Realising Web Diagramming Applications with OpenLaszlo and Eclipse

Tony Ip[1], Kelvin Lomberg[1], John Grundy[1, 2], John Hosking[2], Jun Huh[2]

[1]Department of Electrical and Computer Engineering
[2]Department of Computer Science
University of Auckland, Auckland, New Zealand
{john-g, john}@cs.auckland.ac.nz

Abstract

Rich web client diagram editors have a range of applications but are very challenging to design and build. We describe Marama/Thin, a rich internet application diagramming toolkit than realizes Eclipse-based diagram editors in web browsers using OpenLaszlo. The Eclipse-based Marama meta-tools are used to produce a high-level diagramming tool specification and implementation in Eclipse. Then a Flash-based editor for this tool, communicating with Eclipse, is realized using the OpenLaszlo platform. We describe our approach, an example diagramming tool specification and usage, architecture and implementation, strengths and weaknesses of our tool, and future directions in rich web diagramming meta-tools.

Introduction

Diagramming tools have a myriad of applications including illustrations in documents and presentations, Computer-Aided Design, desktop publishing, and various domain-specific applications like music scores, education, and software design. Traditionally diagramming applications have been provided to end users as thick-client, desktop applications. Examples include MS Visio™, CAD tools, UML design tools, and drawing components in applications like Word and Powerpoint. This was primarily because these applications require advanced graphics and highly responsive user interaction which until recently was difficult to achieve in web applications. However, as more and more applications are realized with web-based user interfaces, demand has grown for rich internet application (RIA) web diagramming applications.

Approaches to providing such web diagramming applications have ranged from early post-reply examples using images and image maps [8], to custom-built Flash and DHTML/JavaScript platforms [2]. The later provide highly interactive, often desktop-like capabilities but like desktop diagramming applications are very challenging to build and update. Desktop meta-tools have been developed to enable highly customisable domain-specific diagramming tools to be realised, often by generating implementations from high-level specifications [711]. However, very few examples currently exist of such tools for rich web client diagramming applications.

We describe Marama/Thin, a RIA diagramming tool with meta-tool capabilities. A visual, high level diagramming tool specification is interpreted to realize a domain-specific diagramming editor using Flash or DHTL/JavaScript in a web browser via the OpenLaszlo platform [9]. We use Marama's

Eclipse-based meta-tools to specify the diagramming tool; a set of OpenLaszlo web components queries the specification and realizes a RIA diagramming tool implementation. We use an Eclipse instance as an application server allowing multiple users to collaboratively edit a diagram. In the following sections we describe the major motivations for our work and key related work. We outline our approach and show examples of interacting with our RIA diagramming tool. We outline our architecture and implementation approaches and describe evaluation of our tool, its key strengths and weaknesses, and areas for future extension.

Motivation

Consider common diagramming applications like MS Visio™ (Figure 1 (a)) and Freehand™ (Figure 1 (b)). These provide users with highly interactive diagram editors featuring drag-and-drop, style tool palettes, automatic layout and detailed property views. Some powerful RIA diagram editors have been developed to try and provide similar capabilities, for example Gliffy (Figure 1 (c)) and Pounamu/Thin (Figure 1 (d)). However, most of these usually have fixed diagram type(s) and functionality, sometimes have severe limitations on user interactivity, and often do not support collaborative diagram editing.

