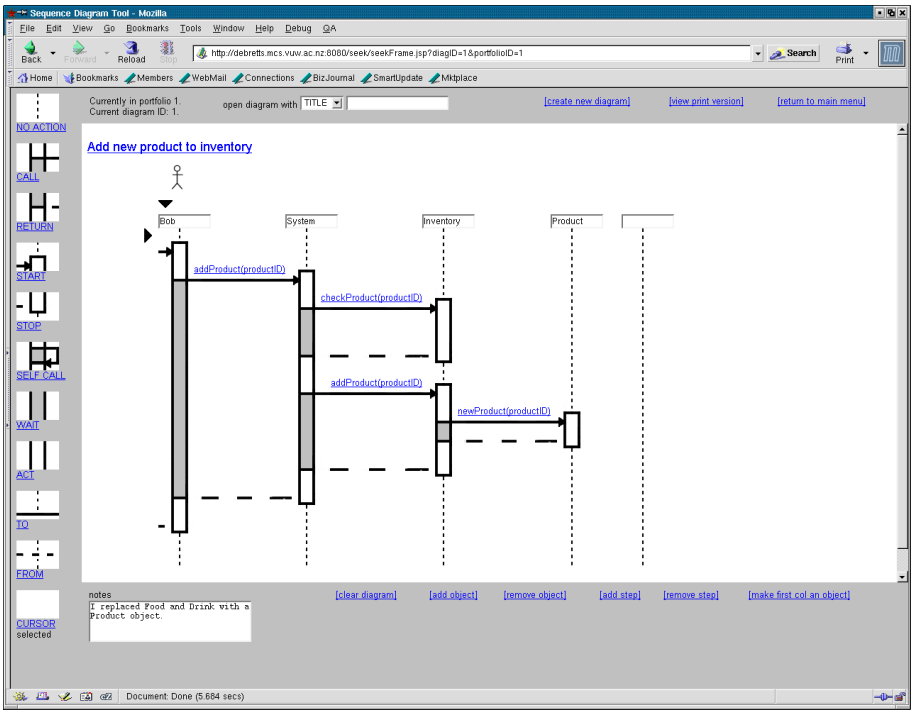


Figure 1: The main screen of Seek, showing a sequence diagram and facilities for editing and manipulation of the diagram. The diagram shows a use case, beginning with an actor at the left, and showing the step-by-step interaction between the objects.

