

# Generating Web-based User Interfaces for Diagramming Tools

Shuping Cao<sup>1</sup>, John Grundy<sup>1,2</sup>, John Hosking<sup>1</sup>, Hermann Stoeckle<sup>1</sup>, Ewan Tempero<sup>1</sup> and Nianping Zhu<sup>1</sup>

Department of Computer Science<sup>1</sup> and Department of Electrical and Computer Engineering<sup>2</sup>,  
University of Auckland, Private Bag 92019, Auckland, New Zealand

{john-g, john, herm, ewan, nianping}@cs.auckland.ac.nz

## Abstract

Thin-client diagramming tools provide a number of advantages over traditional thick-client design tools but are challenging to build. We describe an extension to a thick-client meta-tool that allows any specified diagram editor to be realised as a thin-client tool. A set of server-side components interact with the thick-client tool to generate GIF or SVG diagrams for both display and editing in a conventional web browser. We describe the motivation for our work, our novel architecture, illustrate and discuss interaction issues with the generated diagrams, and describe evaluations of the effectiveness of our approach.

*Keywords:* Thin-client diagramming, meta-tools, web-based design tools.

## 1 Introduction

Traditional diagramming tools use a thick-client, desktop interface and architecture. These generally work very well in terms of providing highly responsive diagram editing and viewing facilities, can leverage sophisticated thick-client interaction techniques and typically manage information on local workstations in a distributed fashion (Gordon et al, 2003; Green, 1989; Khaled et al, 2002). Disadvantages of this approach include the need to install and update software on every user's workstation, the complexity and learning curve associated with using many tool interfaces, the complex, heavyweight architectures needed to support collaborative editing, and lack of support in most tools for modifying diagramming notations and semantics (Lyu and Schoenwaelder, 1998; Maurer and Holz, 2002; Iivari, 1996).

Thin-client diagramming tools use a web browser for editing diagrams. Users view diagrams as content of a "web page" that also includes controls such as buttons and links for modifying the diagram or moving to other diagrams (Graham et al, 1999; Maurer and Holz, 2002). Diagrams may be constructed with combinations of automatic and manual layout, resizing, highlighting and so on. Technologies to render the diagrams might include GIF images, SVG (Scalable Vector Graphics) renderings, and 3D VRML (Virtual Reality Modelling Language)

visualisations. Key potential advantages of this approach are a consistent look and feel across all web-based diagramming tools, use of web page design techniques to aid interaction and learning, no need to install and update copies of tools on every workstation, and use of conventional web architectures to support collaborative work by multiple users on the same design.

We have been developing a meta-tool that provides very flexible diagram and meta-model specification techniques. Our tool, Pounamu (Zhu et al, 2004), is used initially to specify the new diagramming tool, and then is used to interpret the specification to provide the new tool's functionality. This means that, since Pounamu has a thick-client interface, the newly-specified tool also has a thick-client interface. The first question we address in this work is whether thin-client interfaces can be provided for tools specified using Pounamu without significantly changing Pounamu itself. We have built an extension to Pounamu, called Pounamu/Thin that provides a proof-of-concept thin-client diagramming infrastructure that allows Web browsers deployed on each user's workstation to manipulate the diagrams.

Web browser-based user interfaces are more restricted in the types of interaction they allow as compared to thick-client interfaces. The second question we address in this work is whether and to what degree these restrictions can be overcome. We have developed support for different interaction styles in Pounamu/Thin; one that uses the GIF format, which is usable by all browsers, but has limited interactivity, and one that uses SVG, which requires a separate plug-in, but which allows more interaction. We report on the effectiveness of these two styles of user interface

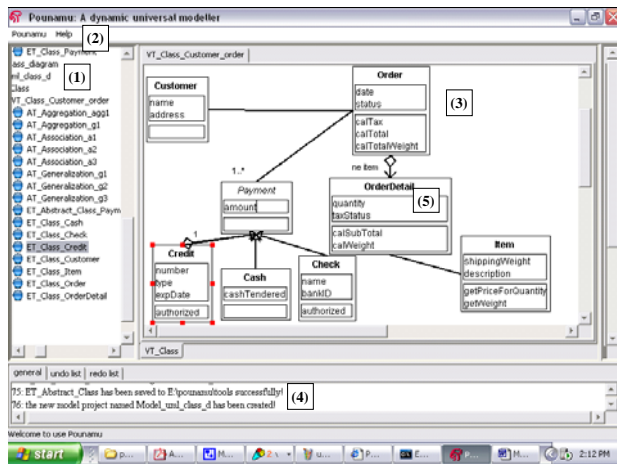
We firstly provide a motivation for this work and review related diagramming tool research. We then outline our approach as an extension to our thick-client meta-tool and describe the architecture of our system. We illustrate the thin-client diagramming support it provides with parts of a UML software design tool, and discuss key design and implementation decisions. We summarise results of evaluations of our prototype tool and then outline directions for future research.

## 2 Motivation

The traditional approach to providing diagramming tools, such as CASE tools, CAD tools and user interface design tools, is to use a thick-client program – one that runs on the host computer and uses its desktop graphics facilities. Key advantages are highly responsive interaction when directly manipulating diagrams and the ability of tool

developers to make use of all the graphics facilities available on the host computer. However, traditional diagramming tools suffer from the common problems of other thick-client applications: they need to be installed and periodically updated on all host computers; they often provide complex and difficult to learn user interfaces; and they require quite complex infrastructure to support multi-user collaborative work, especially synchronous diagram editing.

We have been developing a new thick-client meta-tool called Pounamu. This provides a set of tools for designing shape and connector appearance, meta-model elements, views of meta-model elements using shapes and connectors, and event handlers for specifying semantic behaviour (Zhu et al, 2004). Figure 1 shows an example of a diagramming tool generated by Pounamu, a Unified Modelling Language (UML) CASE tool, being used. A thick-client interface is provided for all Pounamu tools, which includes an element tree (1), pop-up and pull-down (2) menus, drawing canvas (3), shape property editor, status window (4), and directly-manipulatable shapes (5) and shape elements.



**Figure 1. Pounamu thick-client UML design tool.**

As web-based user interfaces have become more pervasive many applications that traditionally used thick-client interfaces, such as enterprise management systems, accounting packages, document management and email tools, have often become available as thin-client versions. Key design features of these web-based applications include a focus on simple, easy-to-use and consistent user interface designs, seamless support for collaborative work via the client-server approach inherent to web applications, and server-side integration with legacy systems and their facilities. We wanted to experiment with thin-client diagramming support for Pounamu-based diagramming applications to try and leverage these advantages in our tools. Our aim was to provide an optional extension to Pounamu allowing some users to choose to access generated diagramming tools via a web-based, thin-client infrastructure instead of a thick-client.

