Software – Practice and Experience, vol. 37, no.12, Wiley, October 2007, pp. 1245-1283. © Wiley 2007

Experiences developing architectures for realising thin-client diagram editing tools

JOHN GRUNDY[1, 2], JOHN HOSKING[1], SHUPING CAO[1], DENJIN ZHAO[1], NIANPING ZHU[1], EWAN TEMPERO[1] AND HERMANN STOECKLE[1]

Department of Computer Science[1] and Department of Electrical and Computer Engineering[2], University of Auckland, Private Bag 92019, Auckland, New Zealand

{john-g,john,nianping,ewan,herm}@cs.auckland.ac.nz, zhuangcao@hotmail.com, dzhao@ist.psu.edu

ABSTRACT

Diagram-centric applications such as software design tools, project planning tools and business process modelling tools are usually "thick-client" applications running as stand-alone desktop applications. There are several advantages to providing such design tools as web-based or even PDA- and mobile phone-based applications. These include ease of access and upgrade, provision of collaborative work support and web-based integration with other applications. However, building such thin-client diagram editing tools is very challenging. We have developed several thin-client diagram editing applications realised as a set of plug-in extensions to a meta-tool for visual design environment development. In this paper, we discuss key user interaction and software architecture issues; illustrate examples of interacting with our thin-client diagram editing tools; describe our design and implementation approaches; and present results of several different evaluations of the resultant applications. Our experiences will be useful for those interested in developing their own thin-client diagram editing architectures and applications.

KEY WORDS: thin-client diagramming, software architecture, web and mobile user interfaces, CASE tools

INTRODUCTION

Diagrammatic applications include design tools for a variety of domains such as information structuring and browsing, concept sketching and discussion, project management, and software and user interface design. Examples of diagram types include general ones, such as trees for depicting hierarchies and graphs for network, relationship and dependency specification together with more specialised domain-specific notations, such as Gantt and Pert charts for project management, or UML diagrams for software design. Traditional diagramming tools for such applications are built using thick client, window-based interfaces that limit their use to desktop and laptop computers. This approach provides highly responsive diagram editing and viewing facilities, can leverage sophisticated thick-client interaction techniques and enables management of information on local workstations[15, 23, 6]. 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[26, 30, 20].

To improve access to these tools, thin client diagramming approaches (Baudisch1 et al 2004; Gordon et al 2003) have been proposed and prototyped. These typically use a web browser for viewing and sometimes editing of 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 include GIF images, SVG (Scalable Vector Graphics) renderings, and 3D VRML (Virtual Reality Modelling Language) visualisations. 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, limited install and update problems, and use of conventional web architectures to support multi-user collaborations. Devices such as wireless PDAs and mobile phones have also become very widely available and used in recent years. Many offer a range of sophisticated content including styled text, rich graphical images and full-motion video streaming. However, very few applications for mobile devices provide dynamic diagrammatic display and almost none interactive diagram editing.

We have been developing Pounamu, a meta-tool that provides very flexible diagram and meta-model specification techniques (Zhu et al, 2004). Pounamu is used initially to specify a new diagramming tool, and then interprets the specification to provide the new tool's functionality. Both the metatool and generated tools have a thick-client interface. We wanted to provide Pounamu users the ability to access diagrams via web browsers, PDAs and mobile phones. To this end the Pounamu/Thin extension provides a thin-client diagramming infrastructure allowing users of web browsers to view and manipulate Pounamu diagrams.

Web browser-based user interfaces are more restricted than thick client interfaces in the types of interaction they allow. To investigate the degree to which these restrictions can be overcome, we have developed support for different interaction styles in Pounamu/Thin. One uses a GIF format and is usable by all browsers, but has limited interactivity, and a second uses SVG, which requires a separate plug-in, but allows more interaction. We report on the effectiveness of these two styles of thin-client diagramming user interface. To explore the issues involved in making thin-client diagramming applications available on mobile devices like PDAs and mobile phones we have also extended Pounamu/Thin to provide such support.

We begin by motivating our research using examples of Pounamu diagramming tools. We then review related research in the areas of thick-client diagramming infrastructure, thin-client diagramming approaches, and small-screen diagramming applications. We then demonstrate by examples how users interact with Pounamu/Thin applications using web browsers and mobile devices. The architecture, design and implementation of Pounamu/Thin are then described in detail. We describe our experiences building diagramming tools with Pounamu/Thin, summarise the results of usability evaluations and discuss the strengths and limitations of our approach. We conclude with a summary of the key contributions of this research and development.

MOTIVATION

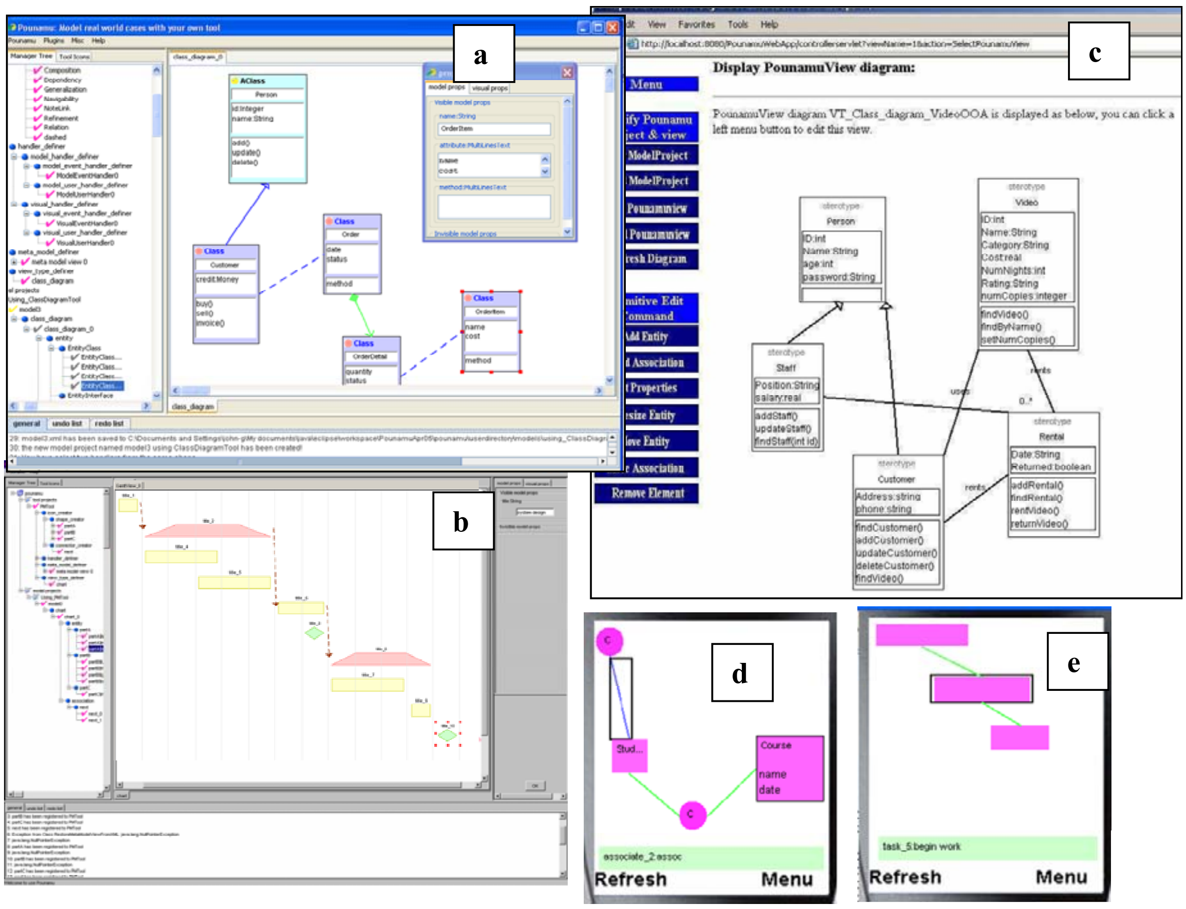
The traditional approach to providing diagramming tools is to use a thick-client program. Our thick-client meta-tool, Pounamu, provides a set of tools for designing shape and connector appearance, meta-model elements, views of meta-model elements using shapes and connectors, and event handlers for specifying semantic behaviour (Zhu et al, 2004). Figure 1 (a) shows an example diagramming tool generated by Pounamu, a Unified Modelling Language (UML) CASE tool, in use. 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 manipulable shapes (5) and shape elements. Figure 1 (b) shows a project management tool. These tools use thick-client interaction technologies and Pounamu must be installed on machines and periodically upgraded. A set of complex collaborative work supporting plug-ins have been developed which use peer-to-peer technologies to support synchronous collaborative diagramming and asynchronous version control.

Key advantages are highly responsive interaction when directly manipulating diagrams and the ability for tool developers to make full use of the graphics facilities available on the host computer. However, as noted earlier, this approach suffers the common thick-client application problems of: the need to install and update applications on all host computers; complex and difficult to learn user interfaces; and complex infrastructure to support multi-user collaborative work, especially synchronous diagram editing. As web-based user interfaces have become pervasive many applications that traditionally used thick-client interfaces have been reengineered with 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.

Our high level aim with Pounamu/Thin was to experiment with thin-client diagramming support for Pounamu-based diagramming applications to try and leverage these advantages in our tools. Similarly, to support pervasive access and mobile collaborative design, we wanted to make Pounamu modelling tools accessible on a range of mobile devices, such as mobile phones and PDAs. While the case for web browser based solutions is relatively easy to make, it is a more open issue as to whether the limited interfaces such mobile devices provide afford sufficient interaction capability for diagrammatic applications. We were interested in providing a proof of concept technology base to explore this, particularly as a number of industry partners we have been working with have expressed strong interest in providing mobile access to their toolsets. These real-world mobile applications included project planning (eg Gantt chart manipulation), health systems information capture and visualisation, and complex interactive enterprise reporting visualisations. In each case the desire was to allow quick access and update of strategic and tactical information in visually accessible and manipulable forms while end users were “in the field”.

Our approach to providing such mobile device interactive diagramming support was to provide an optional plug-in extension to Pounamu allowing users the choice of accessing generated diagramming tools via a web-based, thin-client infrastructure instead of a thick-client. Our lower level aim was to explore the technical and usability issues associated with building complex diagram-based user interfaces for both web browsers and mobile devices. The technical solution we chose to achieve this was to develop a set of web server components that support multi-user access to tools generated by Pounamu, through thin clients via web browsers and mobile devices.

Figure 1 (c) shows a web client user interface implemented by Pounamu/Thin. This requires no host machine installation (other than a generic web browser), has only a single Pounamu upgrade point on the shared web server, and uses a conventional multi-user, client-server web architecture to support multiple, collaborating users. Figure 1 (d-e) shows similar Class and Gantt chart diagrams viewed on a mobile phone. These thin-client diagramming applications provide the same range of diagram display and editing capabilities as the thick-client examples in Figure 1 (a and b). However, alternative mechanisms to access these capabilities are provided and the mobile device uses multi-level interactive zooming to manage the size and complexity of diagrams.



PounamuView diagram VT_Class_diagram_VideoOOA is displayed as below, you can click a left menu button to edit this view.

Figure 1. Examples of Pounamu diagramming tools.

Web browsers and in particular browsers running on mobile devices are constrained in a number of ways over desktop clients. This makes it particularly challenging to realize diagram-based interfaces on them. Some of these constraints include:

- Rendering of images such as those in Figure 1, often lacks the level of precision available in desktop application user interface libraries. This is particularly exacerbated with small screen devices.

- Manipulation of browser-displayed diagrams using only server-side technology means users can only click on the diagram and a location point sent back to the server for processing. This makes implementation of drag-and-drop style direct manipulation interfaces infeasible.

- Using client-side scripting to enable more interactive direct manipulation of diagram content exposes the application to a range of inconsistent “standards” for different browser scripting technologies or reliance on browser plug-ins.

- A range of web browser applications and versions, web browser plug-ins for diagram rendering, and a very wide range of mobile devices exist. This makes the issue of deploying applications onto devices running different browser applications, versions of applications, plug-ins and operating systems complex, a situation we were attempting to avoid by adopting a thin client approach.

- While web browsers running on desktop machines have the potential to provide high performance most mobile devices have relatively low processor speed, small memory and storage high network latency and bandwidth cost.

- Navigation between diagrams displayed in a web browser on a desktop computer or a mobile device require either multiple browser windows or hyperlinks. In addition, for mobile devices zooming and elision techniques are required in order to display large diagrams on small screens and yet still keep the displayed diagram meaningful.

- Ideally a thin-client diagramming application should support user preferences so that different users can specify different diagram content rendering approaches, usage of client side vs server-side scripting, zooming and navigation configurations.

- Collaborative work by multiple users on the same diagrammatic content should be supported with appropriate access control, group awareness and versioning of diagrams. For ease of deployment and use the provision of such capabilities should use standard web client-server technologies rather than custom peer-to-peer or other approaches.

RELATED WORK

Applications that provide thick-client diagramming support have been extensively investigated. Examples of software design tools include Rational Rose™ (Quatrani and Booch, 2000), Argo/UML (Robbins et al, 1998), and JComposer (Grundy et al, 2000). More general diagramming packages and authoring tools include MS Visio™, PowerPoint™ and Photoshop™. More domain-specific tools include project management tools, such as MS Project™; XML Schema editors, such as XML Spy™; and custom visual language tools, such as LabView™, Prograph (Cox et al, 1989) and Forms/3 (Burnett et al, 2001). These all 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 such design tools to different user's preferences and rapidly develop new design methods and tools has led to the development of meta-tools. Examples include MetaEDIT+ (Kelly et al, 1996), Kogge (Ebert et al, 1997), JViews (Grundy et al, 2000), MetaMOOSE (Ferguson et al, 2000), DiaGen (Minas and Viehstaedt, 1995), Vampire (McIntyre, 1995) and Escalante (McWhirter and Nutt, 1994). Despite the fact that each of these meta-tools provides a specification of the intended diagramming tool that could be interpreted to provide a thin-client version of the tool, none do so.

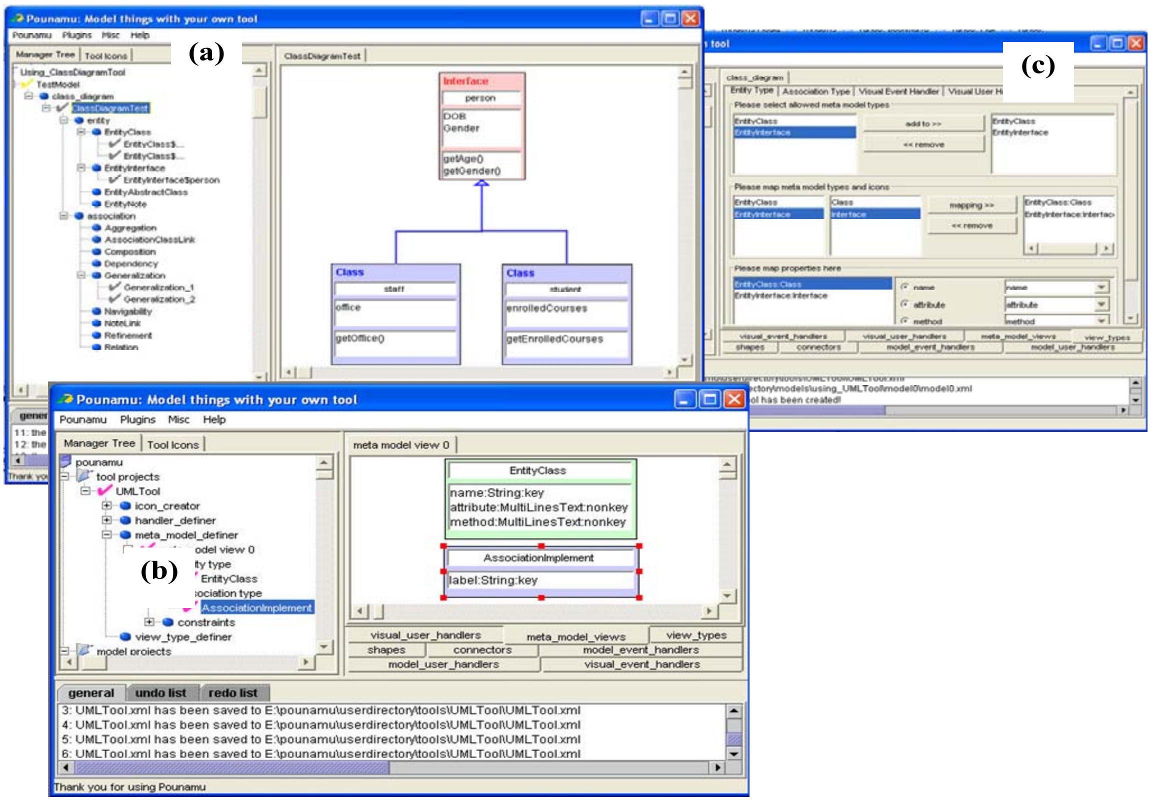
A wide variety of thin-client tools have been developed to exploit web-based delivery of information that have traditionally used non-diagram oriented thick-client applications. Some examples include: MILOS (Maurer and Holz, 2002; Maurer et al, 2000), a web-based process management tool; BSCW (Bently 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). A browser-based user interface for such applications allows users to access the tool via a URL, without the need for application download, install or update, at the expense of more sophisticated user login requirements. A consistent web interface look and feel across applications and browser-based integration of applications can also be facilitated. When diagrammatic content is supported, it is often limited to display-only, e.g. the display of UML diagrams derived from formal specifications in TCOZ (Sun et al, 2001). Most tools providing such thin-client diagram rendering have adopted custom algorithms and implementations to producing the diagrams with fixed diagram layout and appearance.

