Clicki: A Framework for Light-weight Web-based Visual Applications

Donald Gordon **James Noble** **Robert Biddle**

School of Mathematics, Statistics and Computer Science
Victoria University of Wellington,
PO Box 600, Wellington, New Zealand,
Email: {donald,kjx}@mcs.vuw.ac.nz

Human-Oriented Technology Laboratory
Carleton University
1125 Colonel By Drive, Ottawa, Canada
Email: robert_biddle@carleton.ca

Abstract

Web application frameworks typically provide little support for graphical web applications such as diagram editors. This restricts applications developed using these frameworks to either very limited interactions, or using technologies that are only supported by a small subset of web browsers. Clicki is an object-oriented web application framework in the style of traditional graphical user interface frameworks, with minimal requirements on the client web browser. The Clicki framework enables the creation of nontraditional diagram-based web applications within the confines of the architectural constraints imposed by the web.

Keywords: Web, visual, framework

1 Introduction

The web browser provides a user interface paradigm that is both ubiquitous and well-understood. Web-based applications such as Hotmail, and more collaborative applications like the WikiWikiWeb (Leuf & Cunningham 2001) have proved exceedingly popular, despite the constraints placed on them by current browser technology, which limits them to a fraction of the user interface possibilities available to traditional GUI applications. Many software frameworks, including Sun's Java Servlets and JSP (Coward 2001), Microsoft's ASP.net (Liberty & Hurwitz 2003), Zope (Latteier & Pelletier 2001) and numerous others offer support for these traditional web applications.

The most common approach to building graphical web-based applications is to use heavyweight technologies such as Java Applets (Arnold & Gosling 1996) and Flash (Woods 2001). Accessibility of applications using these technologies is problematic due to inconsistent browser support in all but the most recent desktop browsers. The lightweight approach to developing graphical web-based applications, eschewing plugins for simple HTML and images, is useful because it is compatible with many browsers, including older versions on older machines, and the limited browsers found on hand-held computers and cellular phones.

Our research group has developed a number of lightweight visual web-based applications that allow the user to interact with some form of diagram: Seek (Khaled, McKay, Biddle, Noble & Tempero 2002), Ukase (Biddle, Noble & Tempero 2002), NutCASE (Mackay, Biddle & Noble 2003) and the original Clicki drawing editor

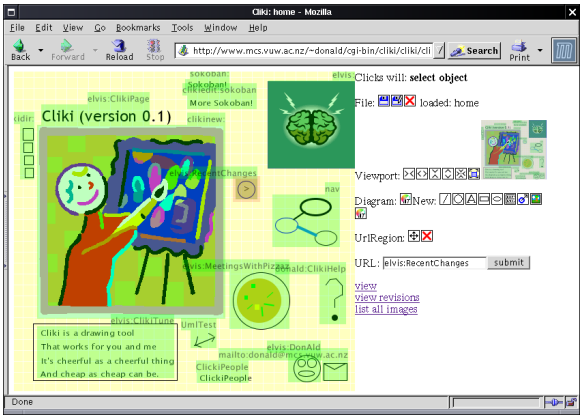

Figure 1: A diagram editor based on an early version of the Clicki framework

(Gordon, Biddle, Noble & Tempero 2003) (figure 1). These applications take advantage of the availability and ease-of-collaboration benefits shared with ordinary web applications, while employing a more visual interaction style usually associated with native applications. Our applications utilised a number of different means of displaying and allowing interaction with such diagrams within the constraints imposed by the web browser; in the end, one approach — image generation — appeared to offer the best solution.

Unfortunately, existing web application frameworks offer little-to-no support for image generation, and none of the higher level abstractions that would aid in building applications that involved the editing of diagrams. We have developed the Clicki framework, which offers facilities to ease the building of web-based applications with an API similar to that provided by frameworks for developing native GUI applications.

Several versions of Clicki, a diagram editor that utilises this approach, were developed. The framework described in this paper is the result of that development. The framework provides an interface to the programmer similar to that provided by traditional GUI frameworks, but the resulting applications have a web-based user interface.

This paper is structured as follows: in the next section, to set the context, we give examples of three applications built using the Clicki framework.

In sections three and four, we describe the Clicki forms framework and the Clicki diagram editor framework. The Clicki framework is divided into two parts: *forms*, which encompasses simple HTML generation and event handling, and the *diagram editor* which is built on top of the forms framework and provides an extensible web-based visual diagram editor. In section five, we discuss the state of the framework and in section six we describe earlier

web application frameworks. We conclude the paper in section seven.

