An architecture for generating web-based, thin-client diagramming tools

Shuping Cao[1], John Grundy[1, 2], John Hosking[1], Hermann Stoeckle[1] and Ewan Tempero[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, herm, ewan}@cs.auckland.ac.nz

Abstract

Thin-client visual design tools can provide a number of advantages over traditional thick-client diagramming tools but are challenging to build. We describe a component-based extension to a thick-client meta-CASE tool that we have been developing that allows any specified diagram editor to be realised as a thin-client tool

Keywords: Thin-client diagramming, meta-CASE, web-based CASE tools, component-based tool extensions

1. Introduction

Traditional CASE (Computer-Aided Software Engineering) tools use a thick-client, desktop interface and architecture. These work well to provide highly responsive diagram editing and viewing, leverage sophisticated thick-client interaction techniques and typically manage information on local workstations in a distributed fashion [1, 3, 5]. Disadvantages of this approach include a need to install and update software on every user's workstation, the complexity and learning curve associated with using many CASE tool interfaces, the complex, heavyweight architectures needed to support collaborative editing, and lack of support in most tools for modifying diagramming notations and semantics [6, 7, 4].

Thin-client diagramming tools use a web browser for editing diagrams. Users view diagrams as content of a "web page" that also includes controls such as buttons and links for modifying the diagram or moving to other diagrams [2, 7]. Diagrams may be constructed with combinations of automatic and manual layout, resizing, highlighting and so on. Technologies to render the diagrams might include GIF images, SVG renderings, and VRML visualisations. Key potential advantages of this approach are a consistent look and feel across all web-based diagramming tools, use of web page-based interface design techniques to aid interaction and learning, no need to install and update copies of tools on every workstation, and use of conventional web application architectures to support collaborative work by multiple users.

We have been developing Pounamu, a meta-CASE tool that provides very flexible diagram and meta-model specification techniques. Pounamu has been developed with a traditional thick-client architecture. We wanted to experiment with thin-client diagramming support for Pounamu-based diagramming applications. In this short paper, we describe how we were able to provide an optional extension to Pounamu that allowed users to choose to access generated diagramming tools via a web-based, thin-client interface generated from the tool specification.

We begin the remainder of this paper by motivating this work and reviewing related diagramming tool research. We then outline our approach as an extension to our thick-client meta-CASE tool and describe the architecture of our system. We illustrate the thin-client diagramming support it provides with parts of a UML CASE tool, and discuss key design and implementation decisions. We then summarise our results.

2. Motivation

We have been developing a new thick-client meta-CASE tool called Pounamu. This provides a set of tools for designing shape and connector appearance, meta-model elements, views of meta-model elements using shapes and connectors, and event handlers for specifying semantic behaviour [8]. Figure 1 (a) shows some of the tool design facilities from Pounamu in use. These include a shape designer, meta-model designer view designer and event handler designer. Figure 1 (b) shows an example diagramming tool generated by Pounamu, a Unified Modelling Language (UML) CASE tool, being used. A thick-client interface is provided for all Pounamu tools, which includes an element tree (1), pop-up and pull-down (2) menus, drawing canvas (3), shape property editor, status window (4), and directly-manipulable shapes (5) and shape elements.

Once a tool has been specified using Pounamu, the specification is then executed by Pounamu itself to provide the new tool's behaviour. Pounamu has a traditional thick-client architecture, and so any tools specified with it have the same kind of interface.

The question we address in this work is whether we can use a thin-client approach, specifically whether we can create a web-based interface to exactly the same tool specification and using the existing Pounamu system as the execution engine for the specification. Our approach is to