Most tools that provide web-based diagramming have used Java Applets or similar heavyweight browser plug-ins (Graham et al 1999). These have often suffered from reliance on particular plug-in versions and lack of consistency across multiple applications. Supporting collaborative work activities in such applications requires similar implementation effort as in thick-client tools. Some recent efforts at building true thin-client web-based diagramming tools include: 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 given a proof-of-concept demonstration that web-based interfaces can be used to both visualise and edit specific kinds of diagrammatic content. All of these have used custom approaches to realise the thin-client diagramming tool. Limited tailoring of notations is supported in Cliki but not the other tools.

Thin-client diagramming systems have focused on supporting standard web browsers on desktop or laptop PCs, rather than smaller screen mobile PDA or phone devices. However some mobile applications provide diagrammatic representations of complex information, including for: tourism and travel planning, map usage, and management (Luz and Masoodian, 2004; Masoodian and Budd, 2004; Rossel, 1999; Stephanidis, 2001;Wobbrock et al, 2002). Most have read-only diagrams that do not support editing. Some provide pan and zoom-based displays enabling users to focus in on areas of interest quickly and navigate complex information spaces. However these capabilities are limited to either the mobile device's standard characteristics or single points of zoom and focus. Several systems have been developed to assist in realising mobile, small-screen device user interfaces. Some provide overviews of complex information such as web pages using e.g zoomed-out copies of pages (Eisenstein and Puerta, 2001; MUPE 2006) or summarized versions of the page (Baudisch1 et al 2004; Chen et al, 2003). These "content outlining" approaches transform pages into sets of tiles (Buyukkoten et al 2000; Baudisch1 et al 2004) and while they assist navigation they are unsuitable for diagram representation. Some mobile browsers combine tiling with zooming, so that users can access individual tiles from an overview (Chen et al 2003; Eisenstein, 2000). These increase the amount of complex content that can be reasonably accessed and browsed on a small screen mobile phone. Approaches that collapse page content allow users to zoom into relevant areas by collapsing less relevant areas (Baudisch1 et al 2004) increasing their ability to access complex information that otherwise could not be presented on a single small screen.

A number of architectures have been developed to support "multi-device user interface" construction (Bonifati et al, 2000; Palm Corp., 2001; Grundy and Zhou, 2003; Van der Donckt et al, 2001; Marsic 2001). Techniques range from "clipping" services, i.e. translating content for conventional web browsers into other forms, to "generic" interface specifications, where a sinclge specification may be realised on a wide range of devices, including standard web browsers and various small-screen devices. However most of these application interfaces have been limited to text and form-based data rather than richer diagrammatic content, typically translating from an abstract XML format into a device-specific format. Any diagramming support is usually implemented in a custom fashion specifically for each mobile device type. While this allows some optimisations to be made in terms of content production, it has the disadvantage of not being able to leverage work on diagramming meta-tools and architectures.

As thin-client applications are generally client-server systems supporting multiple concurrent users, they provide a natural collaborative work-supporting infrastructure. Many collaborative work-supporting thin-client applications have been developed, including BSCW (Bently et al 1995), MILOS (Maurer et al, 2002; Maurer et al, 2000) and various UML design tools (Graham et al, 1999; Mackay et al, 2003). Most support collaboration via awareness of others' activities and limited shared editing of content. As clients very frequently request content for display and editing in the client browser, the thin-client application architectures facilitate the implementation of collaborative work access control, synchronisation and group awareness capabilities. As they use standard internet protocols they are easy to deploy in organisations with firewalls and policies that prevent peer-to-peer approaches. Diagrammatic thin-client interfaces thus have the potential to support both awareness and content sharing by multiple users. However, there are potential disadvantages as well. These include a limited ability to push other users' changes to one's web browser from the server and limited drag-and-drop manipulation of content with synchronous notification of change to other users.



3: UMLTool.xml has been saved to E:\pounamu\userdirectory\tools\UMLTool\UMLTool.xml
4: UMLTool.xml has been saved to E:\pounamu\userdirectory\tools\UMLTool\UMLTool.xml
5: UMLTool.xml has been saved to E:\pounamu\userdirectory\tools\UMLTool\UMLTool.xml
6: UMLTool.xml has been saved to E:\pounamu\userdirectory\tools\UMLTool\UMLTool.xml

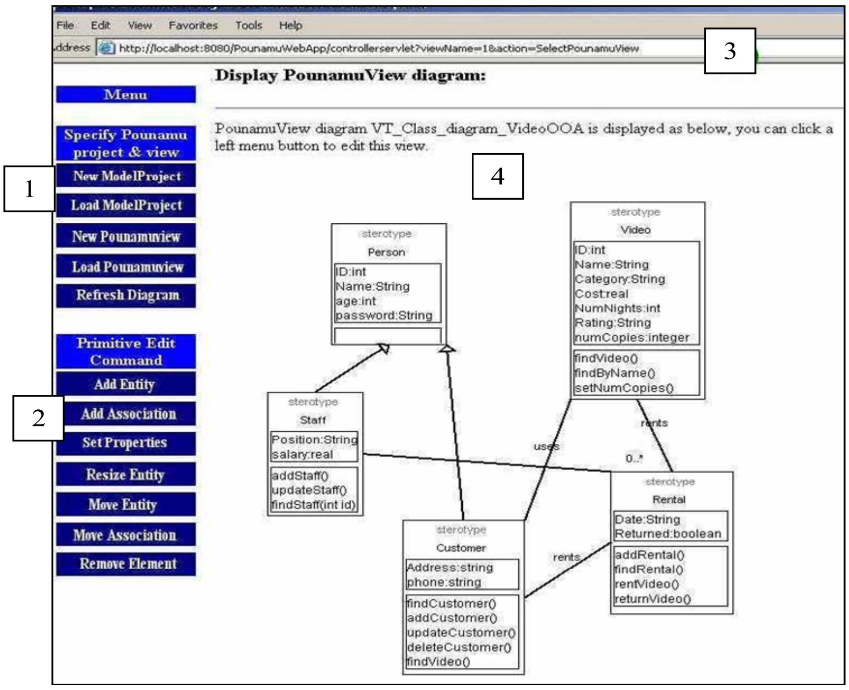
Figure 2. Examples of Pounamu meta-tool UML tool specifications.

OVERVIEW OF POUNAMU/THIN

In this section we illustrate the user interfaces of Pounamu/Thin via an exemplar application, a simple Unified Modelling Language (UML) design tool prototype. This tool, specified using Pounamu (Zhu et al, 2004), includes UML class, sequence, collaboration, and deployment diagrams. This is but one exemplar tool; as Pounamu is a metatool the tool's appearance and semantics can be changed on-the-fly by the user, or a completely new tool specified and used. We also emphasise that the Pounamu/Thin web components illustrated here are completely tool-independent i.e. no code changes are needed to support rendering and editing of any diagramming tool specified using our Pounamu meta-tool.

Figure 2 shows part of the specification of the UML tool using Pounamu. Figure 2 (a) shows the class icon shape being defined. Figure 2 (b) shows part of the UML tool meta-model specification (the entities, associations and various inter-relationships defined for the tool). Figure 2 (c) shows specification of the class diagram view (mapping icon shapes to entities etc). The Pounamu meta-tools are currently all realised via thick-client interfaces. Figure 1 (a), shows the generated UML tool also realised as a thick-client desktop application by Pounamu. To realise a tool the user simply selects its specification and Pounamu on the fly creates an appropriate editing environment for that specification. Users may change the tool specification while their editing tool is in use e.g. modify or add icon shapes; change meta-model types etc. These changes are reflected immediately in the running editing tools (Zhu et al, 2004).

Figure 3 shows the web client diagramming interface provided by Pounamu/Thin when viewing and editing an example UML class diagram with the above UML tool specification This web browser-based user interface includes a set of buttons to control the project (add views, open views) (1); a set of buttons to manipulate the diagram (add shapes and connectors, move and resize elements, edit element properties) (2); a message area to provide feedback to users (3); and a diagram (4), which can be clicked on to manipulate it. A simple user authentication mechanism permits registered users to access and manipulate any Pounamu projects in a shared repository. This could readily be extended to project-specific or role-specific access.


http://localhost:8080/PounamuWebApp/controllerservlet?viewName=1&action=SelectPounamuView

Display PounamuView diagram:

PounamuView diagram VT_Class_diagram_VideoOOA is displayed as below, you can click a left menu button to edit this view.

Figure 3. Example of using Pounamu/Thin.

With such a browser-based interface, there are a variety of approaches we could have taken to allow users to select and manipulate diagram content. We have chosen three basic diagram interaction styles and users can move between them at any time via editing modes buttons. Each approach provides users a different "look and feel" and each has certain advantages and limitations compared to the others. The options are:

- *Fully server-side* interaction processing, where all interactions (i.e. clicks on the diagram) are passed to the server-side web components and onto the Pounamu meta-tool for processing. After updating the shared copy of the source diagram, the diagram is redisplayed in the user's web browser. This style is the least like the thick-client tools as there is no "drag and drop" and any diagram click is posted to the server with high latency and full diagram refresh. However, it is the most portable across browsers and requires no browser (client-side) scripting support.

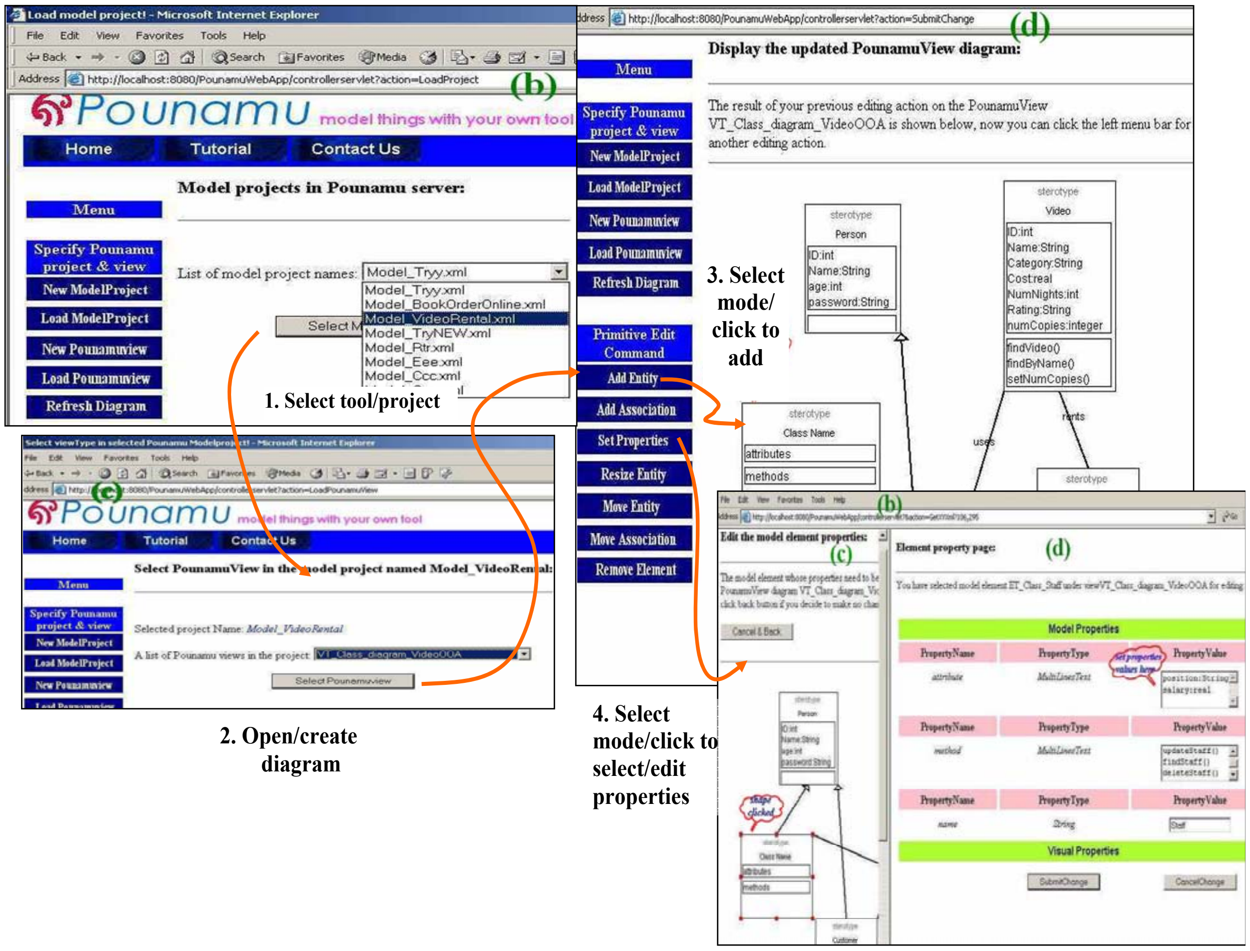
- *Buffered edits*, where a user's diagram edits are applied to their copy of the diagram only and cached on the server. These are subsequently sent on user request as a set of edits to the Pounamu server and applied as a unit on the shared copy of the source diagram. The diagram is then redisplayed in the user's browser. This isolates limited per-user changes and provides faster diagram refresh, at the risk of potential multi-user update conflicts.

- *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 applied as a unit on the source diagram. This approach is the most similar to interacting with Pounamu thick-client diagrams but requires complex browser (client-side) scripting that currently limits cross-browser portability.

Examples using these techniques are outlined in the following three sub-sections. Following this we explore interaction with the same examples using the mobile phone interface and, to demonstrate versatility, a prototype 3D interface.

Fully Server-side Interaction Processing

Figure 4 shows opening and modification of an example UML diagram in Pounamu/Thin. The user first selects the UML model project to use (1). This loads the tool specification and model 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, in this case class, sequence, deployment and component diagrams. Each diagram type has a different set of symbols, editing syntax and semantics. In this example, a user opens an existing class diagram (2) and then adds a class element to the diagram by selecting the Add Entity button (3), specifying the kind of element to add to the diagram (a new Class in this case), and clicking where the new element is wanted, whereupon a blank class shape is displayed.


Load model project! - Microsoft Internet Explorer

Address http://localhost:8080/PounamuWebApp/controllerservlet?action=LoadProject

Model projects in Pounamu server:

List of model project names: Model_Tryy.xml

Model_Tryy.xml
Model_BookOrderOnline.xml
Model_VideoRental.xml
Model_TryNEW.xml
Model_Rtr.xml
Model_Eee.xml
Model_Ccc.xml

Select viewType in selected Pounamu Modelproject! - Microsoft Internet Explorer

Select PounamuView in the model project named Model_VideoRental:

Selected project Name: Model_VideoRental

A list of Pounamu views in the project: VT_Class_diagram_VideoOOA

Address http://localhost:8080/PounamuWebApp/controllerservlet?action=SubmitChange

Display the updated PounamuView diagram:

The result of your previous editing action on the PounamuView VT_Class_diagram_VideoOOA is shown below, now you can click the left menu bar for another editing action.