2 Applications

The rule of three (Roberts & Johnson 1997) suggests that three applications should be built using a still-evolving framework, in order to develop suitable abstractions for the application domain. We have developed three applications, which we describe here.

2.1 Sokoban game

The game of Sokoban involves moving a worker around a map, pushing boxes onto shelves in order to complete a level. The Clicki-based Sokoban game was ported from a Sokoban game used in an assignment for a first-year computer science course. The porting process replaced the user interface with one using the Clicki forms framework. The `Image` page component's support for the standard Java AWT graphics interface meant that the code which handled displaying the levels could be used with only minimal changes.

When the user begins the game, they are presented with the first level (figure 2).

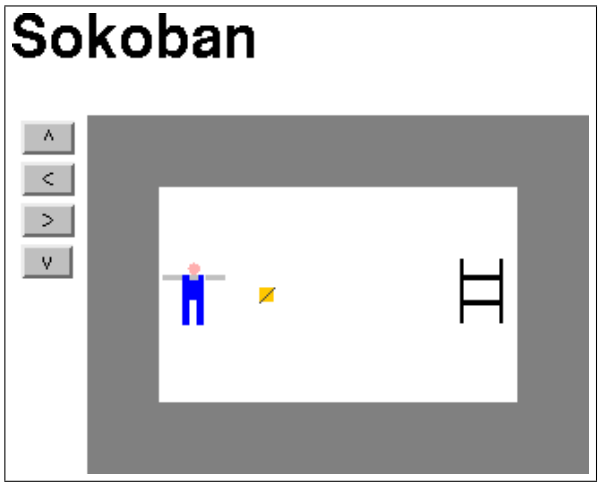

Figure 2: A new Sokoban game.

The game has a very simple user interface. It provides two ways of moving the worker. The buttons with arrows on them can be clicked to move the worker by single grid cells, or a destination cell can be clicked on directly. This causes the worker to move in a straight line, and greatly reduces the number of clicks (and thus time) required to solve levels.

The first level can be solved with a single click. Upon completion, the user is automatically transferred to the next level.

2.2 SokobanEdit

SokobanEdit provides a means for users to edit the levels used by the game of Sokoban that we described previously.

The first screen allows the user to select the level to be edited. Once the user has selected a level, they are transferred to the level editor (figure 3), which uses an early version of our diagram editor framework.

SokobanEdit represents levels as a two-dimensional array of tiles. Tiles may be either empty spaces, boxes, shelves, walls or the worker. They are displayed to the user using the same visual representation as employed by the Sokoban game.

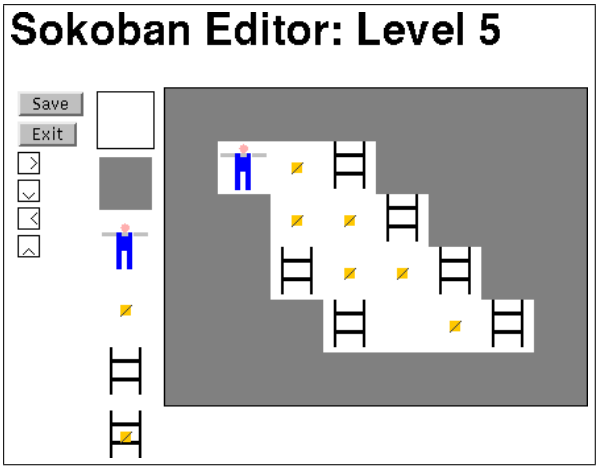

Figure 3: The Sokoban level editor

The Save button on the far left allows the level to be saved, and the arrows allow the size of the level to be adjusted. The pictures of tiles to the right of the buttons represent the available tile types with the currently selected tile enclosed by a black rectangle. All graphics are generated dynamically, taking advantage of Clicki's ability to re-use existing Java drawing code.

When a tile is clicked, it changes to match the currently selected tile-type. This simple interaction is sufficient to build potentially complex levels. When a level has been saved, it can be played with our forms framework based Sokoban game.

2.3 ClickiPoint

ClickiPoint provides an editor for producing simple slide-based presentations, similar to the presentations produced by Microsoft PowerPoint and other presentation tools. When the user first accesses the application, they are presented with a page that allows them to either open a presentation from the provided list, or name and create a new presentation (figure 4).

Figure 4: ClickiPoint: the file open/create page.

After the user has opened or created a presentation, the diagram editor page is displayed (figure 5). This page is provided by the framework, and customised by the ClickiPoint application.

The editor allows the user to perform several actions. They can switch to a different slide by clicking its thumbnail, create a new slide, create a new bullet point on the current slide, or edit one of the existing pieces of bullet-pointed text.