Much work has been done over many years in developing Computer-Aided Software Engineering tools that provide thick-client diagramming support. Examples of such tools include Rational Rose™ (Quatrani and Booch, 2000),

Argo/UML (Robbins et al, 1998), and our own JComposer CASE tool (Grundy et al, 2000). These tools provide a thick-client user interface with each user having a copy of the tool on their desktop. The need to be able to tailor design tools to different user's preferences and to develop new design methods and tools led to the development of meta-CASE tools. These include tools such as MetaEDIT+ (Kelly et al, 1996), Kogge (Ebert et al, 1997), JViews (Grundy et al, 2000) and MetaMOOSE (Ferguson et al, 2000). Despite the fact that these tools include a specification of the intended diagramming tool that might be interpreted to provide a thin-client version of the tool, none do so. Those tools that provide web-based diagramming almost always use Java Applets or similar thick-client browser plug-ins (Graham et al 1999).

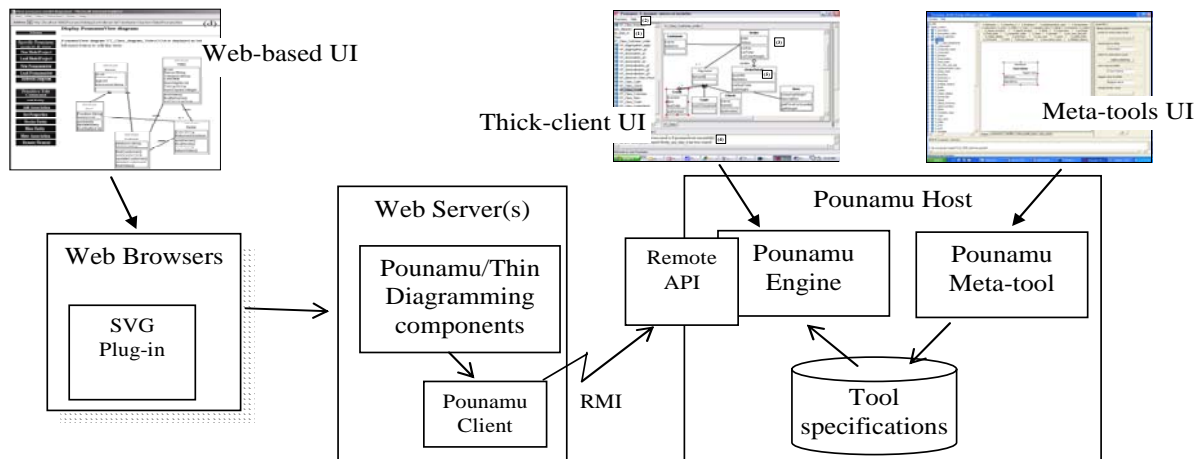
Various thin-client software engineering tools have been developed to exploit web-based delivery of information. Some examples include MILOS (Maurer and Holz, 2002; Maurer et al, 2000), a web-based process management tool, BSCW (Bently et al, 1995), a shared workspace system, software education environments (Chalk, 2000), Web-CASRE (Lyu and Schoenwaelder, 1998), which provides software reliability management support, and web-based tool integration approaches (Kaiser et al, 1998). Most of these tools do not provide any kind of diagram interface. Often web-based interfaces are used for visualising diagrams, such as UML diagrams derived from formal specifications in TCOZ (Sun et al, 2001), but not constructing the diagrams themselves.

Some recent efforts at building web-based diagramming tools have included Seek (Khaled et al, 2002), a UML sequence diagramming tool, NutCASE (Mackay et al, 2003), a UML class diagramming tool, and Cliki (Gordon et al, 2003), a thin-client meta-diagramming tool. These have demonstrated with proof-of-concept diagramming tools that web-based interfaces can be used to visualise and edit specific kinds of diagrammatic content. To date all of these have used custom approaches to realising the thin-client diagramming tool and users must access the systems via a web interface. Limited tailoring of notations is supported in Cliki but not the other tools.

### 3 Overview of Our Approach

Our Pounamu meta-tool was developed to provide flexible, interactive diagramming tool specification. It was assumed that all of the specified diagramming tools would provide a thick-client user interface to tool users, by interpreting these design tool specifications. However, we wanted to experiment with a new approach to providing thin-client diagramming interfaces from these exact same design tool specifications.

To do this we decided to develop an optional thin-client diagramming extension to Pounamu. This is a set of web components developed independently of the Pounamu meta-tool itself and using existing APIs that Pounamu provides. The aim was to support thin-client diagram viewing and editing, but without needing to change any code in Pounamu itself and using the same tool specifications as thick-client tools. Figure 2 illustrates this thin-client extension, which we call Pounamu/Thin.



**Figure 2. Using the Pounamu meta-CASE tool with thin-client diagramming.**

Initially a tool designer specifies a diagramming tool using the thick-client tool design facilities of our original Pounamu meta-tool. These tool specifications are saved to an XML-based tool repository for use by the Pounamu tool interpreter, which originally provided only thick-client diagramming facilities. We deploy Pounamu and Pounamu/Thin web components and the tool specifications are loaded by Pounamu and interpreted to provide the specified diagramming tool. The Pounamu/Thin web components interact with Pounamu via a remote RMI-based API.