Edit the model element properties:

Element property page:

PropertyName	PropertyType	PropertyValue
attribute	MultiLinesText	position:String salary:real
PropertyName	PropertyType	PropertyValue
method	MultiLinesText	updateStaff() findStaff() deleteStaff()
PropertyName	PropertyType	PropertyValue
name	String	Staff

Figure 4. Interacting with the fully server-side processing client.

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 that 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.

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. These might include automatic layout of shapes, automatic creation or deletion of shapes and connectors in response to edits, and 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 5 (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.

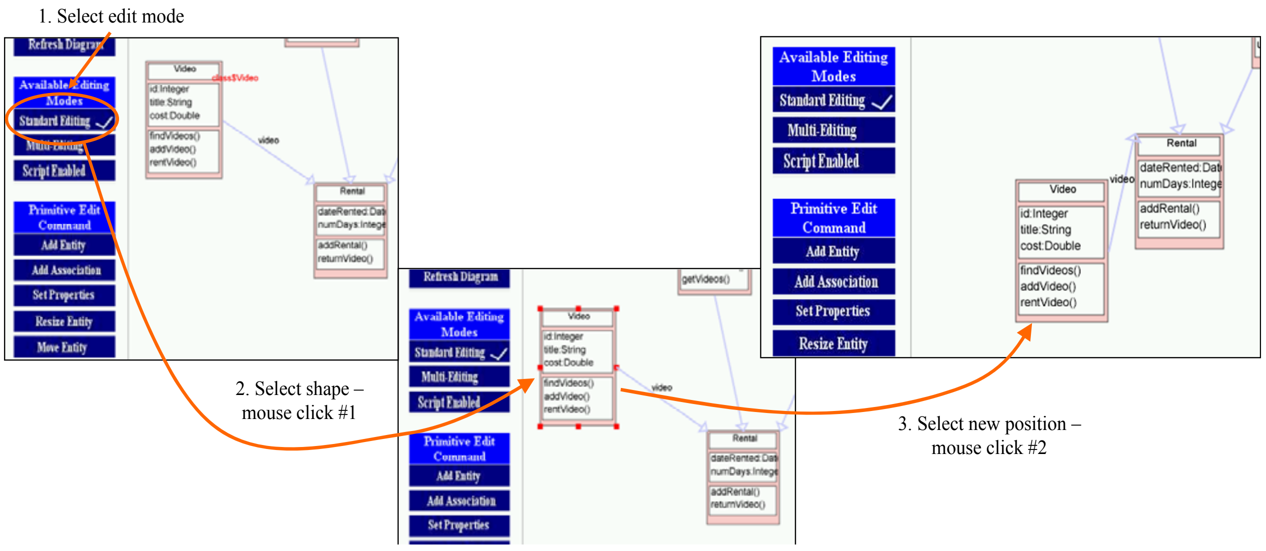


Figure 5. Example of moving a shape.

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.

Edit Buffering

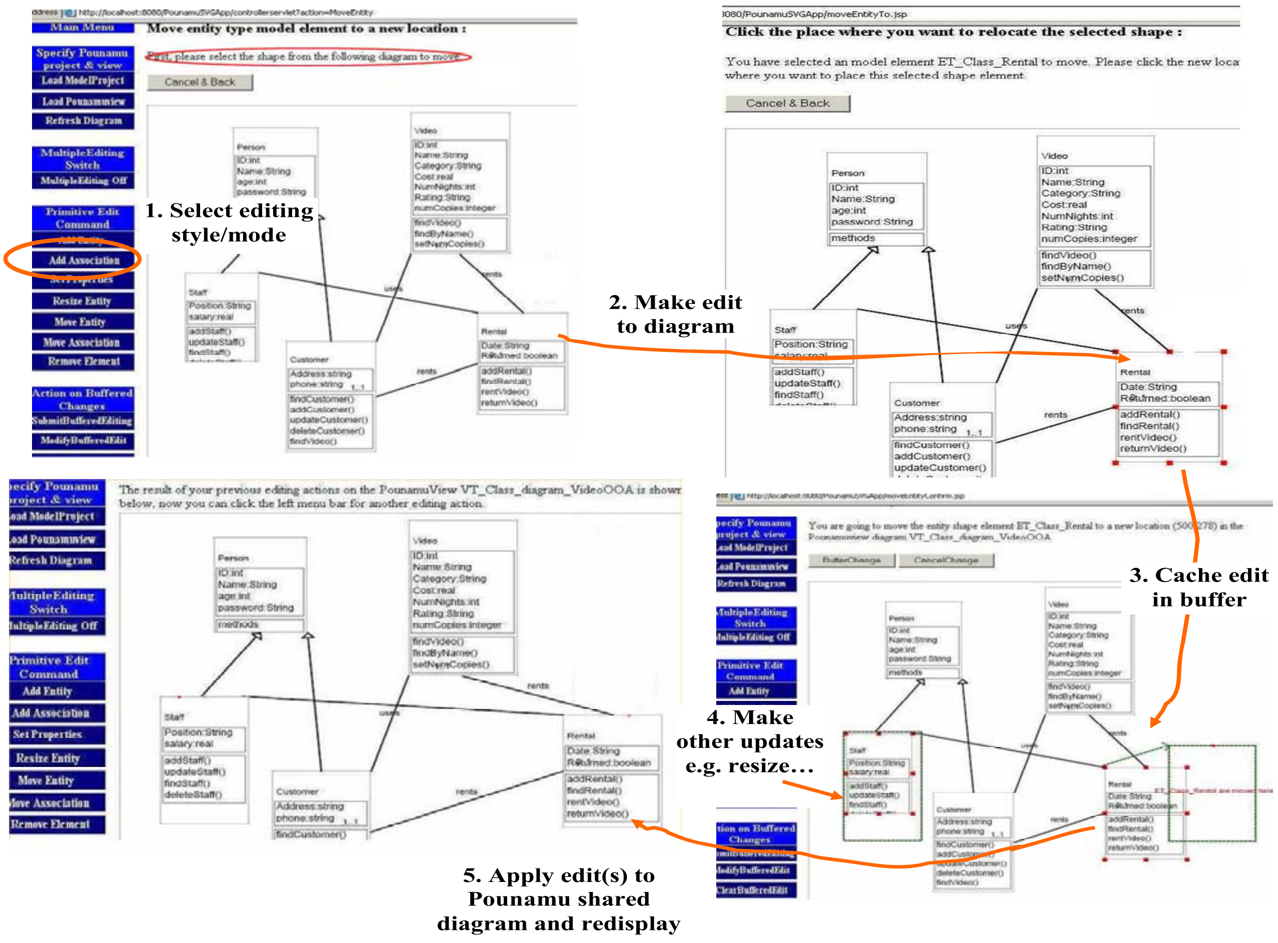
From initial user feedback on using Pounamu/Thin we identified a common user desire to make a small number of "transactional edits" to their copy of the Pounamu diagram only. They will then commit these edits as one unit to the shared Pounamu diagram. Other users don't see these "in progress" edits until all are committed.

To do this we developed a second interaction model that supports edit buffering. In this model each user has their own web server session with their own copy of the diagram. Figure 6 illustrates this. 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). The user can throw away the change or retain it in the current edit buffer (3). Further edits can be made e.g. resize of the Staff class shape (4). Such "pending diagram updates" are highlighted in the diagram, as indicated by dashed outline of the Staff class shape.

When users have 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 are also shown in the re-rendered diagram. Some of these may be in conflict with the set of editing operations submitted by the user. In this case, the conflicts are resolved by the Pounamu server by using semantic constraints specified for the tool.

Client-side Scripting

Both of the previous interaction models require frequent browser-server interactions. The latency introduced can turn traditionally easy and highly interactive operations in thick-client diagramming tools into quite clumsy operations on the thin client. This particularly the case for moving and resizing diagram elements and adding or moving connections between shapes. To overcome this, a third interaction model we developed for Pounamu/Thin web clients uses browser (client-side) ECMA scripts to implement move, resize and connect operations in the browser plug-in.



1. Select editing style/mode
2. Make edit to diagram
3. Cache edit in buffer
4. Make other updates e.g. resize…
5. Apply edit(s) to Pounamu shared diagram and redisplay
Move entity type model element to a new location :
First, please select the shape from the following diagram to move
Click the place where you want to relocate the selected shape :
You have selected an model element ET_Class_Rental to move. Please click the new location where you want to place this selected shape element.
The result of your previous editing actions on the PounamuView VT_Class_diagram_VideoOOA is shown below, now you can click the left menu bar for another editing action.
You are going to move the entity shape element ET_Class_Rental to a new location (500,278) in the Pounamuview diagram VT_Class_diagram_VideoOOA
Cancel & Back
BufferChange
CancelChange
Main Menu
Specify Pounamu project & view
Load ModelProject
Load Pounamuview
Refresh Diagram
MultipleEditing Switch
MultipleEditing Off
Primitive Edit Command
Add Entity
Add Association
Set Properties
Resize Entity
Move Entity
Move Association
Remove Element
Action on Buffered Changes
SubmitBufferedEditing
ModifyBufferedEdit
ClearBufferedEdit
Person
Video
Staff
Customer
Rental
uses
rents

Figure 6. Examples of SVG-based thin-client editing with edit buffering.

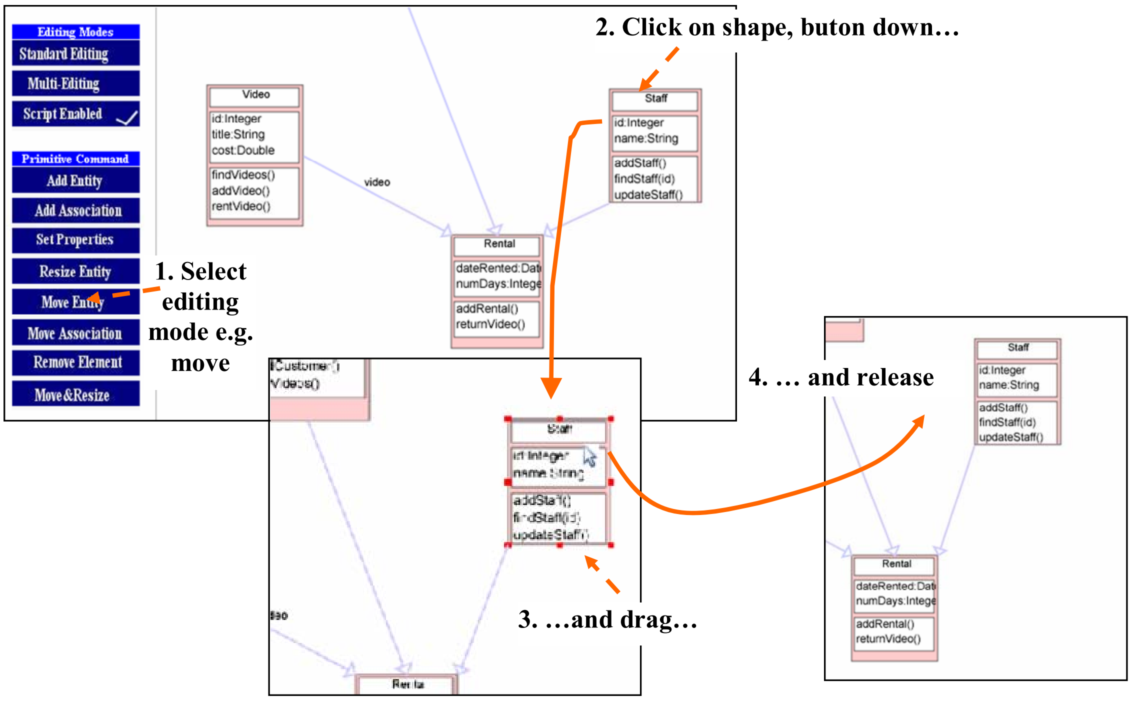


Figure 7. Example of client-side scripted based editing.

Figure 7 shows an example of diagram manipulation with client-side script editing. The user has choosen "Script Enabled" and the diagram has been reloaded, with ECMA script sent to the browser to configure the SVG plug-in. When the user selects a Move operation (1), interactions with the diagram are detected and handled by this client-side ECMA script instead of being posted to the web servlet as previously described. For a move operation on the diagram the user clicks on the shape to move, holding the mouse button down (2), drags the mouse to the desired new shape location (3), and then releases the mouse button (4). This set of interactions is just the same as with conventional thick-client direct manipulation tools like Pounamu. The diagram elements are moved, in this example 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.

Pounamu/Thin on a Mobile Phone Device

A number of industry partners we have been working with have indicated a desire for rich diagram-based user interfaces on PDAs and mobile phones for a range of application domains. To experiment with the feasibility of such interfaces we have developed prototype support for Pounamu/Thin diagrams to be viewed and edited using mobile phones and PDAs. Figure 8 (a) is the post logon Pounamu/Thin screen, showing a list of viewing and editing options. After selecting a diagram to display, an initial overview of it is generated, such as the class diagram overview shown in (b). Views may appear differently on different mobile devices as the overview is generated by proportionally shrinking original diagrams to fit them onto the small screens of mobile devices. The proportion is calculated according to the size of the selected Pounamu view to display and the screen size of the client device requesting a page. Detailed diagram contents are filtered out in overview mode because they are hard to see. Users can press direction keys to select diagram shapes and connectors. The status bar underneath displays certain details of the currently selected item, in this case the name and type of the item.

As web browsers typically run on PCs with large screens Pounamu/Thin diagrams displayed in web browsers are close to the fidelity of thick-client versions. With client-side scripting support user interaction with a web browser-based Pounamu/Thin diagram can be a similar experience to that of thick-client diagram tool interaction. However, PDAs and mobile phones are much more limited in their display and interaction capabilities. To help address the rendering and resolution limitations of most mobile devices, we developed a multi-level rendering and zooming capability. This facility allows users to define multiple representations for diagram elements at different levels of detail. Each diagram element can be separately selected and zoomed among the multiple levels. Automatic focus and context zooming of elements is supported as users navigate a large view. We use several navigation approaches to improve user interaction with diagramming tools on mobile devices. Button-panning let users quickly move around in a big diagram using mobile phone buttons. A floating zooming window allows users to easily and quickly pan to interesting areas from an overview of a large diagram.

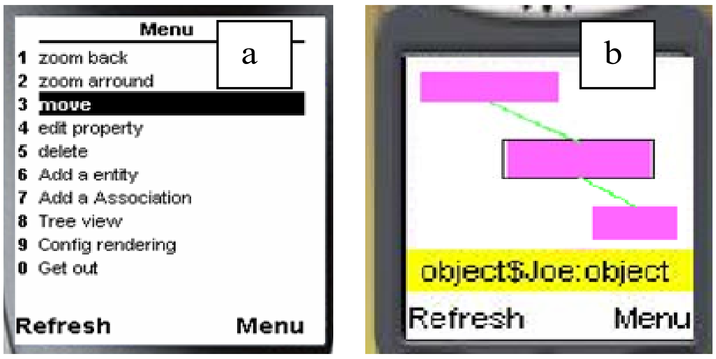


Figure 8. (a) main menu of Pounamu/Thin mobile tools; (b) overview of class diagram view on a colour phone.

Figure 9 shows an example of this multi-level zooming and pan navigation. To zoom in/out diagram shapes to various detail levels, firstly a user selects a diagram shape from the model view, and then presses a pre-defined hot key to zoom into a particular level. In Figure 9 (a), all diagram elements are at "zoom level 1", providing a high level overview of the diagram but no details for each element. In Figure 9 (b), one object entity is zoomed in level 2, and two class entities are zoomed in level 3. The user can select individual elements and zoom them in or out, or can enable an auto-zoom facility that magnifies the selected item and its immediate neighbours while de-magnifying those further away. This forms a basic distortion-oriented display. As each Pounamu diagram type can have different element types and connectivity, different view types have different auto-zoom behaviours. To use pan navigation, a user selects pan navigation from the options menu or a hot-key button. They can zoom out to the top-level overview and a floating window indicates the current zooming area in the model view, as shown in Figure 9 (c). The floating window can be moved across the diagram using mobile phone keys and selecting a hot-key magnifies elements in the selected zooming area.