If the title or one of the points in a slide is clicked while the select tool is active, it is selected. At this point, the display of tools down the left-hand side of the image changes to show those tools which apply to the newly-selected portion of the diagram. The "edit text" tool is selected automatically, as the framework knows that it can be used on the current selection. This tool's interface is displayed

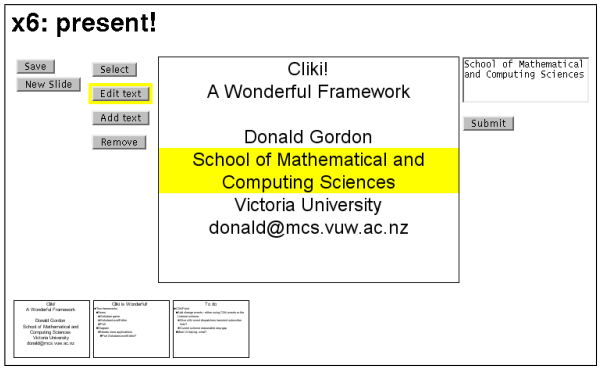

Figure 5: Selected text in the ClickiPoint presentation editor.

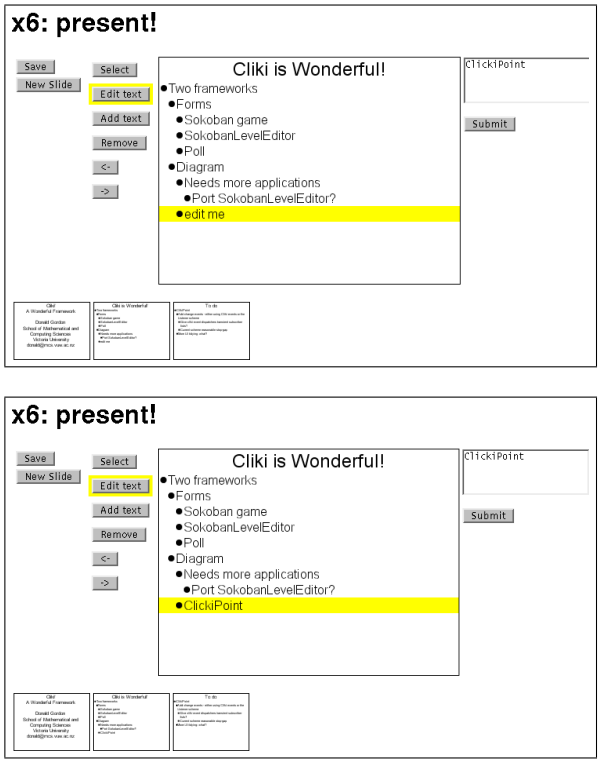

Figure 6: The image is updated after the text in the text box is changed and the submit button is clicked.

to the right of the diagram, allowing the user to edit text within the selected point or title. When the text is modified and the submit button is clicked, the tool updates the text within the selection (figure 6).

The user interface also provides other capabilities. Bulleted points can have their indent level changed via the `<-` and `->` buttons, and they can be removed. Slides cannot have their titles removed, or indented. Therefore, only when a bullet point within a slide is selected (figure 6) are the remove and indent level buttons made visible.

As ClickiPoint is a web-based application, multiple users can access it simultaneously. ClickiPoint contains the necessary support to allow the same presentation to be edited by multiple users simultaneously, by sharing the internal presentation model for a single file between different user session. This allows collaboration between users effectively “for free”.

3 Clicki Forms Framework

The Clicki Forms framework is an object-oriented framework for building web-based applications, similar to more