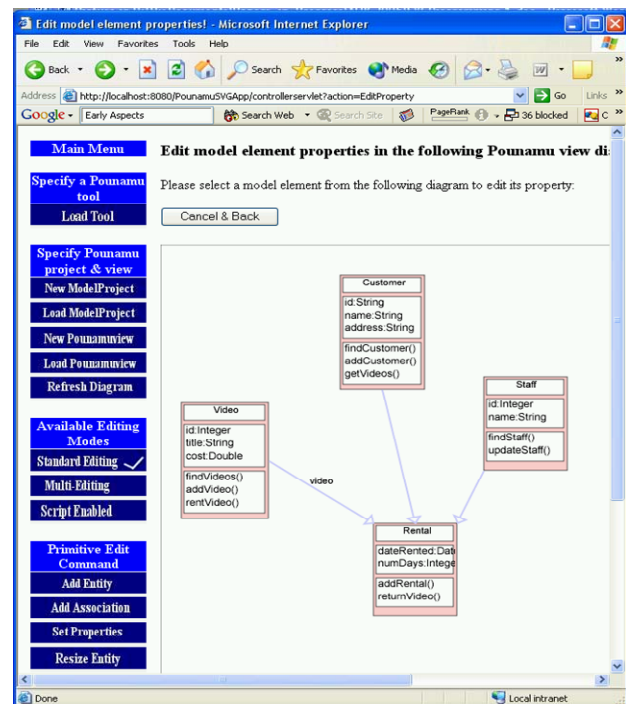
Users access the diagrams generated by the Pounamu/Thin web components via conventional web browsers. SVG-based rendering of diagrams is supported but this also requires an SVG plug-in in the browser. The advantage of SVG diagrams is that they are rendered on the client-side resulting in higher resolution than GIF-based diagrams. In addition, SVG rendered diagrams also support client-side browser scripting for more interactive diagram manipulation, including drag-and-drop direct manipulation for moving and resizing of content.

As Pounamu is a meta-tool, the tool specifications such as diagram element appearance, meta-model entities and attributes, view definitions and event handlers, can be changed while the tool is in use. This is currently done using the thick-client tool designer but future extensions might use a thin-client meta-tool designer via the Pounamu/Thin architecture itself.

#### 4 Thin-client Interactions

In this section we illustrate the user interfaces of our thin-client diagramming tools with an exemplar tool, a simple Unified Modelling Language (UML) design tool prototype. This is but one exemplar tool – as Pounamu is a meta-tool the appearance and semantics of the tool can be changed on-the-fly by the user, or a completely different diagramming tool specified and used. The Pounamu/Thin web components are completely tool-independent i.e. no code changes are necessary to support rendering and editing of any diagramming tool specified in our original Pounamu meta-tool designer.

An example of a UML class diagram view being edited in the thick-client tool interpreter is shown in Figure 1. In this prototype UML tool we have specified UML class diagrams (which include class shapes, generalisation and association connectors, and various annotations), sequence diagrams (which show object and method liveness shapes, and method invocation connectors), collaboration diagrams (showing object and message invocation shapes and message flow connectors), and deployment diagrams (showing machine shapes and structural connectors). Figure 3 shows the web client user interface of Pounamu/Thin when editing a class diagram. This web browser-based user interface includes a set of buttons to control the project (add views, open views); a set of buttons to manipulate the diagram (add shapes and connectors, move and resize elements, edit element properties); and a diagram, which can be clicked on to manipulate it.



**Figure 3. Example of using Pounamu/Thin.**

There are three different interaction styles for users of Pounamu/Thin. Users can move between these different interaction approaches at any time while using the tool. Each of these interaction approaches behaves differently and has certain advantages and limitations compared to the others. The options are:

- Fully server-side interaction processing, where each diagram interaction is always passed to the web components and then onto the Pounamu server for processing. After updating the shared copy of the source diagram, the diagram is redisplayed in the user's web browser.
- Buffered edits, where a user's diagram edits are actioned on their copy of the diagram only. These are subsequently sent on user request as a set of edits to the Pounamu server and actioned as a unit on the shared copy of the source diagram. Then the diagram is redisplayed in the user's browser.
- Client-side scripting, where some diagram interactions are handled by JavaScript in the client web browser. These include move, resize and deletion of diagram components using direct manipulation techniques. These are subsequently sent on user request as a set of edits to the Pounamu server and actioned as a unit on the source diagram.

#### 4.1 Fully Server-side Interaction Processing

A fully server-side processing model is used for GIF image diagrams and can also be used for SVG diagrams. This approach is illustrated in Figure 4. In this approach,

all interactions with the diagram are sent directly to the web server components (1) which immediately pass the interactions onto the shared Pounamu server (2) as updates to the shared diagram. The server updates the shared diagram (3) and then the Pounamu/Thin servlet requests the updated diagram contents (4) to generate HTML/GIF/SVG content for redisplay in the web browser (5).

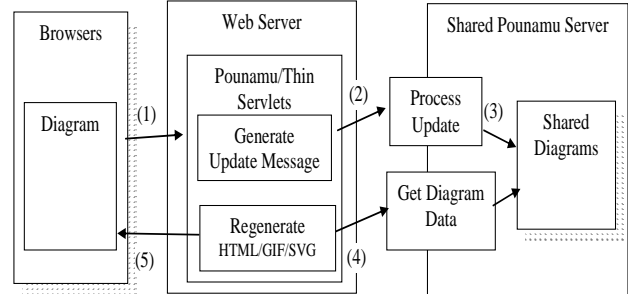


Figure 4. Fully-server side processing model.

Figure 5 shows some examples of creating and manipulating a UML diagram in Pounamu/Thin. The user first selects a Pounamu tool specification to use. This loads the tool specification into the Pounamu server. The user can create or use an existing model project, and within this model project create and use diagrams of various types. For the UML tool, diagram types include class, sequence, deployment and component diagrams. Each diagram type has a different set of symbols, editing syntax and semantics.

**1. Select tool/project**

**2. Select/create diagram**

**3. Select mode/click to add**

**4. Select mode/click to select/edit properties**

PropertyName	PropertyType	PropertyValue
<b>Model Properties</b>		
attribute	MultiLineText	id: Integer title: String cost: Double
method	MultiLineText	findVideos () addVideo () rentVideo ()
<b>Visual Properties</b>		
name	String	Video

Figure 5. Interacting with the fully server-side processing client.

In this example, a user creates a class diagram (2). The user then creates a class element in the diagram. To add an element the user selects the Add Entity button (3), specifying the kind of element to add to the diagram (a ClassEntity in this case). The user clicks where the new element is wanted and a blank class shape is displayed.

To set properties for this new element the user selects Set Properties, clicks on the shape and a property list for the currently selected element is shown (4). The values of properties in the list are editable with text fields, list boxes, radio buttons etc used to display and set values. Pounamu/Thin allows users to specify they want this property list always visible in a frame to the right of their browser window for the currently selected shape, or to be hidden when not in use.