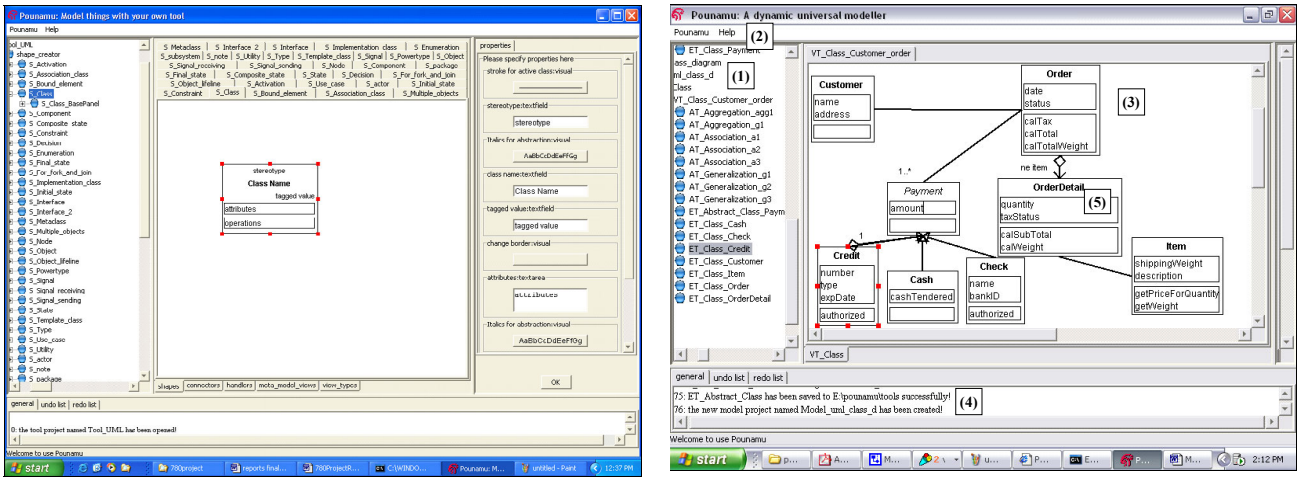

Figure 1. (a) Pounamu meta-tool designer tool and (b) a Pounamu-specified thick-client UML design tool in use.

provide a set of web components developed independently of the Pounamu meta-tool itself that access a remote API provided for Pounamu. We call the resulting system Pounamu/Thin.

3. Pounamu/Thin Architecture

Initially a tool designer specifies a diagramming tool using the thick-client tool design facilities of our original Pounamu meta-CASE tool. These tool specifications are saved to an XML-based tool repository for use by the Pounamu tool interpreter. Pounamu and the Pounamu/Thin web components are deployed and the tool specifications loaded by Pounamu and interpreted to provide the specified diagramming tool. The Pounamu/Thin web components interact with Pounamu via a remote web services-based API. Users connect to the Pounamu/Thin web components via conventional web browsers if using GIF-based diagram rendering components. SVG-based rendering components are also provided, but require an SVG plug-in in the browser. As Pounamu is a meta-CASE 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.

Figure 2 illustrates the architecture of our Pounamu/Thin. The Pounamu tool runs on a host and behaves as the application server/database manager. Tool specifications are loaded from XML repositories and the diagramming tool configured from these. Multiple views of a model are provided by Pounamu, with the diagram views being the point of interaction. View data can be converted into an XML or a GIF image format. Pounamu uses a set of "Command objects" to represent edits to be made to a view (or model) elements. These have execute(), undo(), redo() methods which when invoked carry out the specified modification of Pounamu data structure state and re-render affected views. These command objects can be created and be sent to a Pounamu view programmatically in order to modify it. We have provided an RMI API to enable Pounamu views to be created and modified remotely via command objects, and lately have extended this with a SOAP-based web service API.

Pounamu/Thin is a set of web components hosted by an Apache web server. A set of Java Servlets provide the diagramming interface, and include user login, view manipulation (create, display etc), diagram rendering and manipulation, and diagram shape or connector property editing facilities. A set of JavaBeans are used to provide support infrastructure, including GIF-based diagram rendering, SVG-based diagram rendering, diagram editing by Pounamu Command object construction and an SVG edit command cache. Multiple users can view and edit the same diagram simultaneously, the web component servlets providing co-ordination and sequencing of interactions to the single Pounamu tool acting as the shared application server. Users interact with the Pounamu/Thin diagramming tools via web browsers. If the SVG diagram rendering is wanted by a user, an SVG plug-in must be present in their browser.

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 SOAP API is used to synchronise their views with the "shared" views used by the thin-client diagram editing web components.