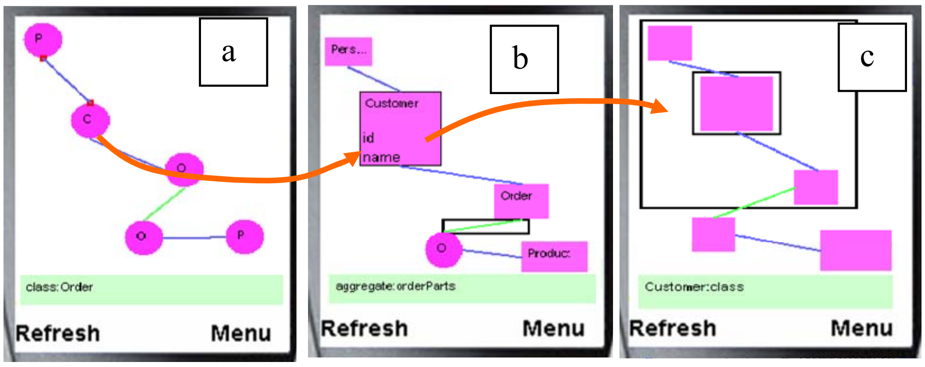


Figure 9. (a) zoomed in level 1; (b) shapes zoomed to different levels; (c) pan navigation to selected area.

Adding, moving, deleting, and editing items in a diagram require similar interactions through the generated Pounamu/Thin mobile device user interfaces. Firstly a user selects an item to manipulate, and then chooses a menu command or a hot-key indicating the action they want to perform on that item. For example, to move a shape, the user selects a shape, then repositions the shape using the mobile device's direction keys, and confirms the new position by pressing e.g. the Select hot-key on the device. Items are added to a place on the screen by moving the pan selection floating window and narrowing it down to an unoccupied space with hot-keys.

To edit properties of an item, the user selects the item, as shown in Figure 10 (a), then selects a hot-key or menu option to edit its properties. A list of the element's properties is displayed which the user edits with conventional mobile device interactions, as shown in Figure 10 (b). The user then selects the OK menu to submit the values back to server. After processing the user request, the server sends back an updated view.

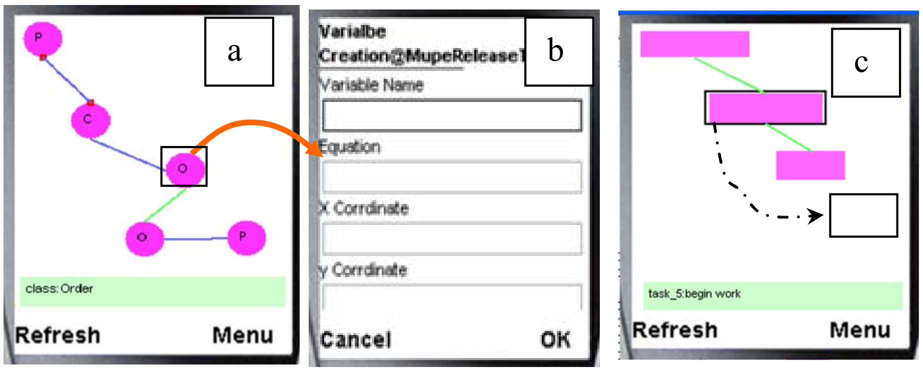


Figure 10. (a) select an element; (b) edit the element's properties; (c) and moving the element in Gantt chart.

One issue we repeatedly encountered when building multi-device user interfaces is the inability of generated interfaces to suit all potential users. In particular, when providing multi-level zooming capability on a mobile-hosted diagram, we have found that different users want different element renderings for different zoom levels. This, along with our desire to experiment with different mobile rendering approaches led us to support such user preferences by using a multi-user runtime Pounamu/Thin application configuration facility. Users can specify different diagram content, zoom, and navigation configurations via their mobile device. These individual configurations are stored in XML format in the Pounamu/Thin Server. At run-time the system obtains model information to display from the Pounamu application server and generates diagram views according to each user's configuration. Default user configurations are generated and these may be tailored to enhance users' display and interaction.

Figure 11 (a) shows an example of tool configuration via mobile user interfaces at runtime to specify different element appearance at different zoom levels. To configure diagram representations, a user selects the type of entities that they want to specify the appearance of and then the menu entry "Config rendering". The mobile server finds the user's Pounamu tool configuration in xml format and generates configuration user interfaces for it and the user selects the zoom level to specify a representation for. A diagram representation currently can be specified by using color, size, icon shape, and properties to show. Figure 11 (b) shows the result of a user configuring a UML class diagram view in which all diagrams are zoomed in level 1. Diagram representations are different from the one shown in Figure 5 (a) as different shapes have been selected for the icon representations.

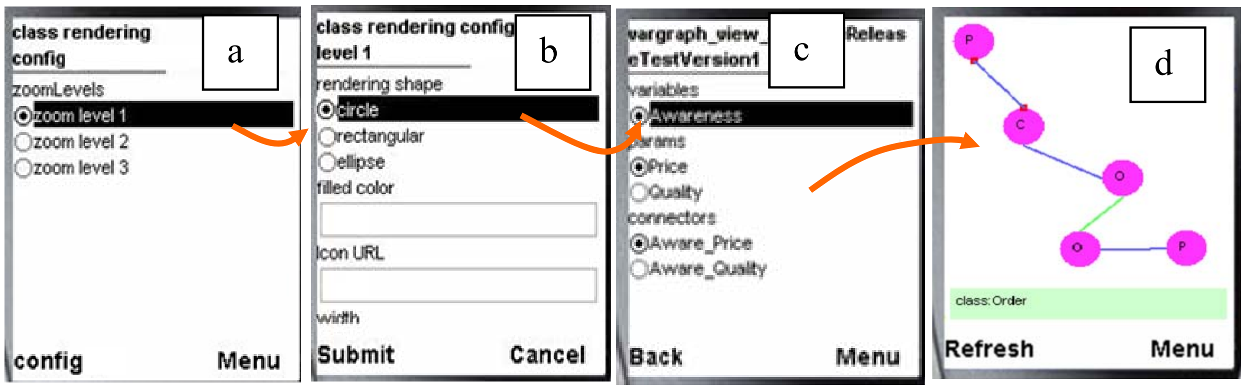
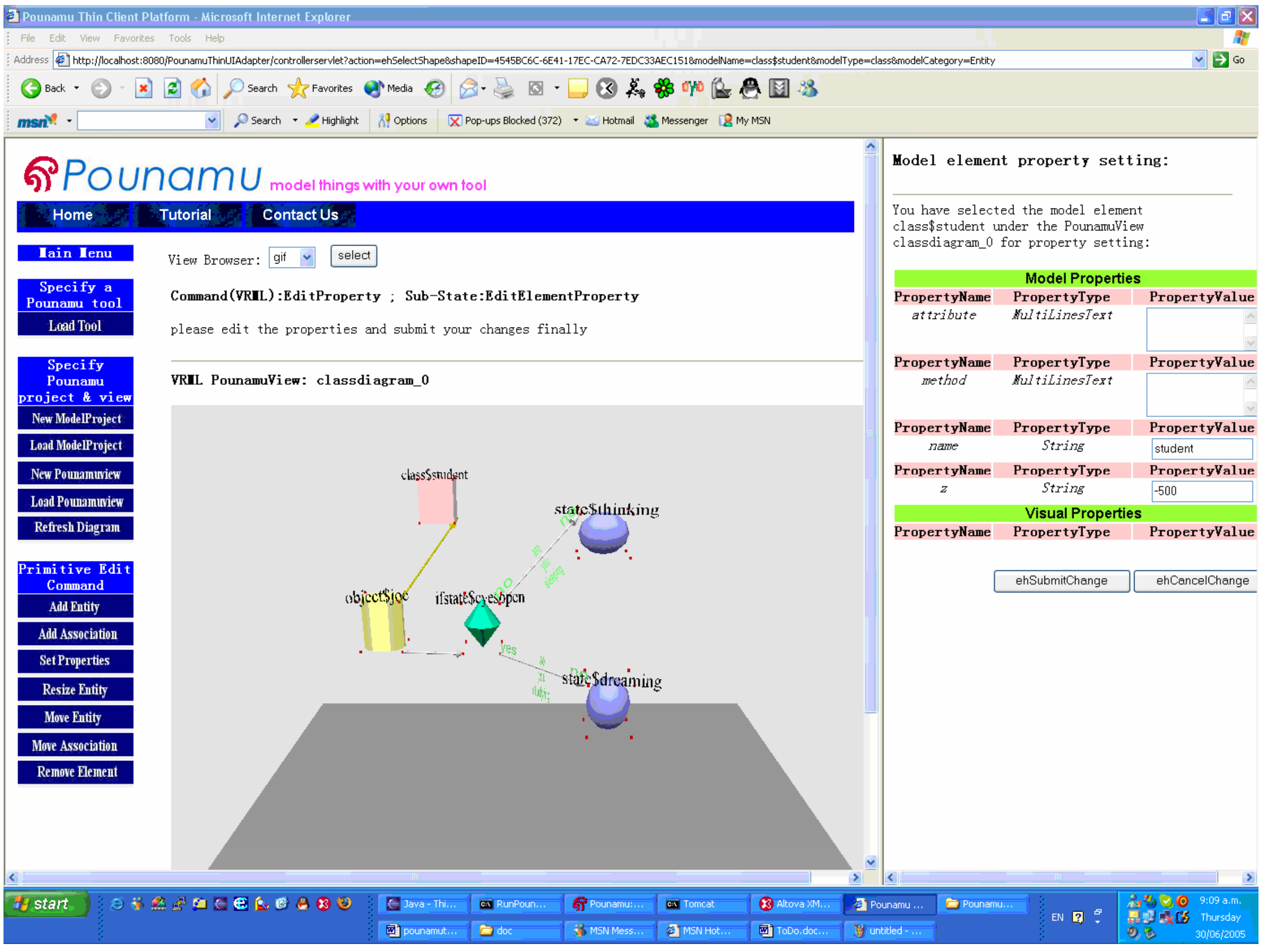


Figure 11. (a-c) user configuring class diagram representations in multi-level; (d) the class diagram with all items zoomed to level 1 as specified by users.



http://localhost:8080/PounamuThinUIAdapter/controllerservlet?action=ehSelectShape&shapeID=4545BC6C-6E41-17EC-CA72-7EDC33AEC151&modelName=class$student&modelType=class&modelCategory=Entity

View Browser: gif

Command(VRML):EditProperty ; Sub-State:EditElementProperty

please edit the properties and submit your changes finally

VRML PounamuView: classdiagram_0

Model element property setting:

You have selected the model element class$student under the PounamuView classdiagram_0 for property setting:

Model Properties		
PropertyName	PropertyType	PropertyValue
attribute	MultiLinesText	
PropertyName	PropertyType	PropertyValue
method	MultiLinesText	
PropertyName	PropertyType	PropertyValue
name	String	student
PropertyName	PropertyType	PropertyValue
z	String	-500
Visual Properties		
PropertyName	PropertyType	PropertyValue

Figure 12. 3D diagram viewing and editing.

3D Views for Pounamu/Thin

To demonstrate versatility of our Pounamu/Thin architecture, and motivated by emerging 3D environments such as Project Looking Glass [43], we have developed a further extension of Pounamu/Thin allowing diagrams to be converted into 3D representations and rendered and edited in a web browser. This is useful in situations where complex diagrams may benefit from a third dimension to help handle complexity and layout. Figure 12 shows an example of a 3D class diagram where the elements have been converted to 3D symbols (cylinders, spheres and cuboids), with selected labels rendered. The user has selected the student class in the 3D diagram to edit its properties, shown on the right hand side. The user may use the 3D plug-in capabilities to zoom in and out, pan and rotate the 3D representation. Client-side scripting is used to support adding, moving, resizing and deleting of diagram shapes as in the 2D web-based client-side editing example outlined previously. Using this extension, we are experimenting with a range of 3D metaphors for visualising model information. These involve configuring additional mappings from underlying Pounamu model elements to generate "the 3rd dimension" from model element attribute information.

POUNAMU/THIN ARCHITECTURE

Our Pounamu meta-tool was originally developed to provide flexible, interactive diagramming tool specification. It was assumed that all of the specified diagramming tools would provide a conventional thick-client user interface to tool users, by interpreting these design tool specifications. However, we increasingly found our end user base wanted to deploy their tools using thin client interfaces. One approach was to build a stand-alone web application that could import Pounamu tool specifications and realise these thin-client editors. However this would mean repeating much of the complex management support already developed for Pounamu e.g. data persistency to XML files and databases; model and view data structures and consistency management; and diagram rendering and editing support.

We pursued an alternative approach developing an optional thin-client diagramming extension to Pounamu. This is a set of web components developed independently of the Pounamu meta-tool itself but that make use of existing Pounamu APIs. The thick-client Pounamu application is thus repurposed to act as an "application server" reusing all of its standard data management, persistency and diagram editing support. The web components communicate with Pounamu to extract diagram information to render and to update diagram content. Our aim in doing this was to support thin-client diagram viewing and editing, but without the need to change or replicate any code in Pounamu itself, and still using the same tool specifications as thick-client tools. Figure 13 illustrates this thin-client extension that we call Pounamu/Thin.

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. To run the tool, Pounamu and Pounamu/Thin web components are deployed and the tool specifications loaded by Pounamu and interpreted. The Pounamu/Thin web components interact with Pounamu via a remote RMI-based API. As Pounamu is a "live" 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.

Having deployed a tool users can then access the tools diagrams generated by the Pounamu/Thin web components via conventional web browsers, using GIF based rendering by default. 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. Similarly 3D viewing of diagrams in the web browser requires a VRML plug-in, which also performs client side rendering and script-based client side interaction.

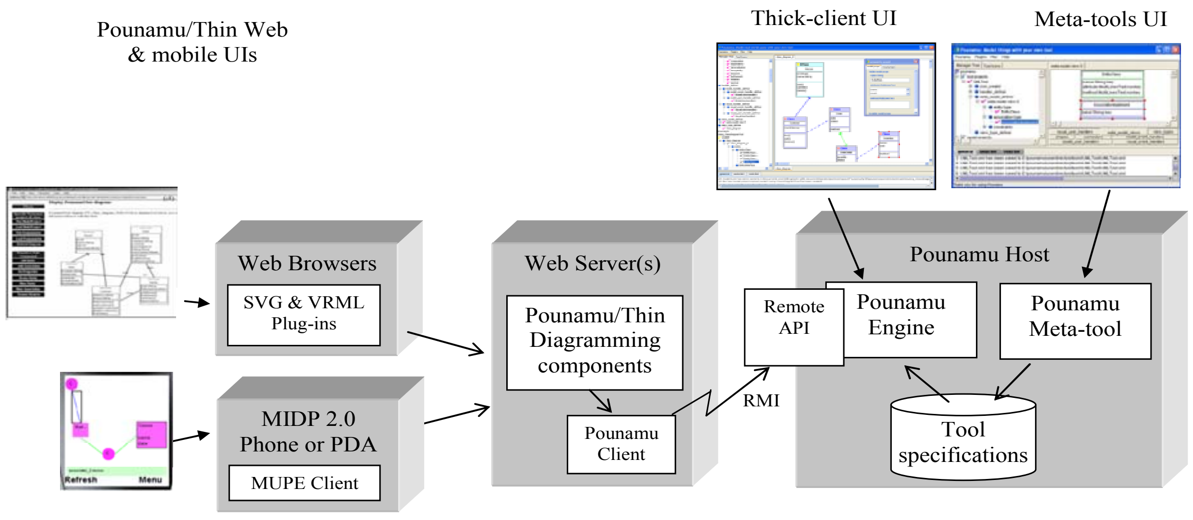


Figure 13. High-level Pounamu/Thin software architecture.