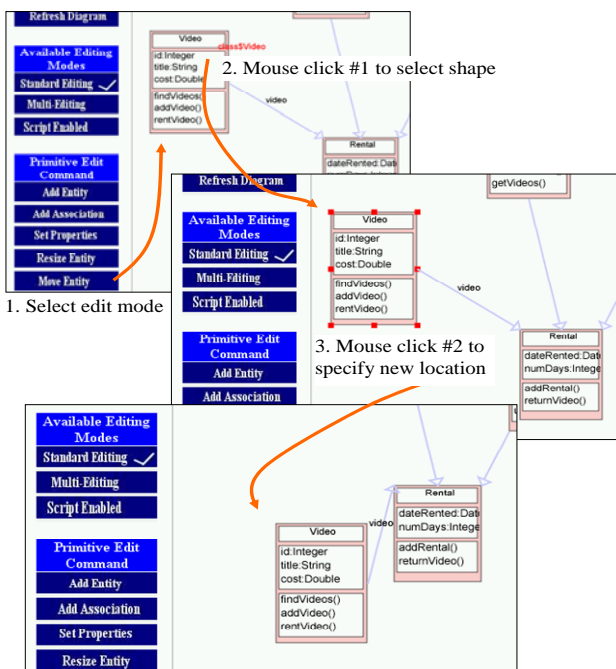


Figure 6. Example of moving a shape.

In this interaction mode for Pounamu/Thin, all editing operations are sent to the Pounamu/Thin web server and sent on to the shared Pounamu server for processing on the diagram data structure. All editing and semantic constraints for the diagram are implemented by the server. Constraints might include automatic layout of shapes, some automatic creation or deletion of shapes and connectors in response to edits, and others semantic constraints on allowable editing actions.

As all diagram manipulations are done on the server-side, editing operations such as moving and resizing shapes require several interactions with the diagram via the browser window. Users firstly indicate they want to move an item by selecting Move or Resize Entity, as shown in Figure 6 (1). They select the element to manipulate by a mouse click (2). They then indicate the position to move it to (or place to resize selected corner to) by clicking in the location desired (3). The specified movement or resize operation is enacted by Pounamu/Thin and the modified diagram is redrawn. Similarly, connecting two shapes requires selection of the connector type, selection

of the first shape to connect and then selection of the end shape to connect, each with a mouse click.

Collaborative diagram editing is supported with multiple users able to connect to the Pounamu/Thin web server and specify the same tool/project/diagram to view and edit. Concurrent editing of the same diagram element is prevented by the Pounamu/Thin server only allowing one user at a time to select a shape. Items being edited by another user are indicated as “locked by ...” awareness highlighting whenever the diagram is redisplayed.

## 4.2 Edit Buffering

From initial experiences with Pounamu/Thin we identified a need to allow users to make a small number of “transactional edits” to their copy of the diagram only. They then commit these edits as one unit to the shared Pounamu diagram. Other users don’t see these “in progress” edits of others until all are committed. To do this we developed a second interaction model for SVG-rendered diagrams that supports edit buffering. In this model each user has their own web server session with their own copy of the diagram (in SVG format). This approach is illustrated in Figure 7.

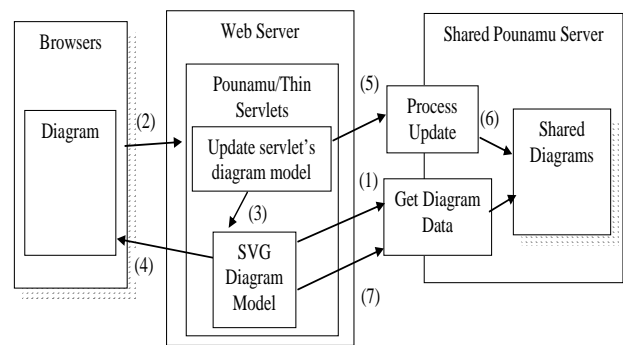


Figure 7. Edit buffering in servlet model.

In this approach, the diagram contents are fetched from the shared Pounamu server (1) and each user’s web server session caches a copy of the diagram. User interactions with the diagram are processed by the Pounamu/Thin servlets (2) which update the user’s copy of the SVG diagram contents in the servlet (3). The updated diagram is then redisplayed (4). At some point the user will ask for their edits to be “merged” into the shared diagram held by the Pounamu server (5), which processes the updates as a transaction on the shared diagram (6). The servlet then fetches the contents of the diagram back from Pounamu, as it may contain edits made by other uses to the same diagram as well as their own (7), and the diagram is redisplayed.

Figure 8 illustrates an example of using edit buffering on a diagram. The user changes the current editing “style” to “Multi-Editing” (1), resulting in an extra set of menu items being shown at the bottom left of the browser window. The user then makes an edit to the diagram e.g. resizing the “Customer” class. Rather than send this edit to the Pounamu server, this user’s Pounamu/Thin indicates the edit is “buffered” i.e. remembered but not yet applied to the shared diagram, by a green dashed highlight (2).

