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 Case Tool for Learning OO Design

Robert Biddle

Technical Report CS-TR-00/2
March 2000

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 Case Tool for Learning OO Design

Robert Biddle

Technical Report CS-TR-00/2
March 2000

Abstract

This paper presents a small CASE tool that uses sequence diagrams based on UML to support learning about OO design. The tool is language independent, and is designed to be highly usable and responsive. Although initially designed for use in team design situations, it is also directly useful in teaching about OO design, as a device for interactive presentation. Moreover, when used interactively, the tool seems to offer some of the same advantages of walkthroughs themselves, suggesting that such lightweight tools could have a role in the design process itself.

Publishing Information

Presented at the Educators Symposium at ACM Oopsla 2000 in Minneapolis, Minnesota, USA.

Author Information

Robert Biddle is a senior lecturer in Computer Science at Victoria University of Wellington, New Zealand.

A Lightweight CASE Tool for Learning OO Design

Robert Biddle
Computer Science
Victoria University of Wellington
Wellington, New Zealand
robert@mcs.vuw.ac.nz

Abstract

This paper presents a small CASE tool that uses sequence diagrams based on UML to support learning about OO design. The tool is language independent, and is designed to be highly usable and responsive. Although initially designed for use in team design situations, it is also directly useful in teaching about OO design, as a device for interactive presentation. Moreover, when used interactively, the tool seems to offer some of the same advantages of walkthroughs themselves, suggesting that such lightweight tools could have a role in the design process itself.

1 Introduction

Object-oriented software design is a process of synthesis, working from a variety of requirements and constraints to the creation of a new structure. This process is difficult, especially for beginners, and many approaches involve iterative and incremental techniques, whereby approximations are reviewed, amended, and refined. For example, consider "Responsibility-Driven Design" [8], where the first "exploratory" phase is an essential part of the process. The promise of CASE tools is explicit in the name: computer assisted software engineering. This paper presents a CASE tool designed to help learners with exploratory OO design.

The tool supports interactive drawing of diagrams based on the layout of the UML "sequence diagram" [5]. The diagrams themselves are simple, but the usefulness of the tool lies in high responsiveness, which allows the diagram to be extended or amended very rapidly. This responsiveness, in association with the nature of sequence diagrams, turns out to provide good support for learning the exploratory design process. The tool is language-independent, increasing its applicability.

This rest of this paper is as follows. Section 2 reviews the essential structure of UML sequence diagrams, and discusses their strengths. Section 3 then presents the CASE tool itself. Section 4 then explains how the tool can be used to support learning design processes. In section 5 there is a discussion of experience and attempts to extend the tool. Finally in section 6 some conclusions are presented.

Description of the research field

This work presented in this paper stems from my research work on software reuse, and especially work on tools to support software reuse. This work typically begins with an analysis of principles that underly software reuse and reusability (e.g. [3]), then uses these as a basis for developing a tool to provide better support (e.g. [2]), and then explores whether the tool in fact provides the support intended (we often use methods from the Human-Computer-Interaction literature such as [7]). This work often crosses over to involve issues in education, because so many issues in improving support for software engineering involve improving understanding of software design and its consequences. Although most of my work in tool support has been in the context of software reuse, the work described in this paper began in the context of education, and only later did I see that it related to software reuse as well.

2 Sequence Diagrams

The UML sequence diagram is one of UML's "interaction diagrams" Explaining the role of the sequence diagram, Fowler [5] says:

> *One of the hardest things to understand in an object-oriented program is the overall flow of control. A good design has lots of small methods in different classes, and at times it can be tricky to figure out the overall sequence of behaviour. You can end up looking at the code*