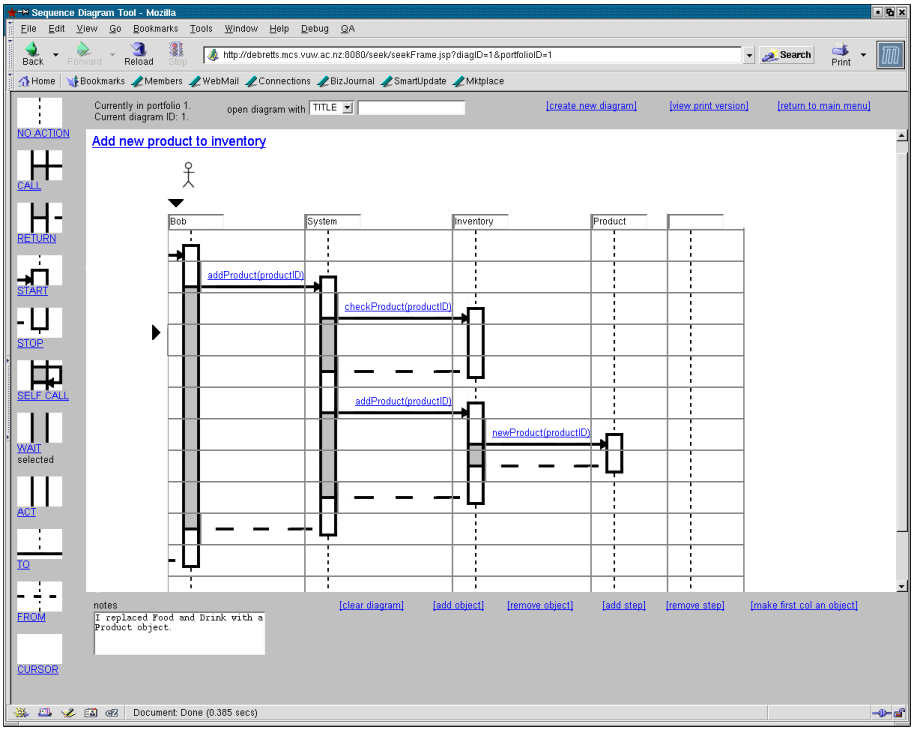


Figure 2: The same screen as above, but showing the image tiles that form the diagram, and that allow the diagram to be manipulated.

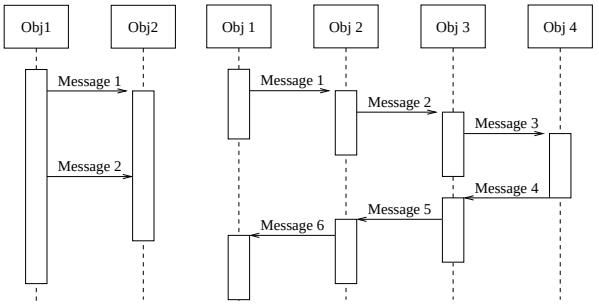


Figure 3: Two examples of unlikely sequence diagrams allowed by most tools: on the left the diagram shows a method activation spanning two calls, and on the right calls are shown where there should be returns.

Seeker was a stand-alone program. From our experience in working with designers in industry, the early phase of system development often involves collaboration and review of designs by different members of a team and by other stakeholders. This review often happens in different locations, and sometimes informally. We felt therefore that a tool to support this early phase of design would have advantages if, as well as being simple and lightweight, it could be as ubiquitous as a web page accessible from any web browser. Web-based case tools for drawing sequence diagrams certainly do exist, but in order to provide interfaces similar to conventional heavyweight case tools they have typically been implemented using Java applets, and are therefore less accessible to users who have slow connections, old computers or old browsers. Furthermore, they also suffer from the same sorts of usability problems as heavyweight case tools.

DESIGN

Technology

We set out to design a lightweight tool to support the creation of sequence diagrams using web technology. The challenge that we faced was that web support for interaction with diagrams is either very limited, or involves applets or plug-ins that were incompatible with ubiquitous web access. We decided to accomplish what we could using the very limited interaction available to any browser.

The user interface technology consists simply of HTML pages and forms, using hyper-links with images. Our implementation uses Java Server Pages (JSP) together with JSP JavaBeans [10, 1]. The JSP is used to respond to form requests and generate the HTML pages; a MySQL database is used to store the state between displays. This structure is depicted in figure 4.

The tool allows diagrams to be created, edited or deleted, and several diagrams can be grouped together as "portfolios". Seek also supports simple portfolio management activities. Seek lends itself well to use in teamwork situations as a team may own one or several portfolios, the diagrams are accessible from anywhere, and a simple form of diagram locking is provided.

This all achieves advantages similar to those of the earlier tool in supporting the manipulation of sequence diagrams

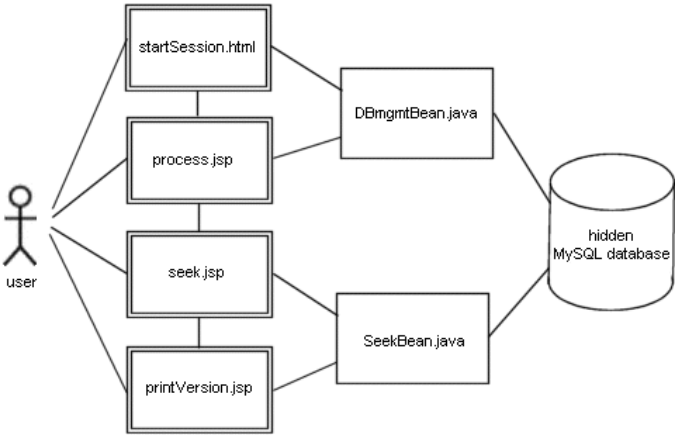


Figure 4: The architecture of Seek.