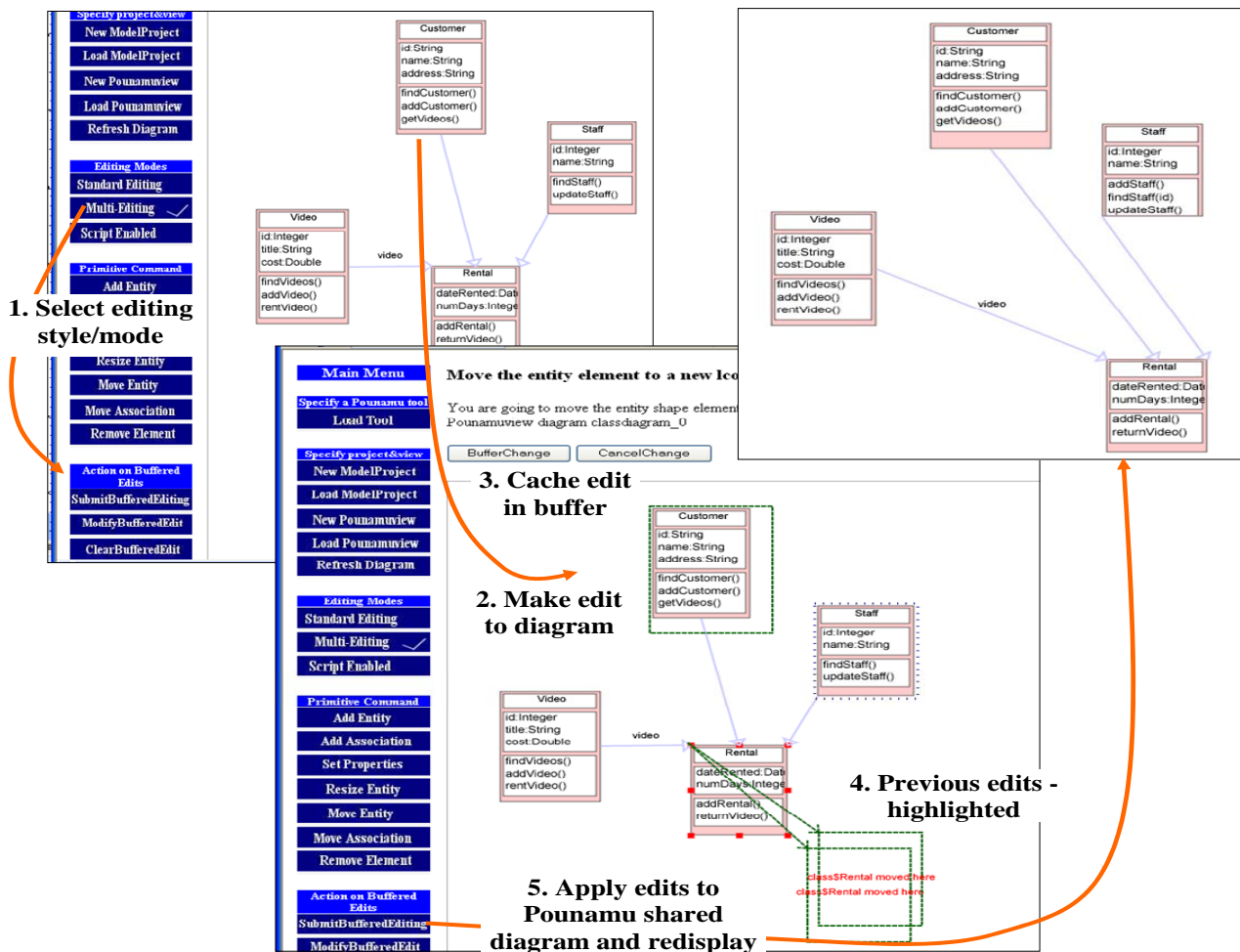


Figure 8. Examples of SVG-based thin-client editing with edit buffering.

The user can throw away the change or retain it in the current edit buffer (3). Further edits can be made e.g. changing the properties of the Staff class shape (indicated by dashed outline of this class shape); two moves of the Rental class shape (4), and so on. All of these “pending diagram updates” are highlighted in the diagram.

When the user has finished making a sequence of edits to the diagram, they select the Submit Buffered Editing operation. This sends the set of edits to the Pounamu server which updates the shared diagram data structures. The updated diagram is redisplayed in the browser (5). Updates made by other users will also be shown in the diagram when it is re-rendered. Some of these may have conflicted with the set of editing operations submitted by the user. In this case, the users edits will either over-ride them or be rejected by the Pounamu server, depending on semantic constraints specified for the tool.

### 4.3 Client-side Scripting

Both of the previous interaction models with the Pounamu/Thin diagramming application require sending user interactions to the Pounamu/Thin web server after each editing operation. This can be quite clumsy for some operations, particularly moving and resizing diagram elements and adding or moving connections between shapes. To overcome this, a third interaction model we developed for the SVG client uses web servlet-generated

ECMA scripts to implement move, resize and connect operations in the browser plug-in. This approach is illustrated in Figure 9.

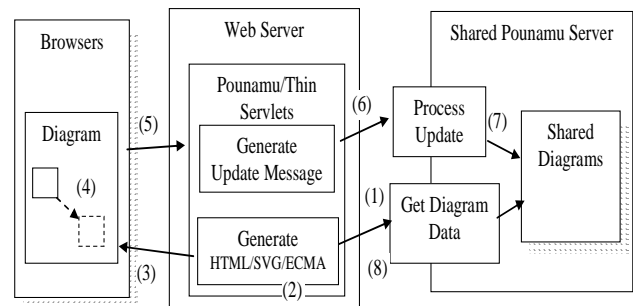


Figure 9. Client-side scripting interaction model.

In this approach, the diagram contents are fetched from the shared Pounamu server (1) and cached for each user. In addition to HTML and SVG content, the servlet generates ECMA scripts (2) which it passes with the HTML and SVG to the web browser client. These scripts program the browser plug-in to provide the client-side editing behaviour (3). When the user moves, resizes or connects shapes, these edits are implemented in the browser and not sent to the web servlet (4). After several edits the user then indicates they want the client-side edits reflected in the shared diagram and a summary of the edits is sent to the servlet (5). Appropriate update

messages are then sent to the Pounamu server (6) where the shared diagram is updated (7). The diagram content is then refetched by the servlet (8) and SVG regenerated by the servlet to reflect both the client-side updates and any other user updates.

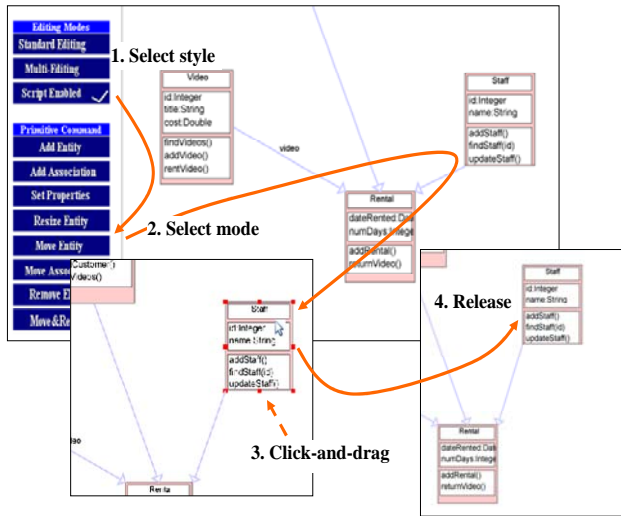


Figure 10. Example of client-side scripted based editing.

Figure 10 shows an example of diagram manipulation with client-side script editing. The user chooses “Script Enabled” (1) and the diagram is reloaded, with ECMA script sent to the browser to configure the SVG plug-in appropriately. When the user selects a Move operation (2), interactions with the diagram are detected and handled by this script instead of being posted to the web servlet as in the previous examples. For a move, the user clicks on the shape (3), drags the mouse and releases the mouse button (4), as with conventional thick-client direct manipulation tools. The diagram elements are moved, in this case the Staff class shape and its association connector line to the Rental class shape. Further move operations can be done in the browser by the user. These are not reflected in the shared Pounamu diagram until the user selects another operation e.g. Add Entity, Resize, Connect Entity etc.