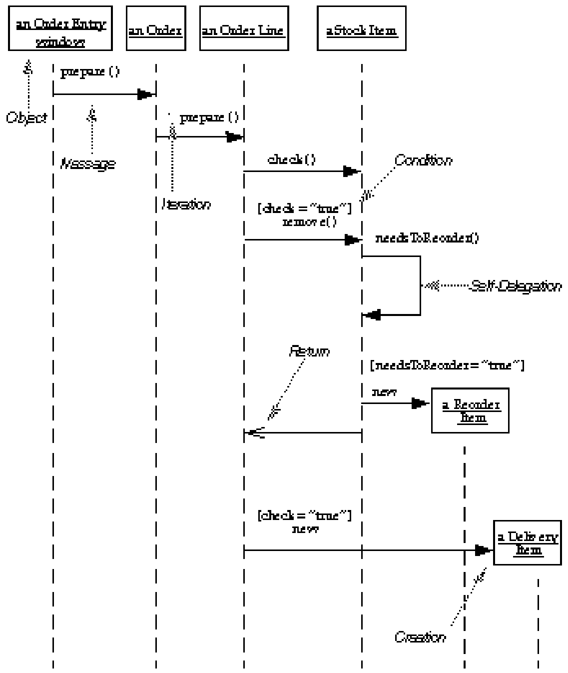

Figure 1: *An example of the UML sequence diagram (from Fowler [5]).*

> *trying to find the program. This is particularly true for those new to objects. Sequence diagrams help you to see that sequence.*

The essential layout of the sequence diagram is simple. The top of the diagram is a horizontal row of boxes, each representing an object. Beneath each box is a vertical "lifeline" representing activity involving the object. Where one object calls a method of another object, a horizontal line is drawn from the column of the calling object to the column of the responding object. When a method returns, a reversed line is drawn as appropriate.

There are other concepts that can be illustrated, including concurrency and delays, but the essential structure is the same.

The sequence diagram is a two dimensional diagram, structured around the interplay between the vertical and horizontal axes. The horizontal axis is a discrete object space which is not strictly ordered. The vertical axis represents time, flowing from the top to the bottom of the diagram, usually in discrete steps.

At a detail level, the sequence diagram is attractive because it allows several useful kinds of information to be displayed at the same time. Firstly, a small set of objects is introduced as playing roles together. While there is no strong ordering, they can be arranged in meaningful ways to make the roles easier to understand.

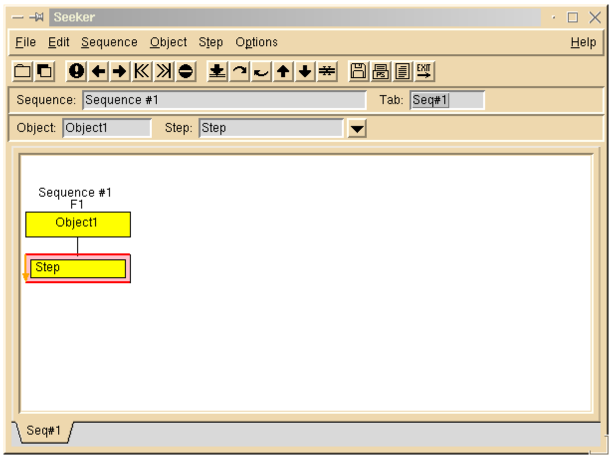

Figure 2: *The Seeker interface after startup.*

For example, they can be grouped, or placed in order of relevance depending on their significance in the diagram. Usually they are arranged so that most calls go from left to right, and returns from right to left.

Secondly, each vertical object lifeline, viewed by itself, can highlight the activity of the object in isolation. In an object-oriented system, encapsulation adds more significance to this view. Normally, object encapsulation assures that only an object itself should be be able to change its state. Therefore, the vertical line in a sequence diagram will normally highlight the only possible places where a change in object state may occur. By following any column in isolation, we should be able to track changes within the object associated with the column.

Thirdly, and most importantly, a sequence diagram highlights the sequence of execution. On the vertical axis, the diagram makes the order clear, and on the horizontal axis, the diagram highlights the distribution of execution between objects.

At a higher level, the attraction of the sequence diagram is that all these elements are brought together in a harmonious way: a set of objects, each with their own flow of access and state change, collaborating in the control sequence.

3 Seeker: A Sequence Diagram Case Tool

The Case tool Seeker was created to support the drawing of simple UML sequence diagrams. The name "Seeker" is a weak pun referring to the sequence diagrams. This section presents the tool itself, and briefly explains the design rationale.