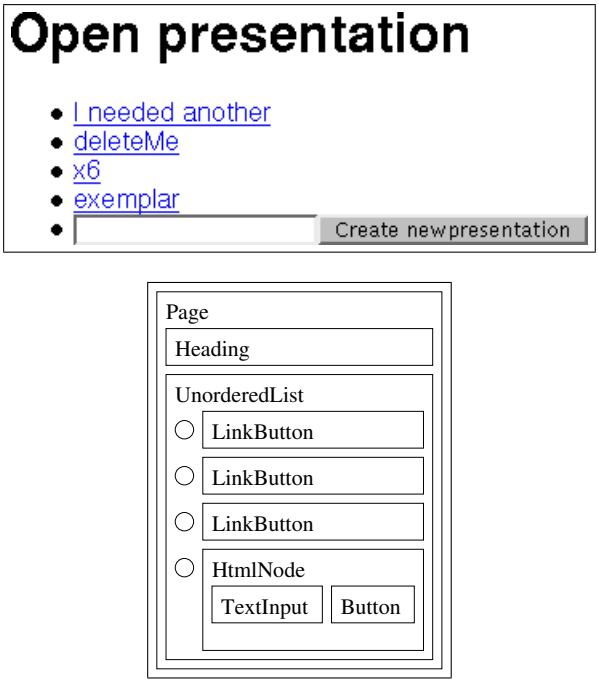

Figure 7: A page and the corresponding component tree

traditional OO GUI frameworks such as Java Swing (Sun Microsystems Inc. n.d.) and Qt (Ward 2000). The forms framework is divided into two sections: state management and a set of components from which dynamic web pages can be built. The state management section handles information tied to a single session with a user. The page they are currently viewing, and any application-specific information, such as the file they may have open, is remembered. Pages are built using a supplied set of page components that represent structures such as lists and paragraphs, as well as simpler items such as text boxes, buttons and links. These components can inform the application of changes to their state by producing events for which the application can listen.

This similarity with traditional GUI frameworks is very different to the template-based approach taken by most web-application frameworks. We believe that it offers greater flexibility than the template-driven approach, as it does not impose a text document-centric model on the application.

3.1 Pages

Forms-based applications represent web pages as a tree of objects of type `HtmlNode` and `HtmlLeaf` built up using the composite pattern (Gamma, Helm, Johnson & Vlissides 1994) in a similar manner to traditional GUI frameworks. Class hierarchies in our framework that conform to the composite pattern are composed of primitive objects (descendants of `HtmlLeaf`), and more complex objects. Complex objects can contain primitive objects (descendants of `HtmlNode`, which itself extends `HtmlLeaf`), all sharing the same interface (that provided by `HtmlLeaf`). This makes building complex aggregate structures out of more primitive objects simple. All objects can be treated similarly due to their implementing the same interface, and new types of objects can be added to the structure without extra code. For example, multilevel bulleted lists can be created in Clicki by adding both `UnorderedLists` and `Text` objects to other `UnorderedLists`. Figure 7 shows the correspondence between the object tree that represents a page, and how the page is displayed to the user.

The Image component makes building visual applica-

tions especially easy. This component provides an image within a page which can be drawn on with the standard Java AWT Graphics and Graphics2D classes. It is automatically re-rendered upon loading a page. When the Image component is clicked, it generates a CLICK event with a parameter describing the location within the image at which the click occurred.

Dynamically generated images must usually be reloaded by the browser on every page load, which can lead to the application appearing slow, due to the extra round-trips to the server which are incurred. Our image component implements the simple optimisation of allowing the browser to use its cache of the previously-generated image, if it is identical to the new one. This means that when the image has not changed, additional time or network traffic is not required to load it.

3.2 Events

All page components provided by the Clicki forms framework are capable of emitting events that an application may subscribe to. These events are used to notify the application of various occurrences. For example, the contents of text boxes changing, buttons being clicked, and page components being re-rendered all cause events to be emitted.

The event types are defined as static fields within the components that generate them. For example, the `CLICK` event for a `Button` is referred to as `Button.CLICK`. Events can be subscribed to in two ways: by extending the class that emits them if they are supported by that class, or by subscribing with a `Event.Target` object.

The general mechanism for subscribing to events involves calling the `subscribe` method on the event-emitting object, and passing it both an event type (e.g. `Button.CLICK`) and an event target. The event target object should extend the `Event.Target` class, whose `invoke()` method will be called when the event is fired.

Traditional GUI frameworks are able to send applications events immediately when a user modifies the value in a text box, or selects a new radio button. Clicki does not have this luxury, as it is only notified of such changes by the client web browser when a button or image on the page is clicked and the form is submitted. This sequence is problematic because the application does not know of value changes until after the button is clicked. We deal with this problem by re-ordering events. Events which cause page submission, such as button or image clicks are sent to the application after other events such as those generated when the contents of some form element, like a text input box being changed. `UPDATE` events are a special case. Provided by all page components, this event is fired and sent to the application immediately before the page component emitting it renders itself to HTML. This is often used to update the contents of some part of the page to the most recent available data.

3.3 State Management

One of the challenges of building web-based applications is that the HTTP protocol used to deliver web pages is stateless. It does not define a mechanism for maintaining a session with the web server between requests for web pages. Thus, web-based application frameworks typically provide a means of simulating some form of persistent session state, allowing the application developer to build applications accessible via the stateless HTTP protocol (Fielding, Gettys, Mogul, Frystyk & Berners-Lee 1997) while maintaining persistent state between page requests.