## 5 Design and Implementation

Figure 11 illustrates the architecture of Pounamu/Thin. The Pounamu tool runs on a host and behaves as the application server/database manager. Tool specifications are loaded from XML repositories and the design tool configured from these. Multiple views of a model are provided by Pounamu, with the diagram views the point of interaction for users. View data can be converted into an XML format or a GIF format.

Pounamu uses a set of “Command objects” to represent edits to be made to a view (or model) elements. These have execute(), undo(), redo() methods which when invoked carry out the specified modification of Pounamu data structure state and re-render affected views. Command objects that represent view edits include AddShapeCommand, AddConnectorCommand, MoveShapeCommand, DeleteShapeComment and SetPropertyCommand. These command objects can be created and be sent to a Pounamu view programmatically in order to modify it. We have provided an RMI API to enable Pounamu views to be created and modified remotely via command objects, and lately have extended this with a SOAP-based web service API.

Pounamu/Thin is a set of web components hosted by an Apache web server. A set of Java Servlets provide the diagramming interface, and include user login, view manipulation (create, display etc), diagram rendering and manipulation, and diagram shape or connector property editing facilities. A set of JavaBeans are used to provide various support infrastructure, including GIF-based diagram rendering, SVG-based diagram rendering, diagram editing by Pounamu Command object construction, a copy of the SVG diagram data per user, and an SVG edit command cache per user.

Users interact with the Pounamu/Thin diagramming tools via standard web browsers. However, if SVG diagram rendering is desired, an SVG plug-in must be present in the user’s browser. Multiple users can view and edit the same diagram simultaneously. The web component servlets provide co-ordination via element locking and sequencing of Command message sending to the single Pounamu tool acting as the shared application server.

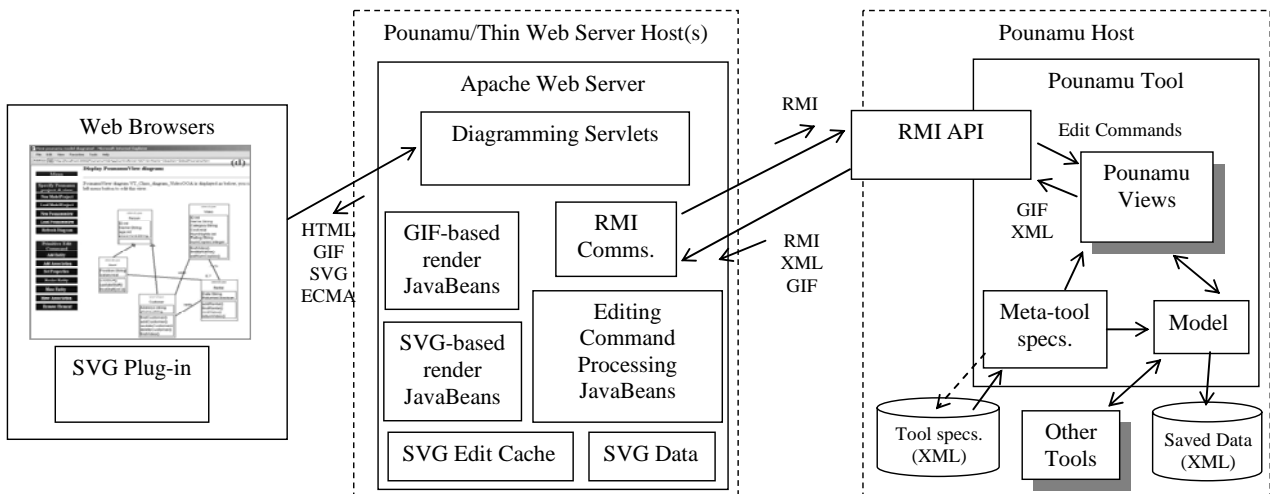


Figure 11. Architecture of Pounamu/Thin.

Integration with other software tools is provided by Pounamu, typically using XML-based data exchanges. In addition, some users can choose to edit Pounamu views using the existing thick-client diagram editor. In this case, they run a version of Pounamu on their own workstation with a copy of all model and view information they share. The SOAP API is used to synchronise their views with the “shared” views used by the thin-client diagram editing web components. We don’t discuss this approach further in this paper, but note that it is possible with our architecture to mix users using both Pounamu/Thin thin-client and thick-client Pounamu diagramming of the same shared view.

We chose to implement our Pounamu/Thin thin-client diagramming architecture using Java Servlets and RMI communication to the Pounamu tool application server. Pounamu itself is implemented in Java and provides a comprehensive RMI interface. Java Swing was used to implement Pounamu’s thick-client view editors. We found it easiest to add a plug-in component to Pounamu to support generation of GIF images from its diagram views, as this was simple to do with its Swing components. This was used to provide the GIF images of views for the GIF-based rendering of diagram components. This facility is also currently used by the thick-client tool for both printing Pounamu diagrams and copying them to other applications, which turned out to be a nice side-benefit of using this approach.

The SVG diagram renderer and editor is implemented quite differently to the GIF renderer. The latter simply requests that Pounamu return a GIF image of a view each time a view is changed. It then generates HTML for the view which includes editing command buttons, property sheet labels and text fields, and the GIF image. The SVG renderer instead requests an XML-encoded version of the view from the Pounamu server. It then traverses the XML and generates an SVG encoding of the Pounamu shape and connector objects in the view XML, along with HTML for view and property editing as per the GIF-based renderer. The SVG is rendered by the browser plug-in, resulting in more polished, detailed diagram appearance than the GIF-based version. We implemented the SVG translator in Java rather than using XSLT transformation scripts as we found using Java to provide much faster performance and because this process required quite complex algorithms. The multiple edit buffer facility uses the web component-stored SVG XML data, meaning buffered edits do not need to be sent to the Pounamu application server and are hidden from other users until committed. This wasn’t possible to do with the GIF-based version as it has to use the Pounamu application server to generate diagram GIF images, whereas the SVG version generates SVG data in the web components.

Pounamu views are saved to an XML format by the tool in order to support both saving and loading views and also to support thick-client editing of views via heavy weight distributed message passing between multiple instances of Pounamu. In addition, Pounamu view editing Command objects can also be converted to and from an XML format, which was done in earlier work to support

thick-client Pounamu tool collaborative view editing. We used both of these facilities to support the editing of Pounamu views and to support the SVG-based view rendering. The Pounamu/Thin diagram editing facilities convert user interactions with diagrams and property forms into XML-format Pounamu Command objects. These are sent by the RMI API to the Pounamu tool acting as Pounamu/Thin application server and are run on the appropriate view. We synchronise access to the Pounamu application server in the servlet components to ensure only a single user’s list of Command objects are exchanged with Pounamu at a time, to ensure consistent transactional behaviour.