The main interface to Seeker is shown in figure 2 as it looks when the application is started. There are menus, toolbars, and a sequence diagram, initially just a single object with a single "step". As the figure shows, there are deviations from the UML standard form, but

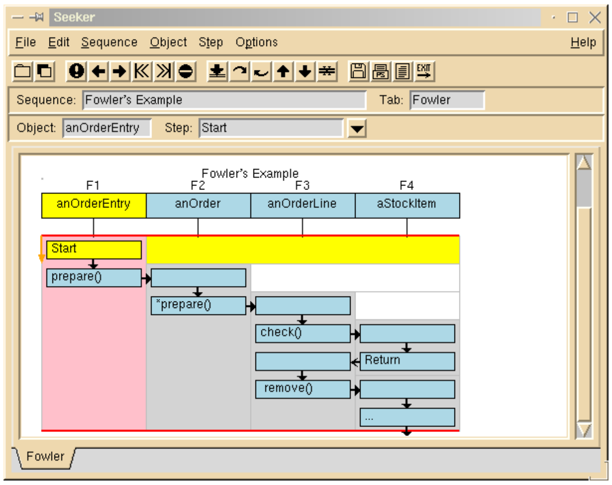

Figure 3: *The Seeker interface after creating a simple sequence diagram based on figure 1.*

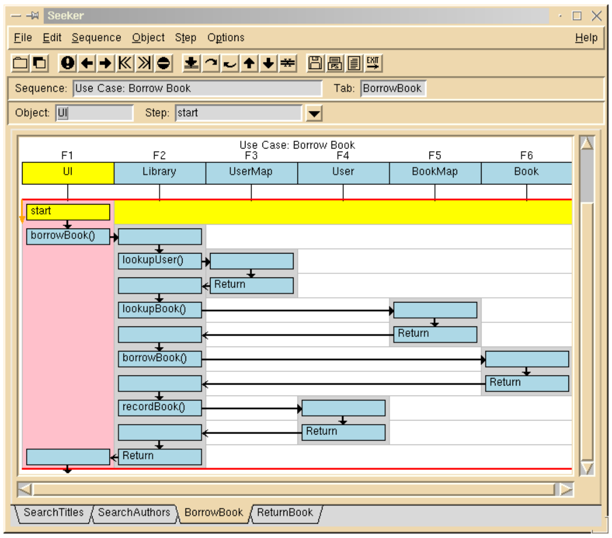

Figure 4: *The Seeker interface showing a portfolio of sequence diagrams.*

nothing truly significant, and simply reflecting experimentation in adapting the UML form to interactive use.

The sequence diagram can be extended by adding more objects, and by adding more steps to the sequence. A "step" is vertical step in the sequence, and can be a method call, a return, or a comment. Comment steps do not involve any transfer of control, but have proven useful in fleshing out design details. Figure 3 shows a sequence diagram after adding some other objects and several steps.

The sequence diagram highlights a "cursor" position identifying an object column and a step row. The diagram may be amended by modifying, deleting or inserting at the cursor position. The cursor can be repositioned by using keyboard arrows or by clicking the mouse. It is also possible to re-order the columns of the display by shifting objects and their associated lifelines to the left or right. This allows the user to manage the arrangement of the diagram, even as the distribution of intelligence evolves with the design. The overall diagram can extend beyond the screen vertically and horizontally, and scroll bars indicate the part of the diagram shown on screen. The visible portion can be moved using keyboard or mouse.

Seeker allows sequence diagrams to be saved to file and loaded, and to be printed. Moreover, several sequence diagrams can be active on the screen at once, and they can be saved and loaded together as a "portfolio". The screen shows tabs indicating and allowing selection of the different diagrams that make up the portfolio. Figure 4 shows a portfolio of four sequence diagrams, with the third tab showing that it is currently displayed. The order of the diagrams can easily be changed by shifting diagrams left or right, and diagrams may be deleted or inserted from other portfolios.

The menus of the application show the full functionality, so that simply browsing the menus gives a quick indication of the capabilities. For functionality that needs to accessible more directly, toolbar buttons and keyboard shortcuts are also available. For example, the "Enter" key inserts a new comment step at the current cursor position. The object columns are associated with keyboard "Function" keys. For example, the "F3" key inserts a new method call from the current object to the object in the third column. Similar keyboard shortcuts are available for all the ways of manipulating the diagram. The buttons on the toolbar present a similar set of shortcuts, making them quickly available if the user wishes to interact just by using the mouse.