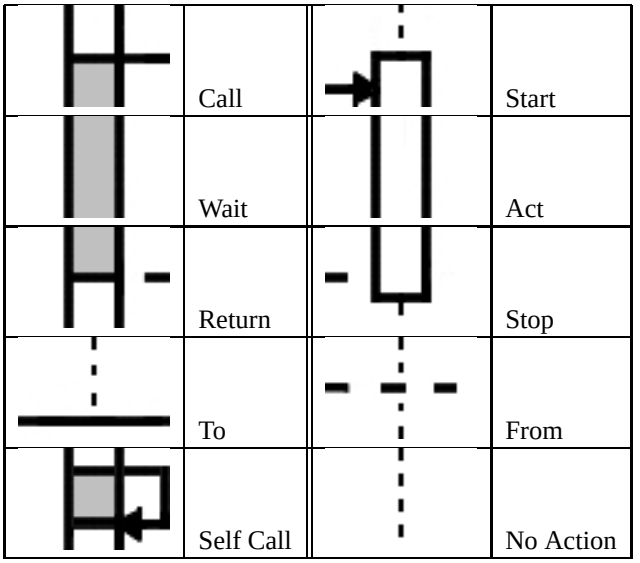


Figure 5: The images used to created a tiled sequence diagram: each image matches a particular state or action in a thread of execution.

with usability and responsiveness. The advantage of its web-based but applet-free approach is that all that is needed to make use of the tool is an ordinary web browser; consequently, anyone anywhere with access to an internet connection can make use of it.

An unusual aspect of our implementation is how we allow effective interaction with the limited support available in common browsers. We represent a sequence diagram as a two dimensional grid of image tiles, as shown in figure 2. Each tile is displayed using one of a set of images that depict each of the possible action states within a sequence diagram. The set of tiles is shown in figure 5. When combined in correct configurations, the small images collectively represent a UML sequence diagram.

The set of images is displayed as a palette at the left on the tool interface. The sequence diagram is created by repeatedly selecting an image by clicking on a palette image, and

then clicking in the grid where that image should be displayed. The clicking is in fact activating hyper-links, which cause the state of the diagram to be changed and then redisplayed. The images are very small and cached at the browser, so the redisplay is quick.

The simplicity and responsiveness of the image tiling technology supports the lightweight nature of the tool. A further consequence of drawing diagrams by building them out of a limited number of small "action" images is that the creation of correct diagrams is partially enforced, because after a short time of using the tool, it becomes evident which actions should be selected to be used above, below and on either side of other actions, due to how they correctly "fit" together. To further enforce creation of correct diagrams, the tool does simple error checking, and reports errors by pointing with small red arrows at the places where "illegal" diagram configurations exist. Enforced correctness of diagrams may help novice programmers better understand composition, collaboration, and object interaction.

Seek is limited in what it can draw: it cannot represent parallel system calls, and right-to-left system calls. While the tool could easily be modified to handle right-to-left system calls, the intended use for the tool was not to represent complex scenarios but rather standard scenarios, which it can indeed depict. Additionally, it does not specify return types for methods — while it could be easily modified to do so, displaying the return types could lead to a cluttered diagram and furthermore, specification of return types is not a UML convention.

Development of Seek

Although our design of Seek was influenced by the technology of the web and our image tiling approach, the actual interaction was designed as a separate process, once we had explored the technology. We used Usage-Centred Design (UCD) [4], and focused on task accomplishment. We analysed potential user tasks, and then for each of them, we created an essential use case (EUC). These EUCs were made and managed using Ukase, another lightweight web-based use case tool we developed in earlier work [3].

How to use Seek

Seek's functionality can be grouped to lie in two areas: diagram editing, and portfolio and diagram management.