## 6 Evaluation

We have carried out two evaluations of our prototype Pounamu/Thin thin-client diagramming system, in addition to reviewing and comparing our evaluation results to those of other thin-client diagramming tools.

Our first evaluation was a user survey via a questionnaire of nine experienced UML users – all either senior academics, experienced industry software designers and post-graduate students. They were asked about their experiences using the thick-client Pounamu-implemented UML diagramming tool, the GIF-based thin-client UML tool, and the SVG-based thin-client tool with multiple edit buffering. A single UML tool was specified in Pounamu and the exact same specification used in all three tools for the survey. The UML tool provided relatively complete class diagrams, collaboration diagrams, sequence diagrams and deployment diagrams.

A set of simple single-user and collaborating user UML modelling tasks was set for these users. The single user tasks included taking a simple software design, an on-line video rental system, and building structural and behavioural models for the system using each of the UML tools. Users were split into three groups of three, each group using the three UML tools (thick client, GIF and SVG) in different orders. The collaborative tasks we set for groups of three users was to review a design for the video system and to make simple enhancements to it.

Feedback from this survey of users of the thin-client tools indicated that the users perceived they provided the same facilities as the thick-client version. However, they noted that the interfaces clearly used a combination of page-based and frame-based web browser metaphors rather than overlaid windows as in the thick-client tool. Feedback on user interaction in the thin-client tools was positive, with users able to create and modify diagram components easily. The learning curve for inexperienced users of Pounamu tools was found to be less for users of the GIF-based thin-client tool. Users preferred the rendering of diagrams in the SVG tool rather than the GIF one. The buffering facility of the SVG-based tool confused some users while others liked it, particularly when collaboratively editing diagrams with others. All users requested the client-side scripting support for direct manipulation of SVG content, specifically for moving and resizing shapes and adding shape connectors.

Users perceived that the thin-client tools do not provide as good awareness facilities as the thick-client Pounamu tool currently does (Mehra et al, 2004).

Our second evaluation was a cognitive dimensions evaluation of the three versions of our UML diagramming tool. The Cognitive Dimensions (CD) framework provides a way of assessing various characteristics of visual languages and tools along a variety of usability “dimensions” (Green, 1989). We compared the Pounamu/Thin tool characteristics to those of the original Pounamu thick-client tool to determine their key similarities and differences. We summarise the results of our CD evaluation below.

- *Viscosity.* The thin-client diagrams are more “viscous” i.e. harder to change, than their thick-client counterparts. This is particularly so for moving and resizing operations that require multiple web browser interactions by users. This is the characteristic usually cited as against the use of thin-client diagramming approaches.
- *Visibility and Juxtaposability.* The thick-client UML modeller allows multiple views to be displayed side-by-side or accessed via both a tree and tabbed panes. The thin-client tools require multiple web browser windows displaying different views to be opened and positioned on the user’s display. A simplified list of views is provided for the user to select from.
- *Hard mental operations.* The learning curve of the thin-client tools is less in many respects to the thick-client tool. It presents interaction options in more explicit ways, such as always-visible buttons and frame-based property editing, rather than pop-up context-sensitive menus in the thick-client version. However, some operations such as moving and resizing require non-intuitive sequences of operations.
- *Hidden dependencies and View support.* Multiple views of UML model elements are supported in both kinds of diagramming tools and dependencies between view elements and model elements are via shape and connector names in both thick- and thin-client tools. Pop-up menus on thick-client shapes allow access to linked information, including other views they appear in, but the thin-client tools currently require access via buttons and non-contextual option lists.
- *Consistency.* All three UML tools discussed use the same visual representation and editing operations. The SVG and thick-client diagrams are rendered with scalar graphics and these tend to look better than the GIF ones. The button-based, page-organised approach to editing and view management operations takes up more screen space in the thin-client versions but provides users with a simpler, less complex interface than the many thick-client tool pop-up and pull-down menus.
- *Closeness of mapping.* The thin-client diagramming tools adopt page-based diagram viewing and interaction, where each user interaction causes a POST to the web server and then refresh of the screen, compared to the drag-and-drop and pop-up menu/selection-based approach of the thick-client tool. The thin-client’s page-based approach thus

supports less direct-manipulation activity but fits a user’s mental model of web page post/display metaphor common to all web applications.

- *Progressive Evaluation.* The SVG thin-client diagram tool’s buffered input approach supports user-controlled editing transactions, allowing the user to indicate when a set of edits are complete and they want them actioned on the shared diagram vs their own copy of the diagram. When performing collaborative work, the thin-client tools require user-directed refresh of pages to show other’s work. In contrast, the thick-client Pounamu UML tool uses push-based view update and redisplay.

In summary, evaluations by ourselves of our thick- and thin-client UML diagramming tool and by others of similar thick- and thin-client tools indicates that both approaches provide suitable support to diagramming and that user preferences lean to using one or the other more than any other consideration. Most users in our evaluation of the same Pounamu UML diagramming tool felt that they would be happy to use any of the three approaches, the thick-client using Java Swing, the GIF-based thin-client version or the SVG-based version. Novice Pounamu users suggested that the learning curve for the thin-client tools was less than the thick-client one and that it was an advantage that they needed no installation and configuration to use. The buffering of edits in the SVG version received mixed reaction. It may be the case that further experience with it is needed and it can be turned off if not wanted. Response time of the thin-client tools was found to be acceptable despite when using full diagram and page refresh after every editing operation. Client-side scripting of some operations e.g. highlighting diagram elements in the SVG plug-in, significantly improves some aspects of response-time and provides a more common direct manipulation model for editing. Currently changing a tool specification can only be done with the thick-client tool designer, seen as a disadvantage by some users if thin-client editing of the resultant tool was desired.