Clicki's session state has different semantics to that provided by most other web-application frameworks. Commonly, web-application frameworks tie session state to the user's browser using a cookie. Cookies are a technology that helps applications to retain state between page requests by leaving a small piece of data — a cookie (Kristol & Montulli 1997) — with the web browser, which the browser will then include with each subsequent request for a new page. This is not suitable for some uses of Clicki, however, as it would constrain the user to accessing at most a single session of a each application, per browser — disallowing, for example, the use of multiple edit windows. To avoid this problem, Clicki's state management uses hidden form fields to recognise a given session. These are specific to the page displayed rather than the user's browser, and thus allow simultaneous access to multiple application sessions with no extra work on the application's part.

The session state mechanisms provided by most web-application frameworks also represent session state as a set of key/value pairs. Such a mechanism can be inconvenient to use, as it requires more type-checking to be performed at runtime as values from the session state map are cast to the correct type. Mistyping a session key can cause bugs which are difficult to trace. Clicki avoids these problems by representing session state as a Java object (`State`) that can be extended by the application to add whatever fields or methods it requires. The Clicki forms framework will keep track of this object between page requests.

4 Clicki Diagram Editor Framework

The Clicki Diagram Editor framework provides a user interface for manipulating diagrams, built on top of the Clicki Forms framework described in the previous section. Diagram editors have a `View`, representing the visual depiction of the diagram, `Selections` representing a currently-selected subset of the diagram, `Operations` which represent actions which can be performed on selections, and `Tools` which provide a user interface to apply `Operations` to `Selections`. `Tools` provide this interface using `Interactions`, which describe a set of steps the user must carry out to use the tool.

Applications using the editor typically also use the forms framework directly. The forms framework is used provide parts of the application not covered by the diagram editor, such as a means of selecting a diagram to edit, and also to describe the interface provided by an `Interaction`.

The diagram editor page (figure 8) provides an area for the diagram `View` to be displayed, an area for UI elements that are only loosely related to diagram state, such as save and load buttons, an area to display the buttons used to select tools, and an area for use by the tool the user is currently using to interact with the diagram. The editor manages the current selection, tool visibility and activation, and redrawing the diagram. It automatically selects the most frequently used tool, in a similar manner to Microsoft PowerPoint selecting the resize/move tool by default. This reduces the number of clicks required to perform operations in the diagram editor, and thus increases the perceived responsiveness of the application.

4.1 View

The application is entirely responsible for the internal representation and view of the diagram: no layout support is provided by the current framework. When the diagram editor page is created, the application must provide an object implementing the `View` interface, which provides methods allowing the diagram editor to:

- draw the diagram.
- retrieve a rectangle describing the view's bound.
- determine whether the view contains anything within a given region.

- obtain a *selection* given a region.

Diagrams are displayed by passing the `View` a Java `Graphics2D` object. This object is provided by the Java AWT, the standard interface for drawing images in Java. The application can then use this to update the display of the diagram.

4.2 Selections

`Selections` represent a portion of selected diagram in the editor. This usually corresponds to one object in the application's internal representation of the diagram, but this is not required. Generated by the `View` that the application provides to the drawing editor when given a `SelectRegion` object, selections:

- provide a rectangle describing the bounding box around the portion of the diagram represented by the selection. This rectangle is commonly used to highlight the selected region.
- determine if the selected item or items supports a given `Operation` type.
- obtain an `Operation` object of a given type.

The framework provides two helper classes for the common cases encountered producing selections. `SingleSelection` represents a single selected object which implements all the interfaces representing the operations it supports, and is capable of providing its own bounding box. If this is not the case, `SingleSelection` can be subclassed to provide one. `AugmentedSelection` has similar behaviour, except that it can be supplied a list of extra operations it should support, and provides hooks for the application to supply these when requested. This is useful for providing sets of compound operations which are not directly supported by the selected object.

4.3 Operations

Operations are represented by Java interfaces extending the `Operation` interface; they specify a set of related methods which allow a tool to perform some operation on a part of the underlying diagram model, such as moving or editing the text within some component of the diagram. This provides a means of formalising the interactions between `Tools` and selected diagram components. For instance, a `TextEditable` operation could be defined, with methods to get and set the editable text within a selected component. Then a generic text editing tool would automatically allow editing of the given component.