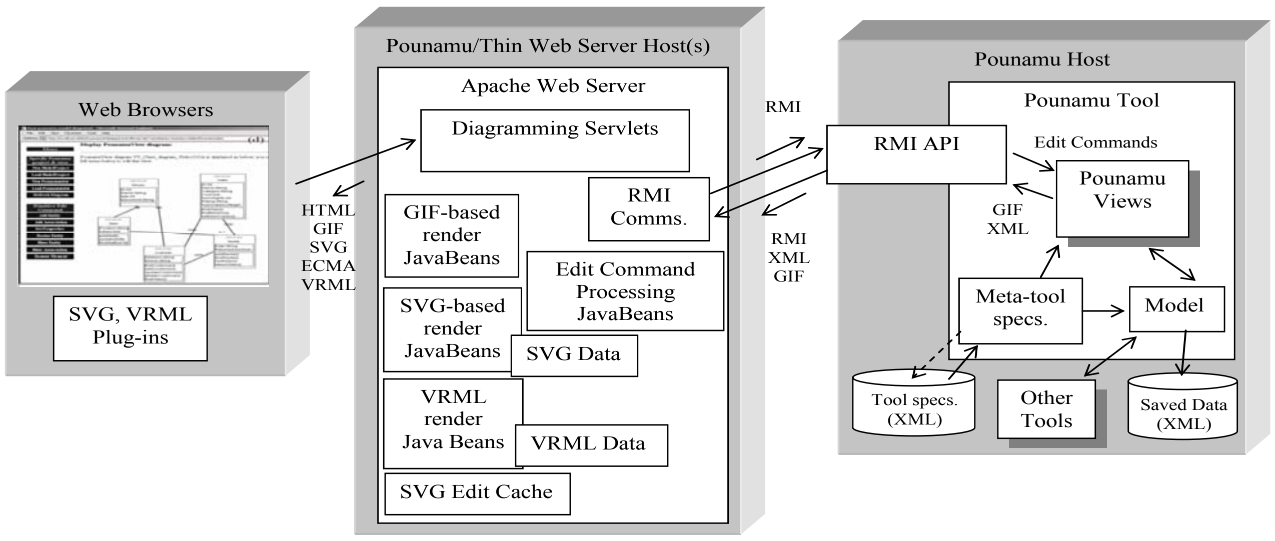


Figure 14. Architecture of Pounamu/Thin.

For deployment of Pounamu/Thin interfaces on mobile devices we chose to use Nokia's Multi-User Publishing Environment (MUPE) mobile client technology. MUPE is a mobile thin client technology developed by Nokia consisting of a MUPE server and MUPE client browser. The MUPE server responds to client requests by generating and sending back pages in XML, which can be browsed in a MUPE client. MUPE clients are relatively lightweight and are based on Java MIDP, which is enabled on most newer mobile devices. By using MUPE, diagrammatic user interfaces generated by our system can be easily deployed on a wide range of MIDP-compatible devices. Initially we implemented a "Pounamu/Mobile" server as a set of components hosted on a MUPE server. More recently we have refactored these components to use the same approach as for the web browser version with translation to and from the MUPE XML mark-up language.

DESIGN

Core Components

Figure 14 illustrates the key architectural components of Pounamu/Thin's web browser support. 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 Command pattern approach with a set of "Command objects" to represent edit operations being 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 include Add ShapeCommand, AddConnectorCommand, MoveShapeCommand, DeleteShapeCommand and SetPropertyCommand. These objects can be created and sent to a Pounamu view programmatically via an API in order to modify it. The API, which has RMI and SOAP variants, thus enables Pounamu views to be created and modified remotely.

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

Users interact with the Pounamu/Thin diagramming tools via standard web browsers. However, if SVG or VRML diagram rendering is desired, an SVG or VRML 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.

We chose to use a set of servlets to produce GIF, SVG, MUPE or VRML content for the browser with optional client-side scripting of SVG and VRML plug-ins to provide a degree of client-side drag-and-drop interaction. This is in contrast to using a Java Applet or Active X control in the browser with a similar editing interface to the Pounamu thick-client application. Our choice of using servlets that generate content to be rendered by the browser rather than an applet or Active X control was motivated by several factors: cross-browser compatability – our own and others experiences with both Active X controls and Java applets for such rich client interfaces in previous work (Graham et al 1999, Evans and Rogers, 1997) demonstrate major browser configuration problems are common occurrences. In addition, the use of servlets and translation to/from browser content and requests allowed us to use a common server-side architecture, rendering components and editing components across a range of diagram representations. However, our approach does not preclude using a Java applet of Active X control as Pounamu/Thin options in the future - both can communicate with the Pounamu/Thin servlets and request XML for rendering diagrams and send Pounamu commands as XML messages to modify diagrams.

Web Browser Interaction Models

As illustrated previously Pounamu/Thin provides several different 2D web browser diagram interaction models for users. These range from a full server-side processing model where images are rendered in a browser and all user interaction sent to the server, to a highly interactive client-side scripting approach. 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 15. Here, all interactions with the diagram are sent directly to the web server components (1). These generate an appropriate diagram update message (a Command) and immediately pass the Command onto the Pounamu server (2). These Commands are processed by the server and updates are made to the shared diagram. The server updates the shared diagram (3), and associated model elements, resulting in updated diagram and model. Periodically the model and diagram data structures are saved to XML files on the server (4). The Pounamu/Thin servlet then requests the updated diagram contents (5) and uses this to re-generate HTML, GIF and/or SVG content as appropriate for redisplay in the web browser (6).

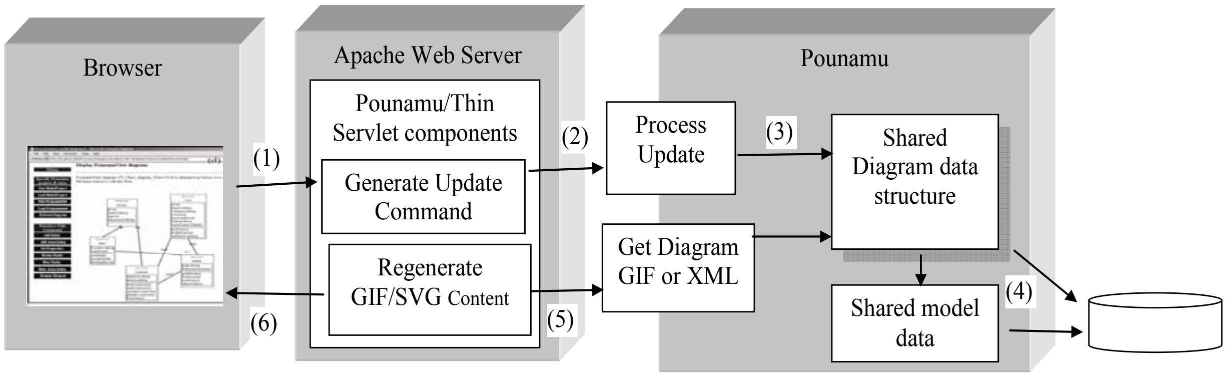


Figure 15. Fully-server side processing model.

The web server-side edit buffering approach is illustrated in Figure 16. In this approach the Pounamu 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), also resulting in shared model data structure updates and save of diagram/model state (7). The servlet will them fetch the contents of the diagram back from Pounamu, as it may contain the effects of edits made by other users as well as their own (8). The updated diagram will then be converted back into SVG or VRML and redisplayed. This approach is only available for SVG and VRML-format diagrams as GIF images are generated on the Pounamu server which does not support edit buffering.

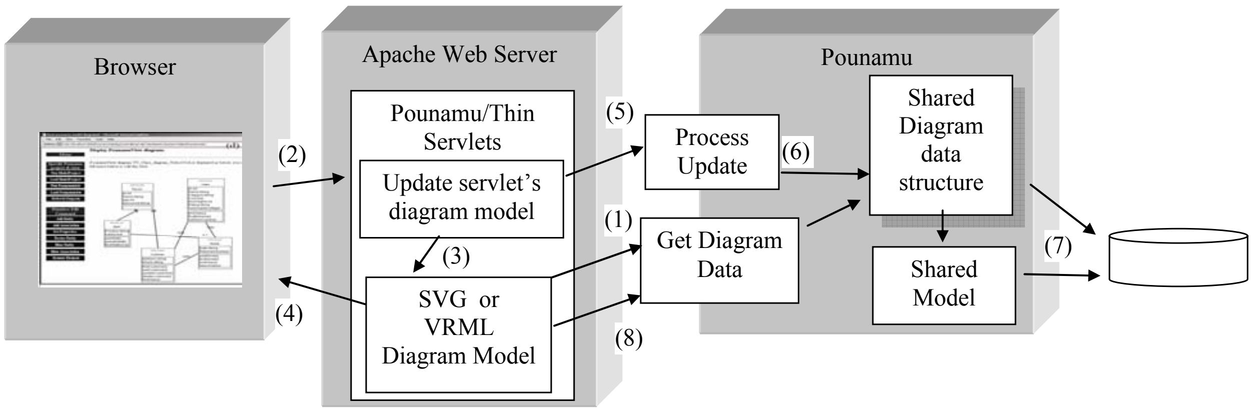


Figure 16. Edit buffering in servlet model.

The client-side scripting approach is illustrated in Figure 17, again available for SVG and VRML format diagrams. In this approach the Pounamu diagram contents are fetched from the shared Pounamu server (1) and cached for each user session in the servlets. In addition to HTML, SVG and VRML content, the servlet generates ECMA scripts for the SVG plug-in and JavaScript for the VRML plug-in (2). These scripts are passed with the HTML, SVG and/or VRML content to the web browser. These scripts program the browser plug-in to provide the Pounamu/Thin client-side editing behaviour (3). When a user moves, resizes or connects shapes, the edits are implemented in the browser and not sent to the web servlet (4). After several edits users can indicate 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 sent to the shared Pounamu server (6) where the shared diagram is updated (7), and the shared model updated and diagram/model data structure state periodically saved (8). The updated Pounamu diagram content is refetched by the servlet as XML data (9) and the SVG or VRML regenerated by the servlet to reflect both the client-side updates and any other user updates.

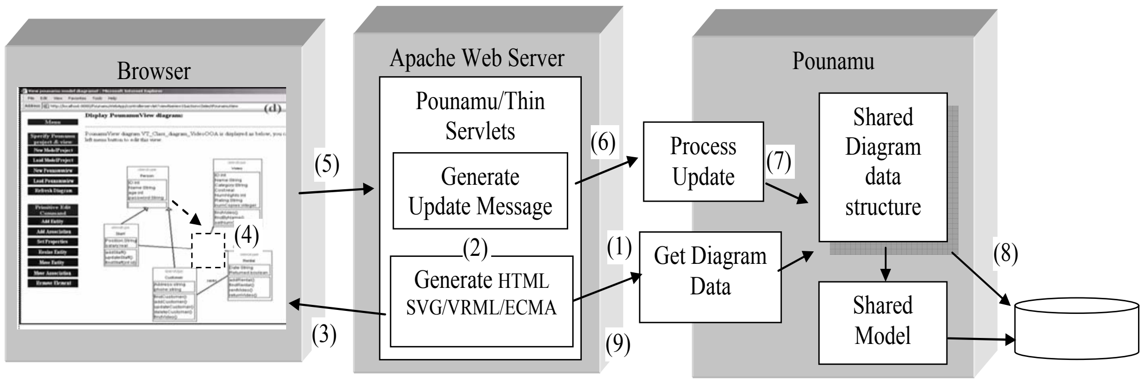


Figure 17. Client-side scripting interaction model.

We used a similar approach for the VRML-based 3D rendering of Pounamu/Thin diagrams. Pure server-side processing of user requests is too cumbersome when trying to navigate and modify a rendering in 3D in a VRML plug-in. Instead we used a client-side scripting approach very similar to the one used for SVG. If a 3D rendering of a Pounamu diagram is requested, the Pounamu/Thin components generating the VRML content include a set of ECMA scripts which program the VRML plug-in to support interaction and modification of the VRML content within the browser. This includes support for double-clicking on items and adding, moving, resizing and deleting them. ECMA script directly modifies of the VRML data structure within the browser plug-in ans uses asynchronous posting of update requests to the Pounamu/Thin server.

Likewise, displaying and interacting with diagrams on a mobile device can be done using fully server-side processing as with GIF diagrams in a PC web browser. However this proved slow and cumbersome due to the high latency and slow speed of mobile device networks. Instead we used a Java-based MIDP2.0 plug-in on the client handset to provide client-side scripting, allowing much of the diagram navigation, zooming and interaction to be carried out on the handset. This greatly reduces network traffic between the handset and Pounamu/Thin server. We used the Nokia MUPE framework to provide a set of Pounamu/Thin server components specifically targeted at providing the mobile device interfaces. While this approach is applet-based, in contrast to the web browser based approaches of the other Pounamu/Thin applications, this is justified by the lightweight and "standardised" nature of the MUPE client in comparison with typical web browsers available on Mobile phones and PDAs.

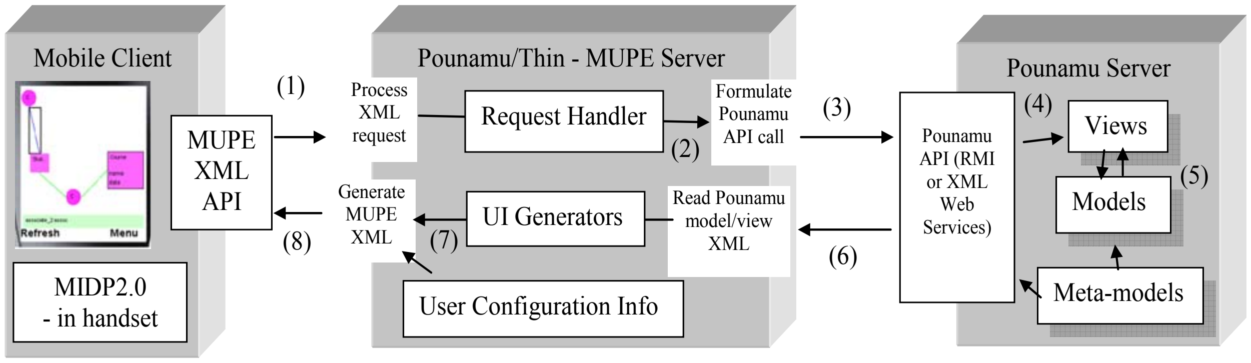


Figure 18. Basic components in Pounamu/Thin for mobile device clients.

Figure 18 describes diagram editing in a Pounamu/Thin mobile client. If a user asks for an element to be deleted, a MUPE XML event message is sent by the MUPE client on the device to the MUPE server (1). A Pounamu/Thin request handler determines the diagram update required (2) and formulates a Pounamu API call to action the diagram update (3). This results in the element being deleted from the Pounamu view (4), also deleting the model element and impacting other views of the deleted element (5). Once the Pounamu server acknowledges the update has successfully occurred, Pounamu/Thin re-reads the updated view's Pounamu XML from the Pounamu server (6). It then synthesises a new view in the MUPE XML format, using the user preferences and Pounamu view XML to do this (7). The MUPE client is returned a new page to display, the updated diagram content (8). Various optimisations are possible, including caching the MUPE XML in the MUPE server and only updating affected portions. However we have found the response time when re-generating the whole diagram content adequate and this simplifies the process considerably.

Rendering Transformations

In order to produce content for a thin-client device, whether web browser, mobile device, or 3D image, Pounamu diagram content must be transformed from its XML format into suitable thin-client device content (GIF, SVG, VRML, MUPE XML). In addition, if client-side scripting is used, and it must be for mobile and 3D content, the payload for the target device must include suitable scripting code as well as content to render. Figure 19 shows a sequence diagram outlining the generic transformation process when a diagram is requested by a client device. The process is slightly different for each kind of content and is currently implemented by different Pounamu/Thin server-side web components, but, as can be seen, the general pattern of execution is the same.