## 7 Summary

We have developed a thin-client diagramming extension for a meta-tool, allowing any specified tool to have either the original thick-client editing or a web-based thin-client editing approach. Our extension is a set of Java servlets implementing thin-client diagram rendering in either GIF or SVG formats with page-based diagram display and editing interaction. Limited client-side scripting for direct manipulation support and per-user multiple edit buffering are supported as user-configurable options for the SVG rendered diagrams. We use the original meta-tool interpreter to provide a shared application server for the thin-client tools, and multiple user diagram viewing and editing is supported using this conventional web-based architecture. Evaluation of our thin-client and thick-client tools indicates both provide equivalent diagramming facilities and the thin-client versions do provide acceptable response-time and user interaction. Pounamu/Thin has limited advantages in the areas of no

installation overhead, less of a learning curve and web-based multiple user diagram editing support.

The current version of Pounamu/Thin has several advantages compared to the thick-client version. Technical staff need only to provide support for the server(s), which minimizes the maintenance effort and avoids circulating versions of the system. Other benefits could include lower cost for hardware, as the Pounamu and web server(s) can be installed on high-end computer(s) and the Pounamu/Thin diagramming interface requires only low-cost machines equipped with simple browsers. Using web servers for deployment has a side effect of providing a centralized, implicit a history (log-file) of all triggered events. A future direction of our research will investigate how this information can be used for several measurements and give insights in how often the first design was modified or by using time recorder based on the log-files. We also plan to enhance the multi-user collaboration support with various awareness and locking features as used in our thick-client collaborative work tools (Mehra et al, 2004; Grundy et al, 2000).

## 8 References

- Bentley, R., Horstmann, T., Sikkil, K., and Trevor, J. (1995): Supporting collaborative information sharing with the World-Wide Web: The BSCW Shared Workspace system. *Proc. of the 4th International WWW Conference*, Boston, MA, December 1995.
- Chalk P. (2000): Webworlds-Web-based modeling environments for learning software engineering, *Computer Science Education*, vol.10, no.1, 2000, pp.39-56.
- Ebert, J., Siittenbach, R., Uhe, I. (1997): Meta-CASE in practice: A case for KOGGE, *Proc. 9th International Conference on Advanced Information Systems Engineering*, LNCS 1250, Barcelona, Spain, Springer-Verlag (1997) 203-216.
- Ferguson, R.I., Parrington, N.F., Dunne, P. Hardy, C., Archibald, J.M. and Thompson, J.B. (2000): MetaMOOSE - an Object-Oriented Framework for the construction of CASE tools, *Information and Software Technology*, vol. 42, no. 2, January 2000.
- Gordon, D., Biddle, R., Noble, J. and Tempero, E. (2003): A technology for lightweight web-based visual applications, *Proc. of the 2003 IEEE Conference on Human-Centric Computing*, Auckland, New Zealand, 28-31 October 2003, IEEE CS Press.
- Graham T.C.N., Stewart, H.D., Kopae, A.R., Ryman, A.G., Rasouli, R. (1999): A World-Wide-Web architecture for collaborative software design, *Proc. of the Ninth International Workshop on Software Technology and Engineering Practice*, IEEE CS Press, 1999, pp.22-29.
- Green, T.R. (1989) Cognitive dimensions of notations, *People and Computers V*, Sutcliffe, A. and Macaulay, L. Eds, Cambridge University Press, 1989.
- Grundy, J.C., Mugridge, W.B. and Hosking, J.G. (2000): Constructing component-based software engineering environments: issues and experiences. *Information and Software Technology* 42, 2, January 2000, pp. 117-128.
- Iivari, J. (1996): Why are CASE tools not used?, *CACM*, vol. 39, no. 10, 1996.
- Kaiser, G.E. Dossick, S.E., Jiang, W., Yang, J.J., Ye, S.X. (1998): WWW-Based Collaboration Environments with Distributed Tool Services, *World Wide Web*, vol. 1, no. 1, 1998, pp. 3-25.
- Kelly, S., Lyytinen, K., and Rossi, M. (1996): Meta Edit+: A Fully configurable Multi-User and Multi-Tool CASE Environment, *Proc. of CAiSE'96*, Lecture Notes in Computer Science 1080, Springer-Verlag, Heraklion, Crete, Greece, May 1996, pp. 1-21.
- Khaled, R., McKay, D., Biddle, R. Noble, J. and Tempero, E. (2002): A lightweight web-based case tool for sequence diagrams, *Proc. of SIGCHI-NZ Symposium On Computer-Human Interaction*, Hamilton, New Zealand, 2002.
- Lyu, M. and Schoenwaelder, J. (1998): Web-CASRE: A Web-Based Tool for Software Reliability Measurement, *Proc. of International Symposium on Software Reliability Engineering*, Paderborn, Germany, Nov, 1998, IEEE CS Press.
- Mackay, D., Biddle, R. and Noble, J. (2003): A lightweight web based case tool for UML class diagrams, *Proc. of the 4th Australasian User Interface Conference*, Adelaide, South Australia, 2003, Conferences in Research and Practice in Information Technology, Vol 18, Australian Computer Society.
- Maurer, F. and Holz, H. (2002): Integrating Process Support and Knowledge Management for Virtual Software Development Teams, *Annals of Software Engineering*, vol. 14, no. 1-4, 2002, pp. 145-168.
- Maurer, F., Dellen, B., Bendeck, F, Goldmann, S., Holz, H., Kötting, B., Schaaf, M. (2000): Merging project planning and web-enabled dynamic workflow for software development, *IEEE Internet Computing*, May/June 2000.
- Mehra, A., Grundy, J.C. and Hosking, J.G. (2004): Adding Group Awareness to Design Tools using a Plug-in, Web Service-based Approach, *Proc. of the 6<sup>th</sup> Int. Workshop on Collaborative Editing Systems*, Chicago, 5<sup>th</sup> Nov 2004.
- Quatrani, T. and Booch, G. (2000): *Visual Modelling with Rational Rose™ 2000 and UML*, Addison-Wesley.
- Robbins, J., Hilbert, D.M. and Redmiles, D.F. (1998): Extending design environments to software architecture design, *Automated Software Engineering* 5 (July 1998), 261-390.
- Sun, J., Dong, J.S., Liu, J. and Wang, H. (2001): An XML/XSL Approach to Visualize and Animate TCOZ. *Proc. of the 8th Asia-Pacific Software Engineering Conference*, Macau SAR, China, December 2001, IEEE Press, pp. 453-460.
- Zhu, N., Grundy, J.C. and Hosking, J.G. (2004): Pounamu: a meta-tool for multi-view visual language environment construction, *Proc. of the 2004 International Conference on Visual Languages and Human-Centric Computing*, Rome, Italy, 25-29 September 2004, IEEE CS Press.