Diagram editing is done from Seek, which contains the drawing grid for the diagram. To make a change to the drawing grid, which initially contains the dotted de-activated lifelines of 8 objects, the user must first select an action from the sidebar on the left-hand side by clicking on it. In order to apply the tool to a certain grid square, the user must then click in the relevant grid square. Clicking in the drawing grid results in an automatic update and save to the database records for the diagram, and a screen refresh so that the square clicked in will now contain whatever image the action happens to be. In other words, Seek has a concept of a "cursor" that gets updated to point to the last place where the user intended to apply an action, and then when the diagram refreshes to show a change, the change is made to wherever the cursor was last pointing.

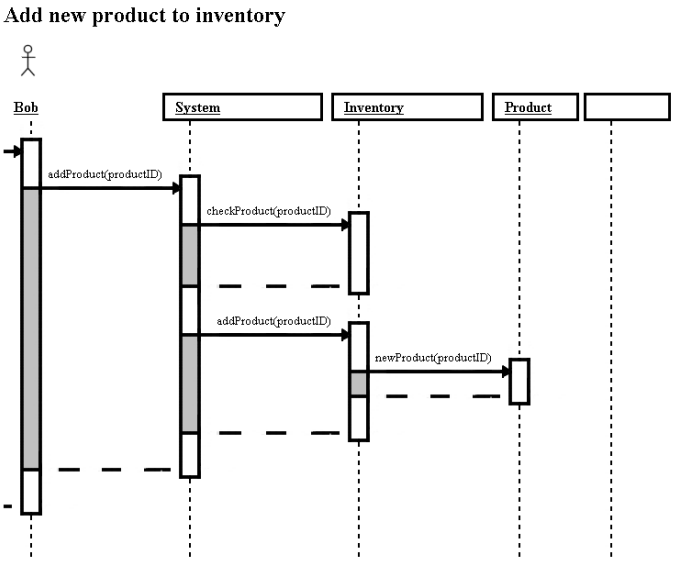


Figure 6: An image generated for printing based on the diagram created as shown in the earlier figures.

The grid update method changes some grid contents automatically by employing a "smart-tiling" policy — if a start image is already contained above a stop image that the user has just added to the diagram, then as long as there are blank grid spaces between them, the blanks are replaced with act images. The reverse case also causes an automatic update, i.e. if a start image is added directly above a stop image. The same sort of thing is done when a return is added below a call (the space between them is changed to wait images) or if a call is added above a return. Likewise, to lines are drawn between a call image in line with a start image (to the right) and from lines are drawn between a stop image and a return image (to the left).

The diagram features many embedded facilities to support editing. To add and remove steps or objects into the diagram, the user can either just select the appropriate link from the bottom toolbar, for example "add step", "delete object", and so on to add or delete steps or objects from the last position the cursor was stationed at. If they want to change the position of the cursor before adding steps, without carrying out an action, they can select the "move cursor" tool and click where they want the cursor to now point to. From this page, objects can also be given names (by filling in the form boxes at the top of the page), method names can be changed (by clicking on the hyper-links that say `call()`), diagrams can be given titles, and also have "notes" written about them.

There is also a link to display a non-interactive representation of the diagram, as shown in figure 6. This allows the user to print or capture a diagram suitable for inclusion in other documents.

Suggested sequence of steps for task two

- enter "Add new product to inventory" into the "Title of diagram to open" textbox at the top of the page.
- click on the "move cursor" image/link in the lefthand-side toolbar
- click anywhere on the food column
- click on the "remove object" button at the bottom of the page
- click anywhere on the drinks column
- click on the "remove object" button at the bottom of the page
- click on the "act" image/link in the lefthand-side toolbar
- click in all the grid squares in the Inventory column that have red arrows pointing to them
- click on the Bob-to-System call() link and enter addProduct(productID) into the pop up window
- click on the System-to-Inventory call() link and enter checkProduct(productID)
- click on the "stop" image/link in the lefthand-side toolbar
- click under the Inventory activation cell
- click on the "return" image/link in the lefthand-side toolbar.
- click on the grid square in the System column adjacent to Inventory's "stop"
- click on the "call" image/link in the lefthand-side toolbar
- click on the grid square under System's return
- enter the method name addProduct(productID)
- click on the "start" image/link in the lefthand-side toolbar
- click in the grid square adjacent to the System call and under Inventory's column
- click on the "call" image/link in the lefthand-side toolbar
- click under the Inventory activation, enter the method name new Product(productID)
- in the column next to Inventory, enter the name "Product"
- click on the "start" image/link in the lefthand-side toolbar
- click in the Product column, adjacent to the Inventory call
- click on the "stop" image/link in the lefthand-side toolbar
- click in the Product column, under its activation
- click on the "return" image/link in the lefthand-side toolbar
- click in the Inventory column, adjacent to the Product "stop"