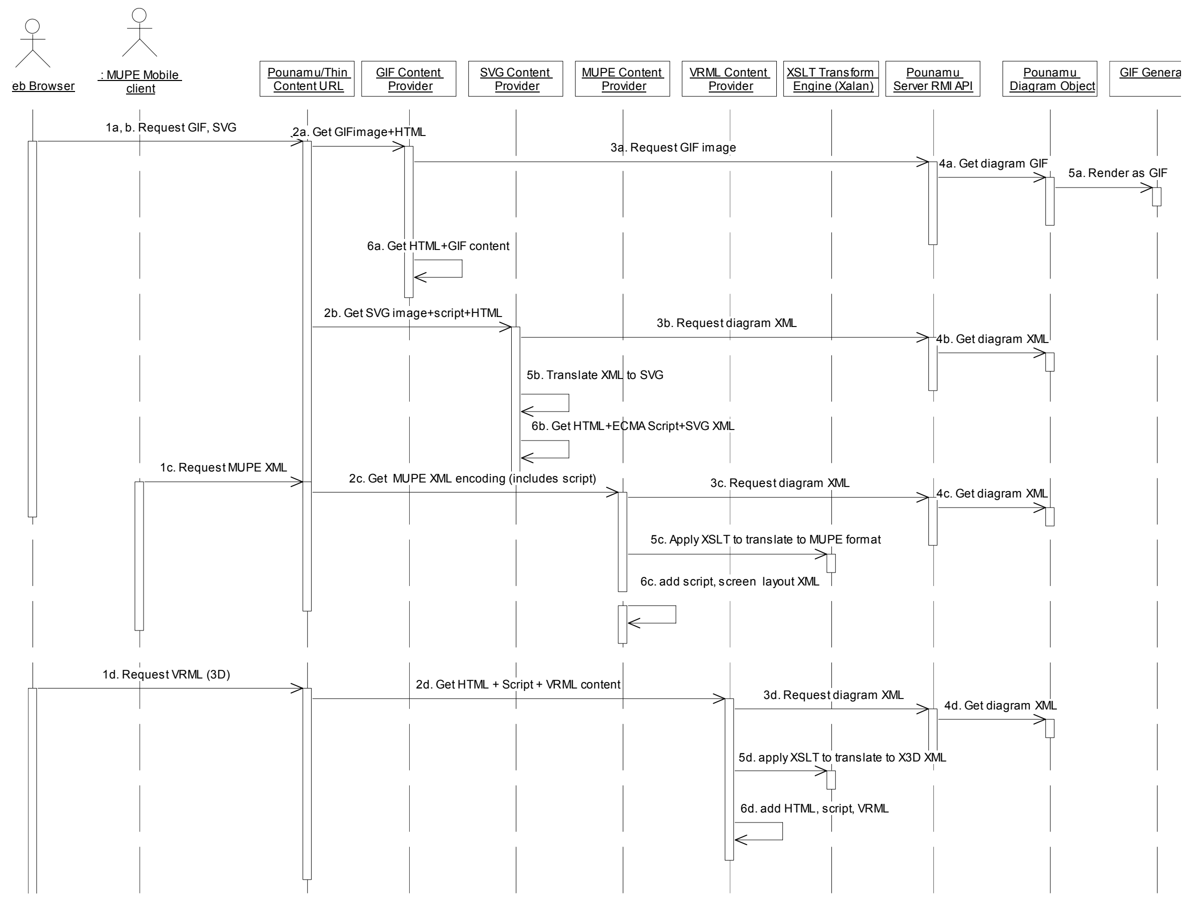
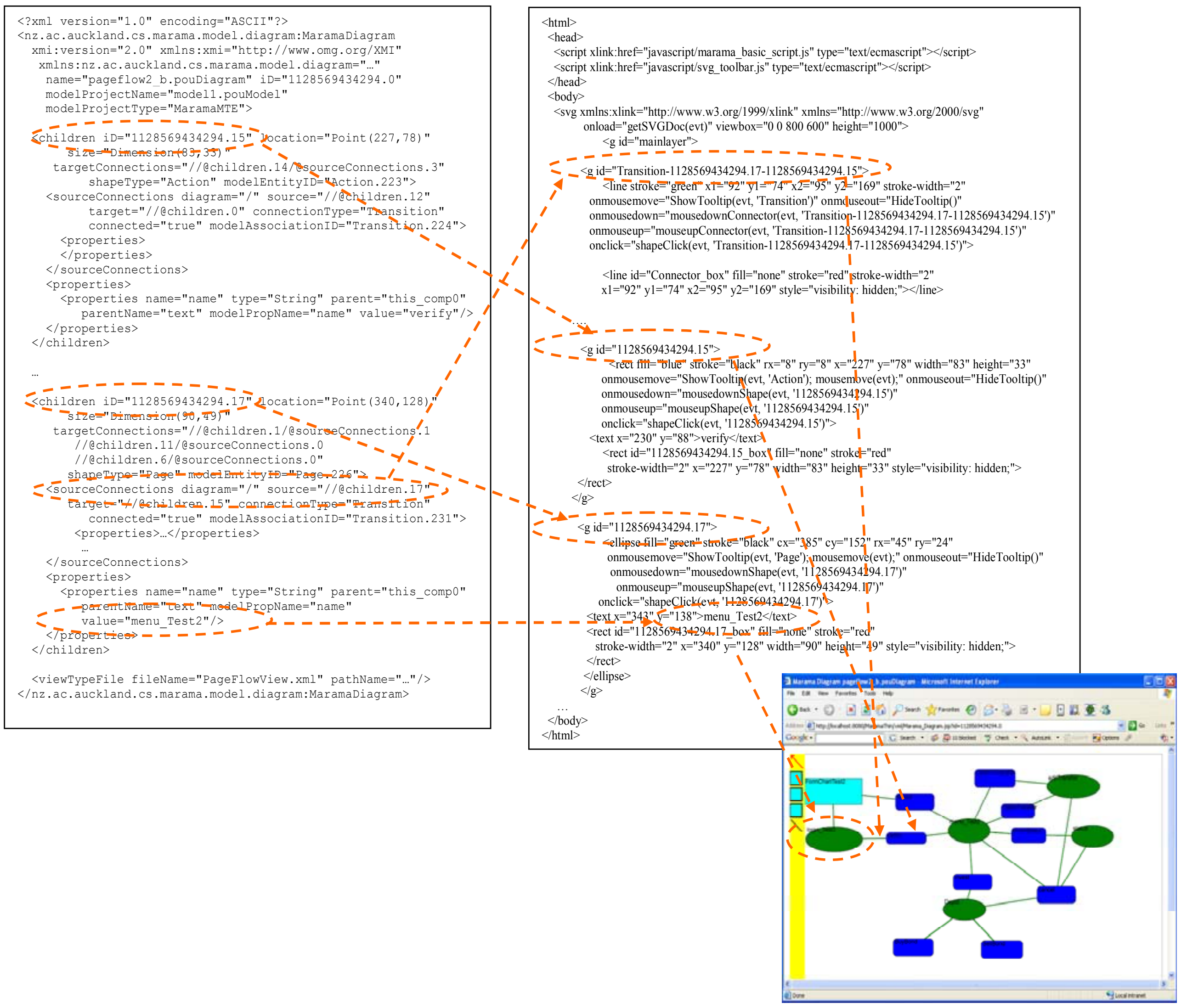


Figure 19. Outline of the Pounamu/Thin XML transformation process.

All requests are sent to a Pounamu/Thin URL with arguments to indicate the kind of content to provide (Figure 19 1 a-d). When requesting a GIF image the GIF Content Provider servlet is used to generate HTML and the GIF image to return to the client web browser (2a). This requests a GIF image for the requested diagram via the Pounamu server API (3a). Unlike other content the GIF image currently must be generated by a plug-in to the shared Pounamu application server; we use a separate Thread to generate these images and re-render a copy of the diagram to a binary array in a GIF format (4a, 5a). The GIF Content Provider servlet then returns HTML markup providing the title, menus, frames and the GIF image to the client browser. SVG content requests may include scripting (if client-side scripting mode is requested by the user) as well as HTML and the SVG image (2b). The SVG Content Provider requests the Pounamu diagram as an XML document (3b, 4b) and transforms this into an SVG image by traversing the diagram XML in a Document Object Model (DOM) (5b). We currently implement this transformation as optimised Java code over the DOM structure. In future it could use an XSLT or similar transformation engine as do the MUPE and VRML content production.

Mobile devices are currently provided a MUPE XML document as their content to render and are currently required to have Java MIDP2.0 and the MUPE client handset plug-in installed (1c). The MUPE Content provider requests the Pounamu diagram XML document (3c, 4c) and then applies an XSLT transform to the document to convert it from the Pounamu's diagram XML format into MUPE diagram rendering XML format (5c). The MUPE Content Provider then adds this diagram content XML inside a new XML document incorporating screen layout and handset scripting XML (6c), producing the final MUPE XML document to be returned to the mobile device.

Requests for 3D rendering of Pounamu diagrams (1d) are processed by a VRML Content Provider (2d) which requests the Pounamu diagram XML (3d, 4d) and transforms this into X3D, an XML document structure (5d). The VRML Content Provider servlet could equally produce a VRML document (non-XML) but we chose to use X3D, the newer XML-based representation for VRML content. HTML and ECMA script content is generated by the VRML Content Provider servlet and the X3D document included by the HTML to provide the 3D representation and associated scripting and HTML menu and frames content for the client web browser (6d).



```
<?xml version="1.0" encoding="ASCII"?>
<nz.ac.auckland.cs.marama.model.diagram:MaramaDiagram
  xmi:version="2.0" xmlns:xmi="http://www.omg.org/XMI"
   xmlns:nz.ac.auckland.cs.marama.model.diagram="…"
    name="pageflow2_b.pouDiagram" iD="1128569434294.0"
    modelProjectName="model1.pouModel"
    modelProjectType="MaramaMTE">

  <children iD="1128569434294.15" location="Point(227,78)"
       size="Dimension(83,33)"
     targetConnections="//@children.14/@sourceConnections.3"
          shapeType="Action" modelEntityID="Action.223">
    <sourceConnections diagram="/" source="//@children.12"
          target="//@children.0" connectionType="Transition"
          connected="true" modelAssociationID="Transition.224">
      <properties>
      </properties>
    </sourceConnections>
    <properties>
      <properties name="name" type="String" parent="this_comp0"
         parentName="text" modelPropName="name" value="verify"/>
    </properties>
  </children>

  …

  <children iD="1128569434294.17" location="Point(340,128)"
       size="Dimension(90,49)"
     targetConnections="//@children.1/@sourceConnections.1
        //@children.11/@sourceConnections.0
        //@children.6/@sourceConnections.0"
       shapeType="Page" modelEntityID="Page.226">
    <sourceConnections diagram="/" source="//@children.17"
       target="//@children.15" connectionType="Transition"
          connected="true" modelAssociationID="Transition.231">
        <properties>…</properties>
         …
    </sourceConnections>
    <properties>
      <properties name="name" type="String" parent="this_comp0"
         parentName="text" modelPropName="name"
         value="menu_Test2"/>
    </properties>
  </children>

  <viewTypeFile fileName="PageFlowView.xml" pathName="…"/>
</nz.ac.auckland.cs.marama.model.diagram:MaramaDiagram>
```

```
<html>
 <head>
  <script xlink:href="javascript/marama_basic_script.js" type="text/ecmascript"></script>
  <script xlink:href="javascript/svg_toolbar.js" type="text/ecmascript"></script>
 </head>
 <body>
  <svg xmlns:xlink="http://www.w3.org/1999/xlink" xmlns="http://www.w3.org/2000/svg"
       onload="getSVGDoc(evt)" viewbox="0 0 800 600" height="1000">
          <g id="mainlayer">

     <g id="Transition-1128569434294.17-1128569434294.15">
        <line stroke="green" x1="92" y1="74" x2="95" y2="169" stroke-width="2"
      onmousemove="ShowTooltip(evt, 'Transition')" onmouseout="HideTooltip()"
      onmousedown="mousedownConnector(evt, 'Transition-1128569434294.17-1128569434294.15')"
      onmouseup="mouseupConnector(evt, 'Transition-1128569434294.17-1128569434294.15')"
      onclick="shapeClick(evt, 'Transition-1128569434294.17-1128569434294.15')">

        <line id="Connector_box" fill="none" stroke="red" stroke-width="2"
        x1="92" y1="74" x2="95" y2="169" style="visibility: hidden;"></line>

    ….

     <g id="1128569434294.15">
         <rect fill="blue" stroke="black" rx="8" ry="8" x="227" y="78" width="83" height="33"
        onmousemove="ShowTooltip(evt, 'Action'); mousemove(evt);" onmouseout="HideTooltip()"
        onmousedown="mousedownShape(evt, '1128569434294.15')"
        onmouseup="mouseupShape(evt, '1128569434294.15')"
        onclick="shapeClick(evt, '1128569434294.15')">
      <text x="230" y="88">verify</text>
        <rect id="1128569434294.15_box" fill="none" stroke="red"
         stroke-width="2" x="227" y="78" width="83" height="33" style="visibility: hidden;">
    </rect>
   </g>

     <g id="1128569434294.17">
        <ellipse fill="green" stroke="black" cx="385" cy="152" rx="45" ry="24"
         onmousemove="ShowTooltip(evt, 'Page'); mousemove(evt);" onmouseout="HideTooltip()"
          onmousedown="mousedownShape(evt, '1128569434294.17')"
           onmouseup="mouseupShape(evt, '1128569434294.17')"
        onclick="shapeClick(evt, '1128569434294.17')">
      <text x="343" y="138">menu_Test2</text>
      <rect id="1128569434294.17_box" fill="none" stroke="red"
        stroke-width="2" x="340" y="128" width="90" height="49" style="visibility: hidden;">
      </rect>
     </ellipse>
    </g>
   …
  </body>
</html>
```

Marama Diagram pageflow2_b.pouDiagram - Microsoft Internet Explorer
File Edit View Favorites Tools Help

Figure 20. Example of Pounamu diagram XML transformation to SVG.

Not shown in Figure 19 are caching and threading structures used. Rather than use the Pounamu modelling tool API calls to get diagram XML the Pouanmu/Thin web server components can share a cached copy. This allows multiple requests for the same diagram content from different users' to be generated once when the content is changed in the Pounamu tool server and served to each user without re-generating the diagram XML. Similarly some client plug-ins cache a copy of the diagram data structure on the client device e.g. the edit buffering and client-side scripting versions of the SVG content, the MUPE mobile client content and the VRML plug-in content. Client-side scripts for each of these target devices can modify the diagram data structure (held as SVG, MUPE or X3D format XML documents in the client-side browser plug-in) without communicating immediately with the Pounamu/Thin web server. This greatly improves response time for e.g. drag-and-drop editing for the user. Threading is used on the Pounamu/Thin server to provide each client their own copy of all Pounamu/Thin servlet objects. Synchronisation points are on the cache update method for cached Pounamu diagram XML documents and on all of the Pounamu server RMI API calls, eliminating concurrent operations on these.

Figure 20 shows an example of the transformation between a Pounamu diagram representation as an XML document (left) and a target SVG XML document (right) for rendering by the client web browser SVG plug-in. Several correspondences between Pounamu diagram XML structures and SVG document items are highlighted. In addition, the SVG Content Provider generates HTML to contain the SVG document and includes several JavaScript files providing client-side scripting support for drag-and-drop move, resize and delete implemented by the client web browser SVG plug-in. The resulting rendering of the diagram in the browser is also shown (bottom).

In order to update shared Pounamu server information, all modifications to diagrams by user interaction with their clients must be converted into API calls to modify Pounamu data structures. Figure 21 shows a sequence diagram outlining the process of converting user interactions into such Pounamu data structure updates. Again this is slightly different for each kind of thin client. The request to generate the Pounamu API messages is done differently depending on the kind of client device and whether or not client-side scripting is used. A POST from the browser (either script-generated or user-generated) is processed by appropriate Pounamu/Thin components: User POST operations (button click in browser) 2a); client-side script POST operations; and MUPE XML messages are all translated into appropriate Pounamu diagram update Command objects (3a-c) e.g. AddShape, MoveShape, DeleteConnector or composite lists of Commands. These Command objects are sent to the Pounamu Server via RMI API call(s) (4) and are run by the Pounamu modelling tool server (5) to effect diagram updates. Diagram objects (shapes, connectors and the diagram itself) are modified by each operation (6), and underlying shared model instances (entities and associations) are modified appropriately (7). Diagrams sharing modified model information are then updated by further Command objects generated by Pounamu (8-9). A subscribe-notify mechanism is used by the Pounamu/Thin server diagram XML cache to refresh the cache contents for all modified diagrams in the Pounamu server (10). New client data is then generated and returned to the client device using the mechanism described in Figure 19.