The keyboard and toolbar shortcuts represent more than convenience afterthoughts. The main design rationale for the tool was high usability and responsiveness. Paper-prototype studies ([9]) and task analysis showed that the key requirement was that usable effects should be obtainable with minimal user interaction. For example, depiction of a complete method call should ideally require no more than a single keystroke or a single mouse click. This has been achieved, and other features were added on the same basis.

There are many existing CASE tools and tool suites that claim to support the OO design process, and most include a capability to support drawing sequence diagrams. However, such tools typically approach sequence diagram drawing in a way similar to that of more general drawing tools. They provide a drawing tool bar, and allow the user to draw boxes and lines to form a sequence diagram. This approach allows the creation of beautiful diagrams. However, there is a significant cost because the detail user interaction both consumes time and demands attention.

Poor usability has been cited as a key reason that some Case tools have been less successful than had been hoped [6]. The exploratory nature of design demands especially high usability so that the tool does not interfere with the design work. For these reasons, the high usability and responsiveness of Seeker seem to suggest it is a reasonable tool even though it has a narrow function. Of course, similar functionality could be integrated into other Case tools. However, even the comprehensive nature of many other Case tools and suites can work against them, if they are so large they require significant time to learn, and a powerful computer to run them. Seeker is lightweight, easy to learn, and easy on resources.

4 Using the Tool: Findings applicable to OO educators

Essentially, Seeker simply supports drawing of sequence diagrams, and so can be used whenever there is a need to generate such a diagram. However, the high usability and responsiveness make this tool especially useful in more interactive settings.

4.1 Rapid design documentation for review

The original intended use of the tool was for rapid documentation of design walkthroughs. I had been working with student design teams who were using the CRC (Class-Responsibility-Collaborator) card technique [1] along with role-play. In this approach, team members identify with a class or object, and interact with each other to walk through a use case. This method has several advantages, and allows rapid evaluation and development of design.

In my teaching, I often arrange several teams to work on the same design problem, they then present their designs to each other, and then we discuss differences and similarities. Especially in a short time frame, we often found it difficult to understand other designs, and even harder to compare them. We found it facilitated understanding to have teams actually present their walkthrough, rather than present a class diagram. While this did help, it was time-consuming, and it was still often difficult to recall and compare different design alternatives.

I realised then that a sequence diagram matched the structure of the role-play design walkthrough, and would thus be a good way of documenting the walkthrough for later comparison and analysis. I started by drawing sequence diagrams by hand while watching teams role-play. However, the drawings were necessarily crude and time-consuming to draw. I then created the tool specifically for this purpose: to rapidly document live design walkthroughs. I believe it serves this purpose well, but I soon realised it had other uses.

4.2 Early Teaching and Learning

The second use for the tool I found was in teaching. To some extent, of course, the main use described above also supports teaching. But in fact, the tool is also useful for teaching in a more direct way, and at an earlier point in teaching OO design. I teach some elements of OO design at quite an early point, midway through a first course (CS1) in programming. At this point, I am more interested in the students ability to understand a design, rather than their ability to create designs themselves. I introduce the ideas of composition and collaboration, and stress how programs are formed by objects working together.

Many students take in the general concepts of composition and collaboration, but still have difficulty understanding how it works in practice, especially when several objects are involved. To better support this topic, I first started to use sequence diagrams informally. As I explained a program, I quickly drew a fragment of a sequence diagram. Seeker makes this easier, and allows me to rapidly consider and present alternatives.

The sequence diagram approach illustrates the process of objects working together in a step-by-step way, making the distribution between objects clear, while keeping the execution sequence as context. Students find it illuminating to trace the sequence through method calls from object to object to object, following returns and later method calls. Understanding this can improve a weak understanding, limited to small parts of a program, to a more powerful understanding of how all the parts work together. The approach also leads to a better foundation for later design work, where the students undertake the team exercises described earlier. Of course, the very nature of the tool means that all this support is independent of any particular OO language.