Figure 7: Notes made while planning for heuristic evaluation, showing the steps necessary to create a diagram such as shown in the earlier figures.

EVALUATION

Initial concerns during development

Smart-tiling

Initially we were slightly dubious as to the usability of the image tiling technology — we could foresee that users might quickly tire of the process of having to select actions from the sidebar and then click in the drawing grid in order to draw just part of an object activation.

The smart-tiling functionality was added for this purpose. One of the problems with the smart-tiling was working out just how much to automate — it is important for users to feel that they are in control and guessing what they want to do may in fact create more work if the guess was incorrect. The smart-tiling that was finally implemented is fairly unobtrusive but still greatly reduces "total clicks taken" to draw diagrams. However, having implemented the smart-tiling ourselves, while experimenting with the tool and drawing diagrams, we were able to maximise the effects of the smart-tiling simply because we knew exactly which sequence of events would trigger it. Due to its unobtrusiveness, we were aware that it would take users a while to familiarise themselves with its behaviour.

Undo

Some functionality that would have lessened the effects of incorrectly guessing a user's intentions, and would have created a better working environment for the user in general, would be an "undo". As Seek is a prototype, it does not currently feature one, but any future versions of it will feature an undo, as it is a function generally available in drawing programs, and furthermore, it supports the idea of exploratory design (or rather, exploratory un-design).

HTML conformance and diagram quality

We thought that it was important that the diagrams always conform to UML conventions. However, due to the diagrams actually being formed out of smaller images arranged in a two-dimensional table, various HTML tricks had to be employed to get the images stretching whenever table cells became of uneven widths due to their contents. While in most browsers the HTML tricks work, in some they do not. With the aid of cascading style sheets and their capacity to positions elements either absolutely or relatively, this problem could probably be avoided. It must be kept in mind however, that while the style sheet technology is not new, certain older browsers do not support them completely.

Heuristic evaluation

In order to evaluate the usability of Seek, we carried out a heuristic evaluation to Nielsen's guidelines [6, 7].

Evaluators were given 3 tasks to complete, during which they were able to explore various features of the tool and inspect elements of its interface. The first task involved creating protfolios and diagrams completely from scratch. The scenario to be depicted consisted of a simple sequence of calls and returns between an actor and several objects. The second task involved retrieval of an existing diagram and making "correct" modifications to it by deleting objects, creating new objects, relabelling method calls and specifying new method calls. It also involved printing. The third task again consisted of modification of a diagram, but this time by potentially completely changing the contents in the centre of the drawing grid. In particular, it involved adding new objects between existing objects and producing a "correct" diagram.

One major criticism of the tool expressed by all evaluators was that there was no undo function, which broke the "user control" heuristic. This would be particularly annoying during the familiarisation phase with the tool as users would not know how to make use of features such as the smart-tiling, hence the process of creating a diagram would be much more arduous, and any mistake made along the way would just add to the perceived workload. As mentioned above, future plans for the tool do include implementation of an undo function, so this problem will be addressed.

Another criticism was that there was little feedback on whether changes to diagrams were being saved — in fact, changes are *always* saved as the contents of the drawing grid are written back to the database with every modification. Evaluators did eventually realise this, and went on to say that constant saving was not necessarily desirable as it meant that even mistakes got recorded. However, the flipside to any incorrect changes being recorded automatically is that corrections are also recorded automatically. It could be argued that while the auto-save may initially seem confusing, it actually supports ease of use and rapid diagram creation because the tool takes care saving and lets the user concentrate on fast diagram creation. Nonetheless, the concept of "saving" is one that users have come to expect, so if they were notified at the start of an editing session that all changes made to the

diagram are saved, and if a little more system feedback was given in applicable places, regarding whether the change had been successful, they would have more of a sense of closure with regards to the effects of their actions.