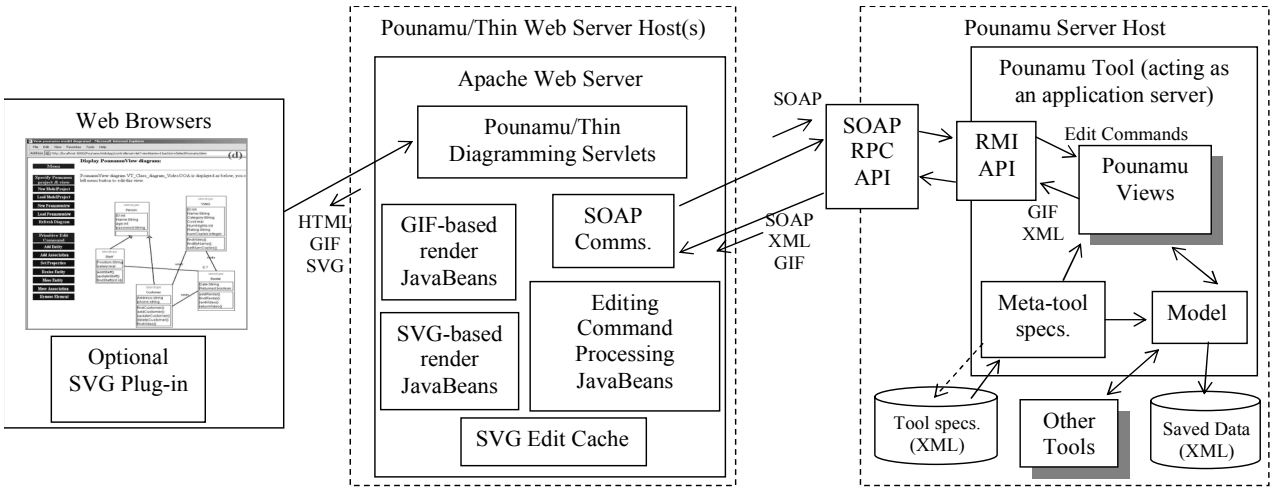

Figure 2. Architecture of Pounamu/Thin.

4. Example Usage

In this section we illustrate the user interface of our thin-client diagramming tools with an exemplar tool, the UML design tool shown in Figure 1.

Figure 3 shows the web client user interface of Pounamu/Thin when selecting a class diagram view to edit (1-2) and the resultant rendered diagram (3), in this example using a GIF image to render the Pounamu class diagram view. This web browser-based user interface includes a set of buttons to control the project (add views, open views) (4); a set of buttons to manipulate the diagram (add shapes and connectors, move and resize elements, edit element properties) (5) and a diagram, which can be clicked on to manipulate it (6).

Moving and resizing shapes requires several interactions with the diagram via the browser window. The user firstly indicates a desire to move or resize an item by selecting the Move or Resize Entity button, then selects the element to manipulate. The user then indicates the position to move the element to (or place to resize a selected corner to) by clicking in the location desired. The specified movement or resize operation is enacted by Pounamu/Thin and the modified diagram is redrawn.

We have also developed an SVG-based diagram renderer. The Pounamu/Thin servlets providing this interface query the Pounamu server for an XML encoding of the view and transform this into an SVG representation. An SVG plug-in in the web browser is required to render the diagrams in the browser. This results in more crisply rendered design diagrams that look very close to the Swing-based diagram rendering in our original thick-client Pounamu tool. An additional capability provided by our SVG diagram rendering servlets is an option for buffering of edits on the diagram in the servlets. Users may make a number of changes to their diagram without immediately communicating them to the Pounamu application server. These are stored in the SVG edit cache. Once the user is satisfied with the changes, they can then be committed back to the server.

5. Implementation Issues

Pounamu is implemented in Java and little effort was required to define and implement the SOAP-based API extension to its existing RMI interface. Java Swing was used to implement Pounamu's thick-client view editors. Swing provides a mechanism for generating GIF images of views of diagram components. This facility is used for both printing Pounamu diagrams and copying them to other applications by the thick-client tool, and is also used for the GIF-based version of Pounamu/Thin.

Pounamu views are saved to an XML format by the tool in order to support both saving and loading views and also to support thick-client editing of views via heavy weight distributed message passing between multiple instances of Pounamu. In addition, Pounamu view editing Command objects can also be converted to and from an XML format. We used both of these facilities to support the editing of Pounamu views and to support SVG-based view rendering. The Pounamu/Thin diagram editing facilities convert user interactions with diagrams and property forms into XML-format Pounamu Command objects. Our SVG diagram renderer and editor behaves 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. The SVG renderer instead requests an XML-encoded version of the view from the Pounamu server, which it then traverses to generate the SVG. We implemented the SVG translator in Java rather than using XSLT transformation scripts as we found using Java to provide much faster performance and because this process required quite complex algorithms.