I have also experimented in using the tool to illuminate recursion. Sequence diagrams can show self calls, method calls from an object to itself, by simply showing a line from a column back to the same column. This approach is a start to explaining recursion, but it conceals the nested structure of the method calls. The nature of recursion is suggested a little better by nested overlaid rectangles on the lifeline, as in Buschmann's diagrams [4] on which the UML sequence diagrams were based.

For teaching, however, we really want to be able to reveal the structure of the recursion, showing nested calls and returns clearly. I do this by creating several lifelines with the same object name, and explain that the same object plays several roles in the sequence. I can then illustrate recursion in a way similar to delegation, tracing the steps as an object makes a method call to another view of the same object, and so on, tracing return back the same way. This approach does rely on the students understanding that the several objects are all the same really, but with that proviso it does illuminate recursion well.

4.3 Direct design support

The CRC card approach, with team role-play and walkthroughs, is successful for several reasons. For one thing, it allows identification between team members and objects. This harnesses human and social perception, which fosters quick appreciation of object responsibilities, and distribution of intelligence. Another advantage is that the walkthrough process allows a design to be explored very quickly in the context of a use case, and changes and refinements can be both added and discarded rapidly. Somehow, the walkthrough allows us to *experience* a design.

As I experimented with early versions of the tool, I found myself doing something unexpected. I had intended the tool to be used to document live walkthroughs, but of course I didn't always test it that way. Instead, I just used it by myself, pretending I was watching a walkthrough. While doing this, however, I found myself considering the designs that I was documenting.

When I saw the sequence diagram appear on the screen, I sometimes was tempted to change the design and the diagram. The tool supported this well by allowing rapid entry and modification, as indeed this can happen in the middle of live walkthroughs. However, I was in fact starting to do the design task myself. The tool seemed to support the live walkthrough, even when I was making it up myself. With the aid of the tool, I was doing by myself what happens in team role-play. The phenomena that happens in role-play is a significant aid to design, and can be simulated with a tool such as this. The tool helped me experience the design, and that helped me explore design alternatives. This is the other reason for the name "Seeker": I found it helped me to seek better designs.

When working by oneself, it is obviously not possible to have the individual identification with objects and separation of concerns that plays a part in successful team CRC processes. However, I still found that the tool helped to some extent. The presentation of the object lifelines helps appreciate the different objects, and the layout of the sequence between objects seems to convey the distribution of intelligence.

While there have been attempts to support the CRC card technique in CASE tools, there has also been skepticism that any tool can match the lightweight requirements that are such a critical part of the CRC techniques. I had originally strongly agreed with such skepticism, but now feel there may yet be a role for lightweight CASE tools. I believe that Seeker gives some indication of how such tools might work.

5 Extensions

As I have used various prototype versions of the tool, I have felt a continual urge to tinker with the tool itself. The typical cycle has involved adding some feature that seemed useful, and later removing it because it seemed to interfere with the lightweight nature of the tool. There are several extensions that I am still experimenting with.

An extension I resisted for some time was the ability to work with sets of sequence diagrams, rather than single diagrams. Once implemented, I quickly decided to keep it because it clearly added to the usability. It supported the need to keep sequences related to use cases together, and allowed rapid review of several related sequences. This was so often desirable that doing without it seemed a pointless nuisance.

One new extension included a typical class browser, which included classes, objects, and methods. The browser was read-only, and merely presented a differently structured view of what was shown in the sequence diagram. The idea was that the browser permits a view that suggests the static class structure. Moreover, this structure could be exported as a text file, forming a text skeleton of comments for later implementation. I actually didn't find the browser much use, but I retained the ability to export the skeleton text to assist beginning implementation, or assist writing text descriptions of the design.

Several extensions have involved embellishment of the sequence diagram in some way. One such extension aimed specifically at supporting teaching. Even though the diagram is static, it does depict a sequence of method calls and returns over time, which means that for any cursor position, the current run-time stack can be determined. To assist learners, and help me make points in teaching, the current run-time stack is therefore always shown by highlighting the appropriate sections of active lifelines in a different colour. As the cursor is changed, the highlighting also changes automatically.