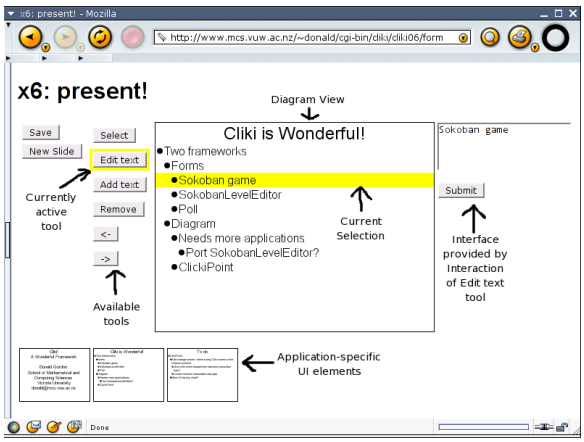

Figure 8: Typical interface produced using the Clicki Diagram Editor framework.

Simple operations are often implemented directly by the underlying classes represented by the selection. Complex operations, such as those which affect the selected objects indirectly – for example, the object's position within an enclosing list — must be handled as a special case by the selection.

4.4 Tools

The application must also provide a set of `Tools`. These provide the user interface that allows the user to manipulate the diagram. `Tools`, except for those which create new objects within the diagram, typically do not operate directly on the diagram. They access it via the `Operations` made available by a `Selection`, which helps to decouple the tool's user interface from the underlying representation of the diagram. When the tool is activated, the tool's user interface, often comprising a sequence of steps to be followed by the user, is represented with an `Interaction`. `Tools` can also be queried to see if they can be used with the portion of a diagram represented by a `Selection`; this is used by the diagram editor to decide which tools to display, and which tools to automatically activate. `Tools` commonly determine this by querying the set of operations supported by the given selection.

One of the most frustrating parts of using lightweight web-based visual applications is having to wait for a network round-trip to redisplay the page and image every time the user clicks on something. By automatically activating tools that are likely to be used, the number of clicks required to perform common operations are reduced.

4.5 Interactions

The interaction model, used successfully by NutCASE (Mackay et al. 2003) and earlier versions of the Clicki drawing editor (Gordon et al. 2003), allows interaction with diagrams while staying within the constraints imposed by the web. Many of the interactions with the diagram end up as a sequence of events similar to the following:

1. The diagram is displayed to the user.
2. The user clicks on the diagram to select an item to manipulate, such as a UML class.
3. The view of the diagram is updated to highlight the selected region.
4. The user utilises the application's interface (potentially involving more clicking on the diagram) to manipulate the newly-selected portion of the diagram.

Some actions, such as moving a selection around until it is in the correct location, or adding extra points to a polygon, can involve a large number of clicks on the diagram. The `Interaction` is one of the abstractions carried over from the earlier Clicki diagram editor appear in the diagram framework in a form that takes advantage of the forms framework. They provide sequencing facilities to elicit a set of clicks or other information from the user. They also provide a forms interface representing the user-visible interface of the current step, and a method for handling clicks on the diagram, which has the ability to indicate whether or not the click was any use to the interaction.

`Interactions` also have the ability to draw on the diagram display area of the diagram editor. This is often used to indicate the currently selected item, and remind the user of the results of previous steps when progressing through a multi-step interaction, as seen in figure 9. One common use for multi-step interactions is obtaining a set of points from a user. An interaction could provide the user interface for drawing a polygon by accepting clicks