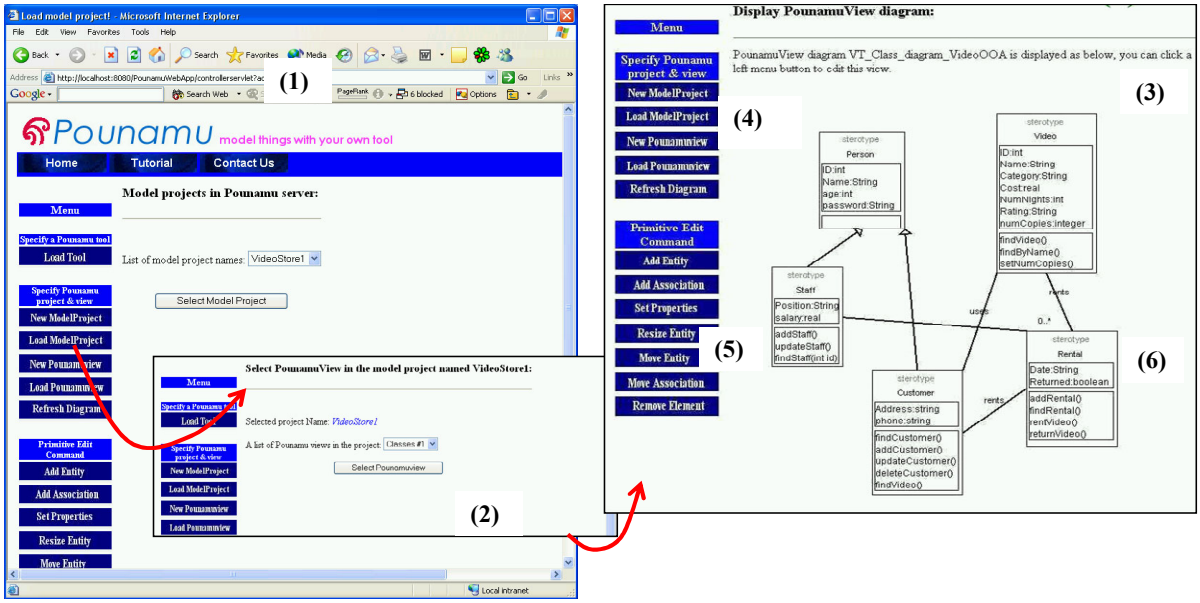

Figure 3. Web client user interface of the Pounamu/Thin UML tool.

6. Summary

We have developed a thin-client diagramming extension for a meta-CASE tool, allowing any specified tool to have either the original thick-client editing or a web-based thin-client editing approach. Our extension is a set of Java servlets implementing thin-client diagram rendering via GIF images with page-based diagram display and editing interaction, or SVG rendering with optional drag-and-drop move/resize and edit buffering. We use the original meta-tool interpreter to provide a shared application server for the thin-client tools, and multiple user diagram viewing and editing is supported using this conventional web-based architecture. Evaluation of our thin-client and thick-client tools indicates both provide equivalent diagramming facilities and the thin-client versions do provide acceptable response-time and user interaction, with limited advantages in the areas of no installation overhead, less of a learning curve and web-based multiple user diagram editing support.

References

1. Ferguson, R.I., Parrington, N.F., Dunne, P. Hardy, C., Archibald, J.M. and Thompson, J.B. MetaMOOSE - an Object-Oriented Framework for the construction of CASE tools, Information and Software Technology, vol. 42, no. 2, January 2000.
2. Gordon, D., Biddle, R., Noble, J. and Tempero, E. A technology for lightweight web-based visual applications, In Proceedings of the 2003 IEEE Conference on Human-Centric Computing, Auckland, New Zealand, 28-31 October 2003, IEEE CS Press.
3. Graham T.C.N., Stewart, H.D., Kopaee, A.R., Ryman, A.G., Rasouli, R. A World-Wide-Web architecture for collaborative software design, In Proceedings of the Ninth International Workshop on Software Technology and Engineering Practice (STEP '99), IEEE CS Press, 1999, pp.22-29.
4. Grundy, J.C., Mugridge, W.B. and Hosking, J.G. Constructing component-based software engineering environments: issues and experiences. Information and Software Technology 42, 2, January 2000, pp. 117-128.
5. Kelly, S., Lyytinen, K., and Rossi, M., Meta Edit+: A Fully configurable Multi-User and Multi-Tool CASE Environment, In Proceedings of CAiSE'96, Lecture Notes in Computer Science 1080, Springer-Verlag, Heraklion, Crete, Greece, May 1996, pp. 1-21.
6. Khaled, R., McKay, D., Biddle, R. Noble, J. and Tempero, E., A lightweight web-based case tool for sequence diagrams, In Proceedings of SIGCHI-NZ Symposium On Computer-Human Interaction, Hamilton, New Zealand, 2002.
7. Mackay, D., Biddle, R. and Noble, J. A lightweight web based case tool for UML class diagrams, In Proceedings of the 4th Australasian User Interface Conference, Adelaide, South Australia, 2003, Conferences in Research and Practice in Information Technology, Vol 18, Australian Computer Society.
8. Stoeckle, H., Grundy, J.C. and Hosking, J.G. Approaches to Supporting Software Visual Notation Exchange, In Proceedings of the 2003 IEEE Conference on Human-Centric Computing, Auckland, New Zealand, October 2003, IEEE CS Press.