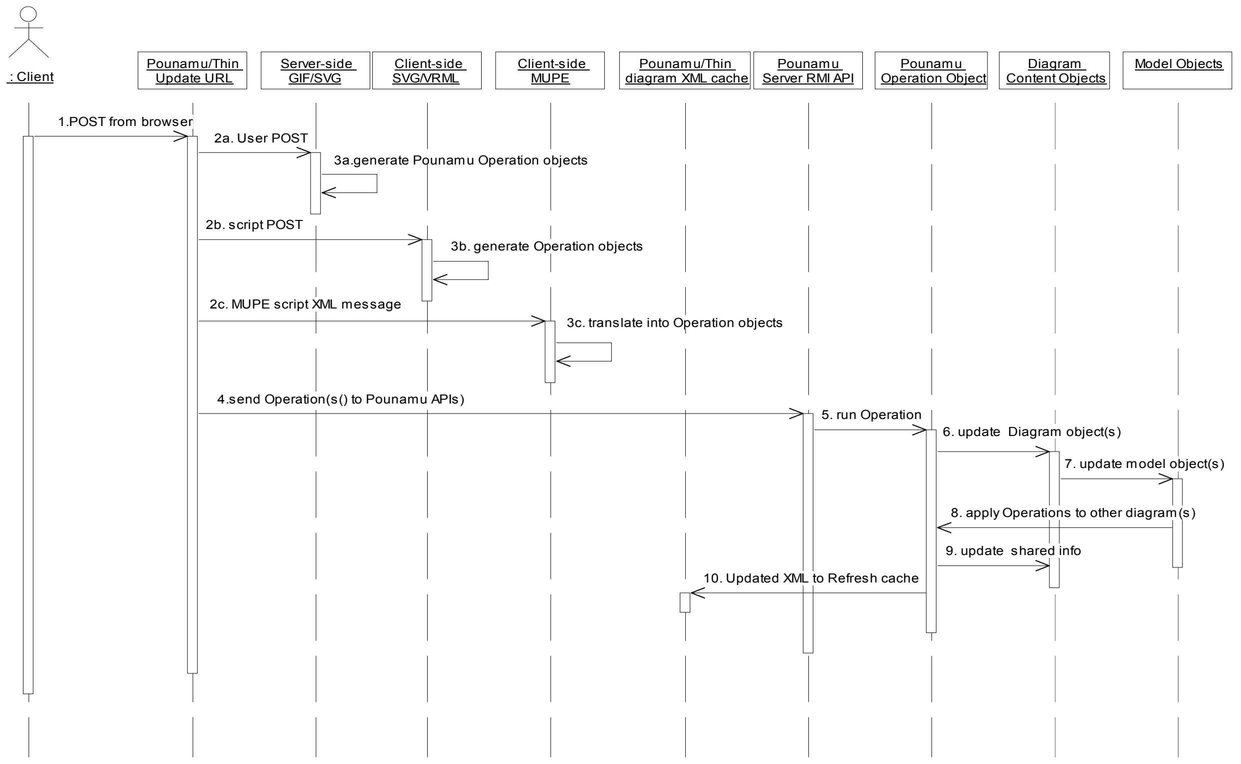

: Client
Pounamu/Thin Update URL
Server-side GIF/SVG
Client-side SVG/VRML
Client-side MUPE
Pounamu/Thin diagram XML cache
Pounamu Server RMI API
Pounamu Operation Object
Diagram Content Objects
Model Objects
1.POST from browser
2a. User POST
3a.generate Pounamu Operation objects
2b. script POST
3b. generate Operation objects
2c. MUPE script XML message
3c. translate into Operation objects
4.send Operation(s() to Pounamu APIs)
5. run Operation
6. update Diagram object(s)
7. update model object(s)
8. apply Operations to other diagram(s)
9. update shared info
10. Updated XML to Refresh cache

Figure 21. Update request processing from client device.

IMPLEMENTATION

Figure 22 provides an outline of the key technologies we have used to date to implement the Pounamu/Thin thin-client diagramming architecture. We chose to use a set of Java Servlets hosted by a Tomcat servlet engine and RMI communication between the servlets to a single instance of the Pounamu meta-tool acting as a shared application server. This Pounamu instance acts as both the meta-tool capability, allowing definition and modification of diagramming tools, and the data and processing application server for Pounamu/Thin.

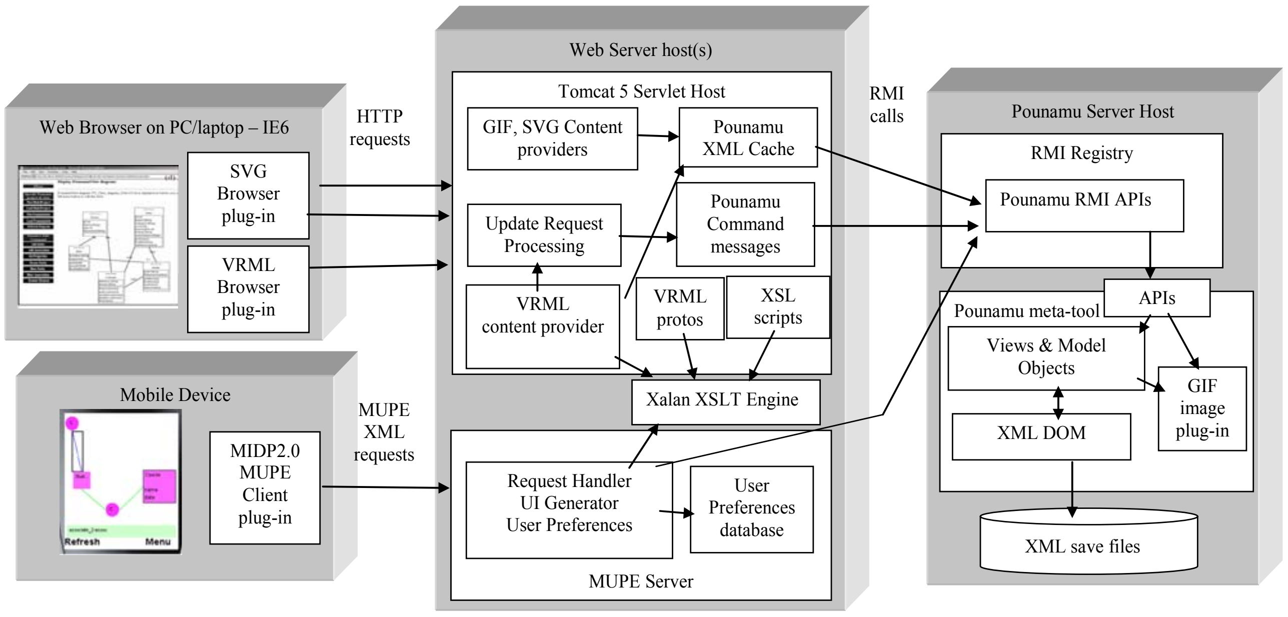


Figure 22. Pounamu/Thin APIs and technologies used.

Pounamu itself is implemented in Java and provides comprehensive RMI and web services interfaces (Zhu et al, 2004; Mehra et al, 2004). The RMI interface to Pounamu performs much faster than the web services one and we chose to use this for Pounamu/Thin. Java Swing was used to implement Pounamu's thick-client view editors and Pounamu makes extensive use of XML DOM structures to manage diagram and model data structure representations. 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 useful side-benefit of 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 provided much faster performance and because this process required quite complex calculation algorithms not well-suited to implementation in XSLT. 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.

As the structural transformations were relatively straightforward, we used XSLT to support the generation of VRML renderings of Pounamu diagrams. The VRML content provider servlet requests a Pounamu diagram's XML content and applies a set of XSLT transforms to the XML to generate a X3D (VRML XML) diagram. One VRML prototype shape must be defined for each Pounamu diagram shape and connector type which is used by the transforms to generate a VRML element for each Pounamu diagram element. Currently we do not provide a tool to generate these automatically from Pounamu meta-tool shape and connector specifications as we do for SVG and MUPE content. We found the range of possible renderings of 2D Pounamu components using 3D VRML elements to be too complex for fully automatic transformation as we did with SVG and MUPE diagram content. One straightforward future extension would be to add automatic generation of a first-attempt prototype VRML shape and allow users to modify these prototypes with a VRML authoring tool.

Pounamu views are saved to an XML format by the tool for both saving and loading views and also to support collaborative thick-client editing of views via heavyweight distributed message passing between multiple instances of Pounamu (Mehra et al, 2004). In addition, Pounamu view editing Command objects can 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 thin client editing of Pounamu views and 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.

We chose the Nokia MUPE mobile device technology to implement our Pounamu/Thin mobile thin-client diagramming architecture with Java RMI communication to Pounamu. The MUPE architecture provides a rich mobile client development platform that is device-independent. It also provides a server with a Java-based plug-in architecture, enabling a wide range of server application components to be developed. We developed several of these to implement Pounamu/Thin support for mobile devices. The Pounamu/Thin mobile device server is made up of a set of MUPE server components that interact with the Pounamu application meta-tool to synthesise mobile device user interfaces. MUPE client content is an XML document containing layout, content and scripting information.

The Pounamu/Thin mobile device server comprises 3 key components: Request Handling, UI Generation, and User Tool Configuration. Request handlers respond to all requests from MUPE clients. They hold the functional services supplied by the server. The Request Handling component is responsible for handling user requests, communicating with the Pounamu/Thin server components, and notifying corresponding UI generators to generate pages. The UI generators generate MUPE format user interfaces of Pounamu views, property forms, and tool configurations. Each UI generator contains a set of objects, which have their own visual representation templates in an XML format and are similar to VRML prototype objects. Each registered user has his/her own tool configuration as an XML file for each tool he/she has loaded. These configuration files are stored in the Pounamu/Thin mobile server. To generate a diagram view, the Model UI generator collects and combines information from multiple sources, including model view information from the Pounamu application server, user personalized specification of diagrams from user's tool configuration file, screen size of client device, and diagram rendering templates of rendering objects.

We initially implemented custom Java code to carry out this UI generation. However, like the VRML content generator we modified this to use XSLT transformations to make them easier to modify and extend, at the cost of some performance loss. Our aim is to replace the MUPE server with Tomcat servlets that provide MUPE-compatible XML to mobile devices and use XSLT-based transformations to process requests. This refactoring will complete the unification of the SVG, VRML and MUPE request/render component architecture.

Limited support is provided for collaborative diagram editing. A cache is used by the SVG and VRML content renderers to reduce requests on the shared Pounamu application server. The RMI plug-in for the Pounamu application server sequences all requests from the Pounamu/Thin web server components ensuring no concurrent modification of Pounamu data structures. A sequence of Commands can be run as a unit to ensure transactional integrity of a set of diagram modifications. Plug-ins in the application server control concurrent access to diagram components via locks and highlighting of diagram elements that are in use by another user. These were developed to support thick-client collaborative editing (Mehra et al, 2004) but also work well for thin-client collaborative editing.

Integration with other software tools is provided by Pounamu, typically using XML-based data exchanges. In addition, some users can chose 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 web services 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 have a mix of users using either Pounamu/Thin thin-client or thick-client Pounamu diagramming of the same shared view.

EXPERIENCES

To date we have built several domain-specific language applications to leverage our Pounamu/Thin thin-client user interface synthesis technology. In conjunction with an industry client producing project management tools we developed several diagram specifications in Pounamu for use in web browsers and mobile devices (Zhou et al, 2006). These included Gantt charts, PERT charts and work break-down schedules. We have developed a thin-client software architecture design tool for use with the MASE project (Chau and Maurer, 2004) and to allow user configuration of Fitness tests. We developed a thin-client XForm design tool with an industry client to support design of forms in a web browser with generation of XForm descriptions from Pounamu design diagrams. We have used Pounamu/Thin with several in-house software design tools including a Pounamu UML tool with class, collaboration, use case, sequence diagram and deployment diagram views, and a business strategy modelling tool with process modelling views (Cao et al, 2005). In no case has any code change needed to be made to Pounamu or Pounamu/Thin components to provide fully-functional web browser and mobile phone diagram editors for these tools.

We have carried out two evaluations of Pounamu/Thin thin-client web-based diagramming, 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 or 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 task 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 indicated that the users perceived the thin client tool provided the same facilities as the thick-client version. Most users in the evaluation 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. Feedback on user interaction in the thin-client tools was positive, with users able to create and modify diagram components easily. 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 learning curve for inexperienced users of Pounamu tools was found to be least for users of the GIF-based thin-client tool. However, 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. It may be the case that further experience with it is needed and it can be turned off if not wanted.

The 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. All users preferred 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). Currently changing a tool specification can only be done with the thick-client tool designer, which was seen as a disadvantage by some users.

Our second evaluation was a cognitive dimensions (CD) evaluation of the three versions of our UML diagramming tool. The 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 generally more "viscous" i.e. harder to effect change, than their thick-client counterparts. This is particularly so for moving and resizing operations that require multiple web browser interactions by users without client-side scripting. This characteristic is cited most commonly against use of thin-client diagramming approaches. However, our evaluation of Pounamu/Thin with end users indicates supporting client-side drag-and-drop for resize, move and delete generally mitigates this problem for most users. Diagram editing on MUPE mobile devices similarly must leverage a degree of client-side scripting to support zoom and pan in addition to resize/move operations.

- *Visibility and Juxtaposability*. The Pounamu thick-client UML modeller allows multiple views to be displayed side-by-side or accessed via both a tree and tabbed panes. Thin-client tools require multiple web browser windows displaying different views to be opened and positioned on the user's display and a simplified list of views is provided for the user to select from. The mobile thin client permits only one view to be visible at any time and thus provides poorer visibility and juxtaposability of diagrams than the other approaches due to its small screen size limitations.

- *Hard mental operations*. Users found the learning curve of our Pounamu/Thin tools to be somewhat less in many respects than the Pounamu thick-client tool interface. Pounamu/Thin interfaces present many interaction options in more explicit ways, via always-visible buttons, a large message panel 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 in some Pounamu/Thin configurations. Navigating a generated VRML

3D model for a Pounamu diagram uses non-intuitive mouse and keyboard manipulations for the VRML browser plug-in. Interaction with 3D diagram elements is similarly non-intuitive at times due to the synthesized Z dimension for some shapes and the un-obvious interaction points on shapes.

- *Hidden dependencies and View support*. Multiple views of model elements are supported in all diagramming tools and dependencies between view elements and model elements are via shape and connector names. 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. Mobile hosted diagrams must incorporate multi-level zoom support in addition to the ability to move to other diagrams, resulting in loss of context for complex diagrams or frequent inter-diagram navigation.

- *Consistency*. The thin-client diagramming tools adopt as their main interaction metaphor page-based diagram viewing and interaction, where each user interaction causes a POST to the web server and then refresh of the screen. The Pounamu thick-client diagramming interface instead uses direct manipulation, drag-and-drop and pop-up menu non-modal approach. 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 behaviour that is common to all web browser-hosted applications. The button-based, page-organised approach of Pounamu/Thin to providing editing and view management operations provides users with a less complex interface than the many thick-client Pounamu tool pop-up and pull-down menus. Once users are familiar with the request-response approach adopted by Pounamu/Thin interfaces for e.g. creating shapes, the exact same metaphor is used with all other editing and view selection operations.

- *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. The MUPE mobile and VRML thin-client diagramming applications in contrast post user updates to the server asynchronously via their client-side scripting to effect server diagram updates. These mechanisms become apparent when performing collaborative work where the thin-client tools require user-directed refresh of pages to show highlighting of others' updates and locked diagram elements. In contrast, the thick-client Pounamu UML tool uses push-based view update and redisplay, providing a more natural, proactive collaborative work supporting user interface (Mehra et al, 2004).

We have evaluated our Pounamu/Thin MUPE mobile device components using several visual modelling tools specified by Pounamu. Again no code change to Pounamu/Thin was required. All that was needed was a configuration specification by the end user for each view type if s/he wanted to modify the default configuration. We have carried out a preliminary evaluation of the usability and performance of these Pounamu/Thin mobile device-based tools with several experienced mobile phone and PDA users. We carried out the experiment with the Pounamu/Thin and Pounamu servers installed on a workstation and users accessing the mobile interface via a MUPE client simulator, also running on separate workstations in different locations. We used two basic experiments, one using the project management tool prototype with users carrying out basic task planning and co-ordination activities. The second involved users reviewing and making minor modifications to (fairly simple) UML class diagrams representing a data model for a business application. Users carried out both concurrent (synchronous) activities as well as asynchronous activities in both experiments. Users also used the Pounamu desktop application to carry out these same tasks as a comparison. We used the exact same tool specifications from Pounamu for both Pounamu/Thin experiments as for the desktop viewing and editing experiments.