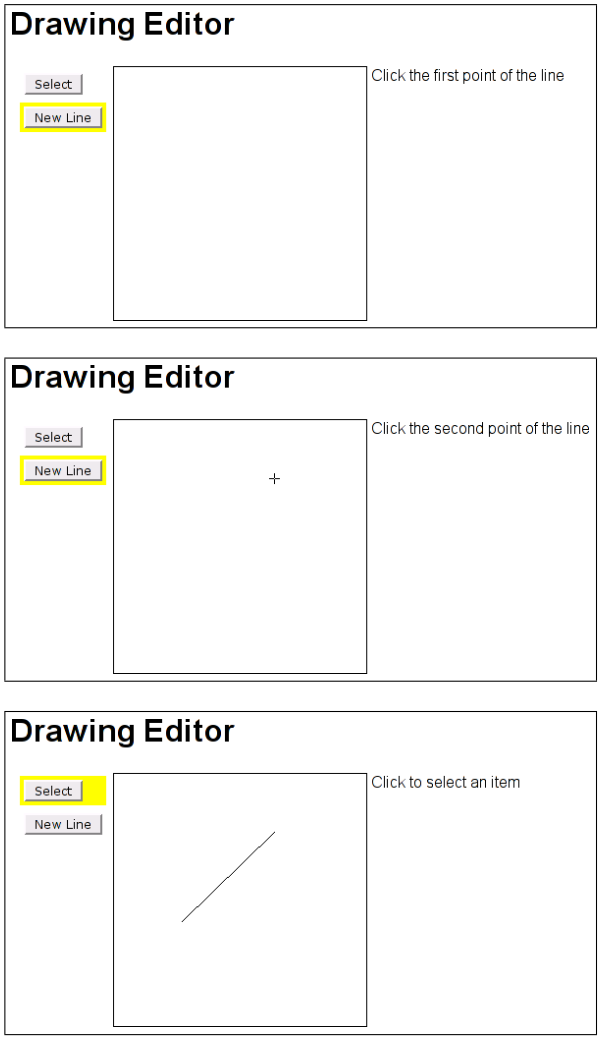

Figure 9: An interaction sequence used in creating a line. As the technology Clicki uses does not support dragging, the user must click twice to select the two line endpoints; then the line is created.

on the diagram identifying vertices for the polygon, with a button for the user to click when the polygon had been drawn. Another common use is for moving an object. The interaction can allow the user to continue clicking on the diagram to select a location until they're satisfied that the object is placed correctly.

5 Discussion

The Clicki framework successfully demonstrates that it is possible to build a framework similar to traditional object-oriented graphical user interface frameworks, but providing a web interface. Other users of the framework have confirmed that it is reasonably easy to rapidly develop new applications using it.

The framework, however, has several deficiencies. Unlike most modern GUI frameworks, Clicki does not provide a visual user interface builder. The framework also does not provide any ability to load an XML file describing a user interface. Transfer delay can make Clicki applications feel more sluggish than applications running locally on the user's computer. Every click on a button, image, or link requires another round-trip to the server to retrieve a new page.

A disadvantage of the present image-generation approach is that new images must be downloaded for every page refresh. This makes the application much slower for users on low bandwidth internet links, such as dial-up or cellular connections, or those who are located some distance from the server.

Clicki does include some optimisations to help in this case. The image component implements the simple optimisation of allowing the browser to use its cache of the previously-generated image, if it is identical to the new one. This means that the image will only be reloaded along with the new page if the image contents have changed. Unfortunately, when only a small part of the image changes, the entire image, sometimes over 40kB in size, must still be reloaded.

We are looking at techniques to ameliorate this problem. One possibility is to automatically slice images into a set of smaller images, and then only require the subimages which have changed to be reloaded. Another possibility is to layer a partially transparent image over the last full-size image to be sent, the only opaque sections being the parts of the image which have changed. This may, however, incur problems with older browsers.

6 Related Work

Object-oriented software frameworks (Johnson & Foote 1988) are class libraries designed to support building applications within a specific domain, by providing suitable abstractions and concrete implementations of classes. Applications can then use these by specialising classes provided by the framework to support the application's specific needs.

Many commercial and open-source frameworks for building web applications are available, including Sun's JSP and Java Servlets (Coward 2001), Microsoft's ASP.net (Liberty & Hurwitz 2003) and Digital Creations' Zope (Latteier & Pelletier 2001). These typically utilise a template-based system, where pages are described as HTML with framework-specific directives that are interpreted dynamically at runtime. As such, they do not attempt to hide the stateless request/response model mandated by the web, beyond providing some form of session abstraction which allows data to be associated with a single user's session with the application. The existing framework which bears most similarity to the Clicki Forms framework is ASP.net (Liberty & Hurwitz 2003), Microsoft's framework du jour for developing web applications. Like most web-application frameworks, ASP.net represents web pages as template HTML documents containing special dynamic ASP.net components. Unlike previous versions of ASP, ASP.net attempts to provide a programming interface more similar to that provided by traditional Graphical User Interface (GUI) toolkits, by allowing the application to receive events from page components. It is still template driven, however, and does not provide any support for image generation.

Java Servlets (Coward 2001) is one of Sun's web-application frameworks. It provides a very spartan feature set, essentially an interface between a web server and an application. The interface represents HTTP page requests as method calls, and requests and responses as objects containing methods that allow extraction of data related to the page request. Several groups, including Sun itself, have built other frameworks on top of Java Servlets, such as JSP (Pekowsky 2003) and Struts (Cavaness 2002). Java Servlets does not provide any support for light-weight web-based visual applications.

Many diagram editors have been built on top of traditional object-oriented graphical user interface frameworks. UniDraw (Vlissides & Linton 1990) is a diagram editor framework built on top of the InterViews (Linton, Vlissides & Calder 1989) graphical user interface framework. It has a comprehensive system for describing inter-relationships between different component types, a means of describing the geometry of symbols that form part of a diagram, and a Model-View-Controller (Gamma et al. 1994) (MVC) style partitioning between the repre-

sentation and view of the diagram. It also provides extensive support for structuring and laying out diagrams, undo for every operation that tools can perform on the interface, and a well-defined user interaction model for tools.