An additional problem that came out of the evaluation was that when method names became exceptionally long, the height of their table cell changed to accommodate the extra data — but as each cell was sized so that it would fit the image inside it, the change in cell height meant that there was now some white space between one row of images and the next, resulting in a confusing "broken" picture. As with the problem of horizontal image stretching, this is a problem that could be remedied by cascading style sheets and relative positioning.

The evaluators also mentioned that they thought it would be helpful if the red "error" arrows somehow explained what was wrong with the diagram. Since the red arrows are images, a simple way to provide error explanations would be to provide image captions for each arrow, where each caption could explain why the arrow was there. Alternatively, explanations of diagram incorrectness could be hyperlinked to the arrows, thus creating a sort of "error log".

Surprisingly none of the evaluators complained about having to click excessively in order to draw parts of sequences, so perhaps it was not as much of a problem as one might initially think (see figure 7). In fact, one evaluator stated that she thought the overall drawing concept was fairly intuitive after she had identified what the elements of each toolbar did and how to make changes to the diagram. As predicted, it took a while for most of the evaluators to discover the smart-tiling option, and one evaluator did not discover it at all. Online documentation of the system's functionality could fix this. In fact, through the use of carefully placed hyperlinks, it might be possible to tie in documentation for each diagram feature or function from each place where the functionality is triggered.

CONCLUSION

In this paper we have presented Seek, a lightweight web-based case tool for creating sequence diagrams. A major feature of Seek is its form of diagram representation and interaction: the image tiling technology. This technology has several advantages. Its simplicity creates a lightweight feel for the tool, and it works across nearly all web browsers. Furthermore, in combination with the error checking feature, it enforces creation of correct UML sequence diagrams, which in turn leads to better understanding by novices about how objects should interact.

Seek was developed with high usability and responsiveness in mind. However, being web-based, Seek has additional advantages: since diagrams can be accessed from anywhere, it is suitable for use in teamwork scenarios or any other "remote" work situation.

The basic philosophy behind using the tool seems to involve fair amounts of clicking. However, evaluation of the tool suggests this is not problematic: the lack of an undo function was more disturbing, and will be included in any future versions. Other improvements suggested were better explanation associated with red error arrows, and more feedback about when changes have successfully saved. We are now addressing these suggestions, and moving on to use the tool for demonstrations and more detailed evaluation.

REFERENCES

1. Bergsten, H. *JavaServer Pages*. O'Reilly & Associates, 2000.
2. Biddle, R. A lightweight case tool for learning OO design. In *Proceedings of Oopsla 2000 Educators Symposium* (2000), ACM, pp. 78–83.
3. Biddle, R., Noble, J., and Tempero, E. Supporting reusable use cases. In *Seventh International Conference on Software Reuse (2002)* (Apr. 2002), pp. 210–226.
4. Constantine, L. L., and Lockwood, L. A. D. Usage-centered engineering for web applications. *IEEE Software 19*, 2 (2002).
5. Iivari, J. Why are CASE tools not used? *Communications of the ACM 39*, 10 (Oct. 1996), 94–103.
6. Nielsen, J. Finding usability problems through heuristic evaluation. In *Proceedings ACM CHI'92 Conference (Monterey, CA, May 3-7), 373-380* (1992).
7. Nielsen, J. *Usability Engineering*. Academic Press, New York, 1992.
8. Object Management Group. *Unified Modeling Language (UML) 1.3 specification*. Object Management Group, 2000.
9. Quatrani, T., and Booch, G. *Visual Modeling with Rational Rose 2000 and UML*. Addison-Wesley, 1999.
10. Sun Microsystems Inc. Javaserver Pages White Paper. *java.sun.com* (2002).