These evaluations indicate that our Pounamu/Thin mobile device diagram approach provides similar viewing and editing capabilities to our previous desk-top thick client and web-based thin client versions of Pounamu. Users were able to access both navigation and editing facilities in the mobile device but as expected the editing of diagrams was much more difficult than desktop or even web browser-based interfaces. Navigation issues on a small screen were to some degree mitigated by Pounamu/Thin's multi-level zoom capability and by users being able to specify multiple visual representations for a single diagram shape. Users found these features to be essential in order to support complex diagram browsing and drill-down to detailed item viewing and editing. Users indicated that more automated support by the Pounamu tool would help diagram editing e.g. automated placement, layout and auto-zooming both during browsing and editing. This was particularly so for the UML tool which provides little such support, as it is not required by users in the desktop and web browser interfaces. The Gantt chart tool provides some automatic layout support which users found helpful on the mobile device. In both examples as soon as diagrams become moderately large (greater than about 15 items) they became very difficult to use on the mobile interface. Overall users thought the approach to be effective for making diagrammatic content accessible via a mobile device while allowing other users to simultaneously access desktop and web browser versions of the content. We are using the feedback obtained from this preliminary survey to refine our approach. It is clear from this feedback that a number of usability enhancements to make browsing and editing faster and easier are required to make the approach feasible for real work. These include user configurable hot-keys to speed up some functions, especially browsing, and user configurable layout in addition to zoom shape specification. In addition, support for collaborative interactions needs to be improved, as has been done in the work of others (Luz and Masoodian, 2003). Nevertheless, the initial feedback is promising and justifies the development of the experimental testbed. Our aim is to conduct further experiments using this framework to assist us in developing guidelines for the types of diagrams able to be usefully deployed in this way and additional mitigation approaches to overcome the limited screen and interaction capabilities.

Like other researchers we have encountered some difficult technical challenges in developing Pounamu/Thin mobile device user interfaces. The graphic rendering limitations of MIDP and difficulty of user interaction on mobile phones means that diagrams specified and rendered on mobile devices cannot be as complex as on a PC. However, as mobile devices continue to become more powerful we expect to be able to achieve more complex diagramming rendering on mobile devices. Icon specification is currently quite cumbersome using the MUPE mobile device interface. This would be much easier and flexible on pen-based PDAs than on the key-based phones that we have used in our experiments to date. Because of the memory limitations on current mobile devices, very large diagrams cannot be rendered, even when using MUPE thin client technology. Instead, at present, smaller overlapping Pounamu views must be constructed. Again the rapid evolution of mobile hardware will remove this limitation. We currently use a basic layout algorithm to transform Pounamu views into an overview for mobile devices. This works fine for e.g. our UML tool which is not so sensitive to spatial layout. However, some tools use layout as an important component of the diagram such as the Gantt charts in our project management tool.

Preliminary experiments have been carried out with loading the Pounamu/Thin web components. We simulated multiple user interaction with the diagramming tools by setting up the Pounamu application on one host and Tomcat server hosting Pounamu/Thin servlets on another machine with a 100MBit network connection. Several concurrent client diagram display and update operations were simulated by a multi-threaded Java application invoking Pounamu/Thin components with diagram request and update operations. We used the server-side SVG diagramming servlets as the target Pounamu/Thin components, assuming no client-side editing support. Performance results indicate a response time to the user of between 0.5-1.5 seconds can be supported for only around 4-7 concurrent users depending on the complexity of the editing operations being performed and the size of the diagram. The current architecture of Pounamu/Thin thus provides good response time for a small number of concurrent users, typical of what is needed for shared design diagram editing. The web components, both in the Tomcat and MUPE servers, can be multi-threaded and the servers themselves can be replicated to provide load-balancing and fault-tolerance. Caching of Pounamu XML diagram data structures in the web server also reduces requests for this data from Pounamu and allows a large number of users to concurrently browse, but not edit, diagrams. Using client-side scripting minimises server loading for simple operations such as move and resize of shapes, but adding shapes and connectors and modifying their properties must produce web component POST operations, which then result in Pounamu application server RMI calls. SVG, VRML and MUPE all support client-side DOM representations of their screens. These diagram data models could be incrementally updated instead of completely re-displayed after editing operations. For large diagrams this would significantly reduce the time to re-render the diagram in the client but also reduce network traffic between the client and Pounamu/Thin server as only changed document content could be sent to the server.

As all diagram update requests to the Pounamu application server must be serialised, there is limited scope for scaling this part of the system for editing any particular model. The Pounamu meta-tool holds both the tool definitions and shared diagram and model data structures and applies complex logic when updating diagrams to modify the model and related diagram elements. This means while editing requests can be buffered in the client browser by scripts or in the web server for buffered editing, eventually they must be applied to the shared application server via its RMI API. Any buffering of edits runs the risk of concurrent user modification of the same diagram elements and either lost updates or the need to merge multiple updates. In earlier work we used a peer-to-peer system to provide collaborative editing for multiple users of the thick-client Pounamu tool (Mehra et al, 2004). It may be possible to use a similar approach to have multiple Pounamu applications running and load-balance Pounamu/Thin requests across these, using the peer-to-peer collaborative editing support infrastructure to keep the Pounamu application data consistent in an asynchronous fashion. Replication of the Pounamu server is also possible when editing different models provided tool specifications are not being concurrently altered by one of the replicated Pounamu instances. This is the most viable short term route to scalability, at the expense of liveness of tool specifications, as only limited numbers of users are likely to be editing any one model at a time.

Key areas for future work we are planning include extending our work to cover more types of devices with other styles of input, such as stylus input on PDA and Tablet PC and speech input. We are keen to see how diagram-based visual design tools benefit from these alternative interaction mechanisms. As we discussed above, specifying representations of diagrams will definitely be improved by using stylus input.

A key issue we have had to address for each of our Pounamu/Thin components is *how much computation is deployed to the client*, or in other words, *how thin can the client be while still being usable?* In the web browser case, we experimented with a “fully thin” approach and one combining additional plug-ins (SVG and VRML) and scripting. It is clear from our user evaluations that the fully thin case leads to low usability rating by end users due to its inherent latency problems. The issue then is what technology provides sufficient local computational power for usability, while minimising both payload cost, and versioning issues. Our feeling is that there is no correct answer here, and that a variety of technologies could and should be used for this purpose. Our SVG + ECMA script approach would be quite similar in approach to one based on Ajax/DHTML, Laszlo/Flash, or small applet/DOM. Similarly for the mobile deployment case, the MUPE MIDP client provided sufficient computational power for our needs but other clients could be equally effective. This suggests a program of future work opportunities around refactoring and generalising the Pounamu/Thin platform.

Unifying the Pounamu/Thin content generation and request handling components into a single set of Tomcat web server components is desired to enable sharing of request processing, content generation and support for individual user preferences. At present, the GIF and SVG generators use custom code to generate content whereas XSLT or another transformation/content generation technologies like Eclipse Java Emitter Templates or Apache Velocity could be alternatives. Similarly, only the MUPE-based mobile clients allow end users to specify their own custom rendering for shapes and this would be useful in SVG and VRML-based renderers. In addition, we believe the multi-level zooming facility employing multiple levels of detail used by the MUPE-based mobile device interfaces would be useful in Pounamu tools in general. We are investigating generating Laszlo interface descriptions instead of SVG as these compile to Macromedia Flash implementations providing much richer web browser interactions and content, while minimising some of the browser inconsistency isses we have noted. We would like to apply our approaches to mobile technologies other than MUPE, such as Mobile SVB, Symbian development framework in C++, and Microsoft .Net Mobile version. We want to improve the layout algorithm used for overview diagrams to better produce layouts for those diagram types that rely heavily on spatial relationships. Potential approaches may include spatial reasoning and other related approaches on mobile devices (Wobbrock et al, 2004). We would like to integrate some of our previous work on collaborative editing support (Mehra et al, 2004) to all the Pounamu/Thin user interfaces. This would provide richer support for group awareness and synchronisation between multiple users of a Pounamu tool. Further investigation is needed to enable scaling of the system architecture beyond a handful of concurrent users. Finally, we see our work as one step in the evolution of a more generic framework for multi-device, multi-platform diagram editing and visualisation support. The work reported here has provided the basis for such a general component based framework. However, using the framework as it currently stands is cumbersome, requiring significant programming effort to introduce a new interface technology, such as Ajax, Laszlo or Symbian. Our next major step, therefore, is to refactor the framework and abstract a set of configuration languages and plug-in points permitting more straightforward adaptation of the framework both for the addition of new thin client interface technologies and for use by metatools other than Pounamu.

SUMMARY

We have described our experiences in developing architectures and solutions for thin client modelling and diagramming tools. Our approach has been to develop several exemplar thin client interfaces to a metatool permitting any tool specified and generated by the metatool to be deployed, without additional programming effort, using any of those thin client interfaces. We have developed interfaces for GIF, SVG and VRML, using web browser clients, and for MIDP compatible phones, using Nokia’s MUPE technology. In the process we have experimented with combinations of client and server side scripting and undertaken a number of user evaluations of the effectiveness of our approaches, all demonstrating that effective thin client interfaces can be automatically generated for tools specified using the metatool. From this experience we have partially refactored our exemplar solutions into a more generic framework, with significant component reuse, and are now working on a more complete refactoring that will lower the cost of reusing the framework through the addition of a set of configuration languages and plug-in points.

ACKNOWLEDGEMENTS

The authors acknowledge the assistance of the New Zealand Foundation for Research Science and Technology in funding this project. We also acknowledge the support of our industry partners in providing relevant tool case studies, and our evaluation subjects who willingly gave their time. Hermann Stoeckle acknowledges the University of Auckland Computer Science Department for scholarship assistance.

REFERENCES

1. Baudisch1, P., Xie, X., Wang C., and Ma, W.Y. (2004): Collapse-to-Zoom: Viewing Web Pages on Small Screen Devices by Interactively Removing Irrelevant Content. In Proceedings of the 2004 ACM Conference on User Interface Software Technology.

2. Bentley, R., Horstmann, T., Sikkel, 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.
3. Bonifati, A., Ceri, S., Fraternali, P., Maurino, A. (2000) Building multi-device, content-centric applications using WebML and the W3I3 Tool Suite, Proc. Conceptual Modelling for E-Business and the Web, LNCS 1921, pp. 64-75
4. Burnett, M. Atwood, J., Walpole, R. and Reichwein, J. (2001): Forms/3: A first-order visual language to explore the boundaries of the spreadsheet paradigm, Journal of Functional Programming, 11(2), March 2001, pp. 155-206.
5. Buyukkoten, O., Garcia-Molina, H., Paepcke, A., and Winograd, T. (2000): Power browser: efficient web browsing for PDAs. In Proceedings of CHI 2000 Conference on Human Factors in Computing Systems, pp. 430–437.
6. Cao, S., Grundy, J.C., Stoeckle, H., Hosking, J.G., Tempero, E., Zhu, N. (2005): Generating Web-based User Interfaces for Diagramming Tools, In Proceedings of the 2005 Australasian User Interfaces Conference, Jan 31-Feb 3, 2005, Newcastle, Australia, Conferences in Research and Practice in Information Technology, Vol. 40.
7. Chalk P. (2000): Webworlds-Web-based modeling environments for learning software engineering, *Computer Science Education*, vol.10, no.1, 2000, pp.39-56.
8. Chau, T. and Maurer, F. (2004): Tool Support for Inter-Team Learning in Agile Software Organizations, in Proceedings of the Workshop on Learning Software Organizations 2004, Springer.
9. Chen, Y., Ma, W.Y., and Zhang, H.J. (2003): Detecting Webpage Structure for Adaptive Viewing on Small Form Factor Devices. In Proceedings of WWW 2003, pp 225–233.
10. Cox, P.T., Giles, F.R. and Pietrzykowski , T. (1989): Prograph: a step towards liberating programming from textual conditioning, In Proceedings of the 1989 IEEE Workshop on Visual Languages.
11. 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.
12. Eisenstein, J. and Puerta, A. (2000): Adaptation in automated user-interface design, Proc. 2000 Conference on Intelligent User Interfaces, New Orleans, 9-12 January 2000, ACM Press, pp. 74-81.
13. Evans, E. and Rogers, D. (1997): Using Java Applets and CORBA for multi-user distributed applications, Internet Computing 1(3), 1997, IEEE CS Press, pp. 43-55.
14. 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, pp. 115-128.
15. 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.
16. Graham T.C.N., Stewart, H.D., Kopaee, 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.
17. Green, T.R. (1989) Cognitive dimensions of notations, *People and Computers V*, Sutcliffe, A. and Macaulay, L. Eds, Cambridge University Press, 1989.
18. 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.
19. Grundy, J.C. and Zhou, W. (2003): Building multi-device, adaptive thin-client web user interfaces with Extended Java Server Pages, In Cross-platform and Multi-device User Interfaces, Wiley, 2003
20. Iivari, J. (1996): Why are CASE tools not used?, *CACM*, vol. 39, no. 10, 1996, pp. 94-103.
21. 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.
22. 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.
23. 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, ACM Press, 2002.
24. Luz, S., and Masoodian, M. (2004): A Mobile System for Supporting Non-Linear Access to Time-Based Data. Conference Proceedings of AVI 2004, The 7th International Working Conference on Advanced Visual Interfaces (Gallipoli, Italy, 25-28 May), ACM Press, 454-457
25. Luz, S., Masoodian, M., and Weng, G. (2003): Browsing and Visualisation of Recorded Collaborative Meetings. Conference Proceedings of HCI International '03, 10th International Conference on Human-Computer Interaction (Crete, Greece, 22-27 June), Lawrence Erlbaum Associates, Vol. 2, 148-152.

26. 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.
27. 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.
28. Marsic, I. (2001): An architecture for heterogeneous groupware applications, Proc. International Conference on Software Engineering, May 2001, IEEE CS Press, pp. 475-484
29. Masoodian, M., and Budd, D. (2004): Visualization of Travel Itinerary Information on PDAs. Conference Proceedings of AUIC 2004, The 5th Australasian User Interface Conference, (Dunedin, New Zealand, 18-22 January), Australian Computer Society Inc, 65-71.
30. 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.
31. 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*, Volume 4 , Issue 3, May 2000, pp. 65-74.
32. McIntyre, D.W. (1995): Design and implementation with Vampire, Visual Object-Oriented Programming. Manning Publications, Greenwich, CT, USA, 1995, Ch 7, 129-160.
33. McWhirter, J.D. and Nutt, G.J. (1994): Escalante: An Environment for the Rapid Construction of Visual Language Applications, Proc. VL '94, pp. 15-22, Oct. 1994
34. 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 6th Int. Workshop on Collaborative Editing Systems*, Chicago, 5th Nov 2004.
35. Minas, M., and Viehstaedt, G. (1995): DiaGen: A Generator for Diagram Editors Providing Direct Manipulation and Execution of Diagrams, In Proceedings of VL '95, 203-210 Sept. 1995.
36. MUPE (2006) : Multi-User Publishing Environment (MUPE), www.mupe.net, June 2006.
37. Palm Corp. (2001): Web Clipping services, http://www.palmos.com/dev/tech/webclipping/, accessed 10th December 2006.
38. Quatrani, T. and Booch, G. (2000): *Visual Modelling with Rational Rose™ 2000 and UML*, Addison-Wesley.
39. 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.
40. Rossel M. (1999): Adaptive support: the Intelligent Tour Guide. 1999 Int. Conf. Intelligent User Interfaces. ACM. 1999, New York, NY, USA.
41. Stephanidis, C. (2001): Concept of Unified User Interfaces, In User Interfaces for All - Concepts, Methods and Tools, Laurence Erlbaum Associates, pp. 371-388.
42. 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.
43. Sun Microsystems, Project Looking Glass, http://www.sun.com/software/looking_glass, accessed October 20 2006.
44. Van der Donckt, J., Limbourg, Q., Florins, M., Oger, F., and Macq, B. (2001): Synchronised, model-based design of multiple user interfaces, Proc. 2001 Workshop on Multiple User Interfaces over the Internet.
45. Wobbrock, J., Forlizzi, J., Hudson, S., Myers, B. (2002): WebThumb: interaction techniques for small-screen browsers. In Proc.UIST '02, pp. 205–208.
46. Zhao, D., Grundy, J.C. and Hosking, J.G. (2006): Generating mobile device user interfaces for diagram-based modelling tools, In Proceedings of the 2006 Australasian User Interface Conference, Hobart, Australia, January 2006
47. 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.