One aspect of sequence diagrams that I liked was the horizontal lines giving some indication of distribution of control. As sequence diagrams get longer, however, much of the diagram scrolls away. To keep some indication of distribution, I created a compressed version of the distribution at the top of the diagram. This is vertically aligned with the main diagram, and shows which objects call methods of other objects. It therefore allows a quick appreciation for how control is distributed, without using much vertical space. Other simple design metrics can also be presented.

Another new extension includes facilities to conceal and reveal steps or objects, and highlight the sequence of execution so far. This allows presentation of the sequence by progressive revelation, in a way similar to that provided in some presentation software. It also seems possible this kind of support could play a useful role in actual design work, by allowing designers to move from more abstract to more detailed views of a sequence diagram.

One weakness in sequence diagrams is also reflected in the tool. While the lifelines can highlight where the state of an object may change, there is little to indicate overall data-flow in the sequence. For example, if a method call results in the return of data which must then be passed as a parameter, then this is invisible in a sequence diagram, but possibly critical in understanding the overall design concept. This weakness is also reflected in CRC role-play and walkthroughs, where careful attention is required to be sure that objects work only with data that is actually available. In the tool, I have experimented with annotations to record data flow, but in complicated situations the diagrams can easily become cluttered. There remains a challenge is to provide useful support while still retaining the lightweight nature of the tool.

6 Conclusion

In this paper I have presented a language-independent OO design CASE tool based on UML sequence diagrams. The tool is aimed at supporting beginners by allowing rapid interactive documentation of design walkthroughs, and accomplishes this support by a high level of responsiveness and usability.

While the initial intended use for this tool was quite narrow, it also turns out that the tool is also very useful directly in teaching, and possibly in the early design process itself. Moreover, because the tool is aimed at lightweight support early in the OO design process, and because the diagrams used are based on UML, the tool does not interfere with later adoption of any full CASE tool or suite.

CASE tools are all about assisting software engineering. In practice, many such tools have attempted much by being very comprehensive, but accomplished less because of weaknesses in usability. Seeker has been created with relatively narrow goal, but does succeed by providing high usability for the purpose intended. I hope that the tool proves useful itself, or in making a small contribution toward better tool support for learning OO design.

7 Software Status

I still regard the current version of Seeker as a prototype, but am willing for others to try it out. It is implemented in Incr TCL/Tk, and may therefore be used with the free versions of Incr TCL/Tk available for Unix, Windows, and MacOs. The current version of Seeker is available free on an as-is basis for non-commercial use. For commercial use, please contact me. The software and documentation is available from the web site:

`http://www.mcs.vuw.ac.nz/research/design1/`

References

[1] K. Beck and W. Cunningham. A laboratory for teaching object-oriented thinking. In *Proc. of OOPSLA-89: ACM Conference on Object-Oriented Programming Systems Languages and Applications*, pages 1–6, 1989.

[2] Robert Biddle, Stuart Marshall, John Miller-Williams, and Ewan Tempero. Reuse of debuggers for visualization of reuse. In *Proceedings of the Symposium on Software Reusability*, 1999.

[3] Robert Biddle and Ewan Tempero. Understanding the impact of language features on reusability. In Murali Sitaraman, editor, *Proceedings of the Fourth International Conference on Software Reuse*. IEEE Computer Society Press, April 1996.

[4] F. Buschmann, R. Meunier, H. Rohnert, P. Sommerlad, and M. Stal. *Pattern-Oriented Software Architecture: A System of Patterns*. Wiley, New York, 1996.

[5] Martin Fowler and Kendall Scott. *UML Distilled: Applying the Standard Object Modeling Language*. Addison-Wesley, 1997.

[6] J. Iivari. Why are CASE tools not used? *Communications of the ACM*, 39(10):94–103, October 1996.

[7] Jakob Nielsen. *Usability Engineering*. Academic Press, New York, 1992.

[8] Lauren Wiener Rebecca Wirfs-Brock, Brian Wilkerson. *Designing Object Oriented Software*. Prentice Hall, 1990.

[9] Marc Rettig. Prototyping for tiny fingers. *Communications of the ACM*, 37(4):21–27, April 1994.