HotDraw is a framework for building structured two-dimensional drawing editors for specialised types of drawings, either standalone or as part of a larger system. Elements of these drawings can have constraints between them, and can be animated. They can be aggregated to produce more complex elements. Tools can be defined, which allow the user to manipulate elements of the drawing or the entire drawing, and create new elements.

7 Conclusion

In this paper we have described Clicki, a framework which enables the building of web applications. Unlike previous web-application frameworks, Clicki provides an interface to the programmer similar to that of traditional object-oriented GUI frameworks.

The lightweight applications built with the Clicki framework are accessible in practically any browser which supports graphics, including older browsers and the limited browsers embedded into hand-held computers and cellular phone. This gives them an advantage over applications built with heavyweight technologies and their accompanying poor browser support. They are still able to utilise an interaction style similar to that of traditional graphical applications.

References

Arnold, K. & Gosling, J. (1996), *The Java Programming Language*, The Java Series, Addison-Wesley, Reading, MA, USA.

Biddle, R., Noble, J. & Tempero, E. (2002), Supporting reusable use cases, *in* C. Gacek, ed., 'Software Reuse: Methods, Techniques, and Tools, 7th International Conference, ICSR-7, Austin, TX, USA, April 15-19, 2002, Proceedings', Vol. 2319 of *Lecture Notes in Computer Science*, Springer.

Cavaness, C. (2002), *Programming Jakarta Struts*, O'Reilly & Associates, Inc.

Coward, D. (2001), *Java Servlet Specification Version 2.3*, version 2.3 edn, Sun Microsystems Inc.

Fielding, R., Gettys, J., Mogul, J., Frystyk, H. & Berners-Lee, T. (1997), 'RFC 2068: Hypertext Transfer Protocol — HTTP/1.1'.

Gamma, E., Helm, R., Johnson, R. E. & Vlissides, J. (1994), *Design Patterns*, Addison-Wesley.

Gordon, D., Biddle, R., Noble, J. & Tempero, E. (2003), A technology for lightweight web-based visual applications, *in* M. Burnett & J. Grundy, eds, 'IEEE Symposium on Visual and Multimedia Software Engineering', Auckland, New Zealand.

Johnson, R. E. & Foote, B. (1988), 'Designing reusable classes', *Journal of Object-Oriented Programming* .

Khaled, R., McKay, D., Biddle, R., Noble, J. & Tempero, E. (2002), A lightweight web-based case tool for sequence diagrams, *in* 'Proceedings of SIGCHI-NZ Symposium On Computer-Human Interaction (CHINZ 2002)'.

Kristol, D. & Montulli, L. (1997), 'RFC 2109: HTTP state management mechanism'.

Latteier, A. & Pelletier, M. (2001), *The Zope Book*, New Riders.

Leuf, B. & Cunningham, W. (2001), *The Wiki way: quick collaboration on the Web*, Addison-Wesley Longman Publishing Co., Inc.

Liberty, J. & Hurwitz, D. (2003), *Programming ASP.NET*, O'Reilly & Associates, Inc.

Linton, M. A., Vlissides, J. M. & Calder, P. R. (1989), 'Composing user interfaces with InterViews'.

Mackay, D., Biddle, R. & Noble, J. (2003), A lightweight web based case tool for UML class diagrams, *in* R. Biddle & B. Thomas, eds, 'Proceedings of the 4th Australasian User Interface Conference', Australian Computer Society, Adelaide, South Australia.

Pekowsky (2003), *JavaServer Pages*, Addison-Wesley Longman Publishing Co., Inc.

Roberts, D. & Johnson, R. E. (1997), Evolving frameworks: A pattern language for developing object-oriented frameworks, *in* 'Pattern Languages of Program Design 3', Addison Wesley.

Sun Microsystems Inc. (n.d.), *Java Foundation Classes: Cross-platform GUIs and Graphics*. http://java.sun.com/products/jfc/.

Vlissides, J. M. & Linton, M. A. (1990), 'Unidraw: a framework for building domain-specific graphical editors', *ACM Transactions on Information Systems (TOIS)* **8**(3), 237–268.

Ward, P. (2000), *QT Programming for Linux and Windows*, Prentice Hall PTR.

Woods, P. S. (2001), *Macromedia Flash(TM) 5 Developer's Guide*, McGraw-Hill Professional.