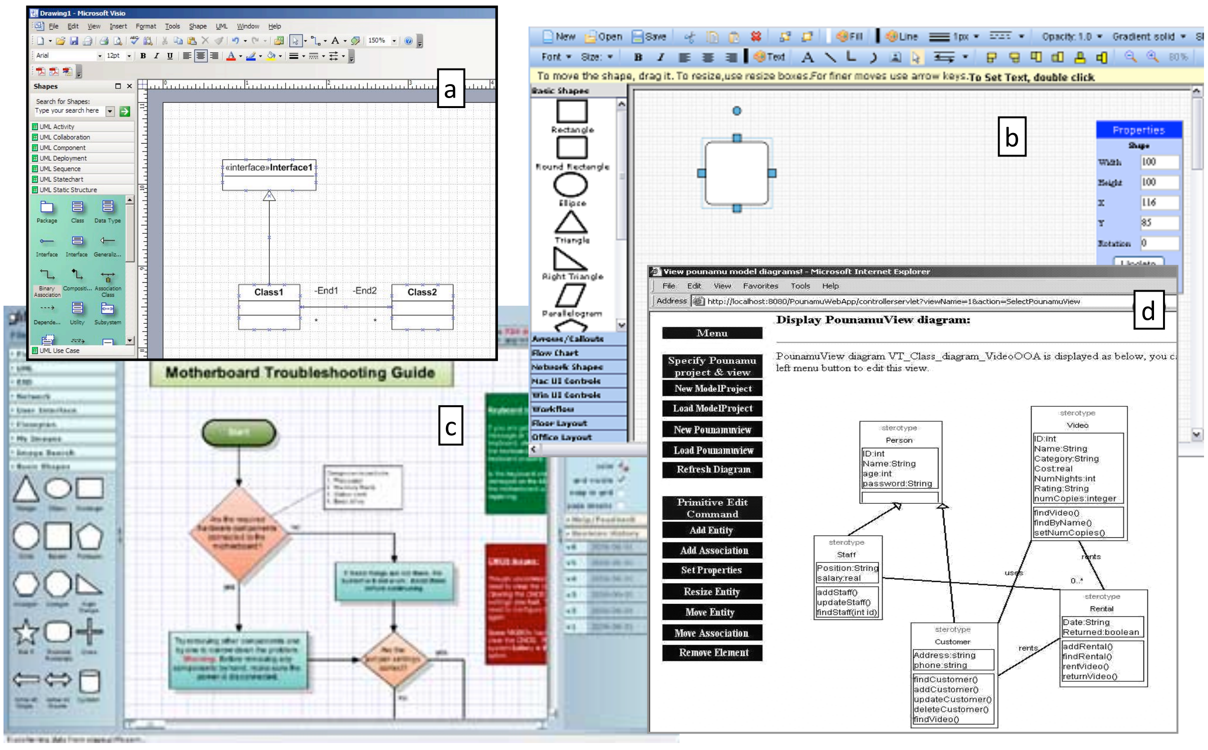


Figure 1. Examples of desktop and rich internet diagramming tools.

We wanted to develop a “next generation” of generated RIA diagram editors that incorporate: meta-tool capabilities allowing end users to modify diagram types, elements, look and feel and underlying models; run in any browser; have desktop-like rich interaction capabilities like drag-and-drop, fast user response etc; support complex diagramming capabilities; and support collaborative editing among several users concurrently.

Sidebar: Diagramming tools

Diagramming applications have a long history for both general-purpose use and special-purpose domains. Tools such as MS Visio™ [10], PowerPoint™ and Freehand™ provide general-purpose diagramming support for a very wide range of possible applications. Various domain-specific tools have also been developed, for example Enterprise Architect software design [12] and Labview™ for instrumentation [6].

As such diagramming tools are complex to build and evolve, a number of platforms have been developed for building such tools and for developed "meta-diagramming tools". For example, Eclipse GEF and GMF [3] provide powerful frameworks for building rich desktop diagram editors on the Eclipse platform. MetaEdit+ [7] provides a meta-tool for CASE tool development, and Marama [5] and DiaGen [11] provide meta-tools for a wide range of diagram-centric applications.

Web-based diagramming applications offer the attraction of wide distribution and accessibility via the browser platform, no need to install and maintain desktop diagramming tools on each users machine, and the possibility of collaborative diagram viewing and even editing using web architectures. A good example of early web diagramming tools is Seek, a lightweight web sequence diagram editor [8]. More recent web diagramming tools include those in GoogleDocs and Gliffy [2]which provide RIA diagramming experiences similar to desktop applications. A drawback of these applications is lack of end user customisation without programming.

Ideally web-based RIA diagramming tools would provide meta-tool capabilities enabling end users to modify their diagramming applications or even create whole new ones. Early examples of such meta-web diagramming tools with limited user editing experiences include Clicki [4] and Pounamu/Thin [1]. These utilised, among other technologies, SVG and JavaScript to provide complex diagramming support within web browsers.

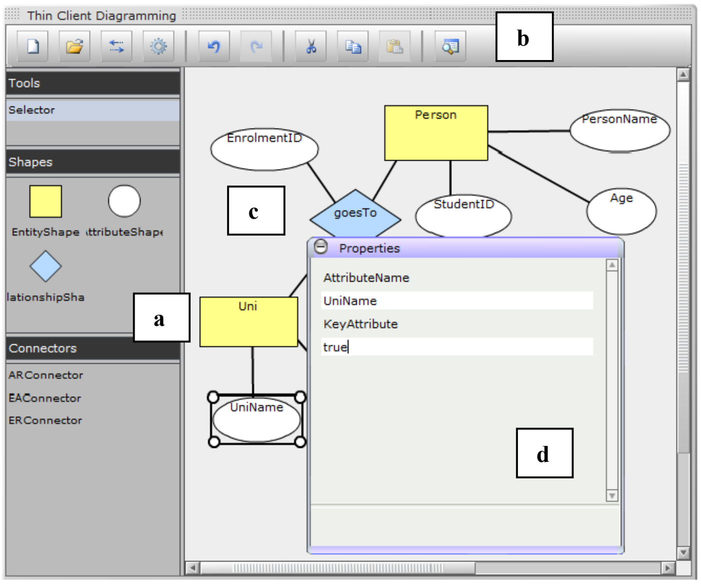
Our Approach

We took an existing Eclipse-based meta-tool, Marama [5], which provides a set of meta-tools for specifying complex diagram editors, and added a RIA diagramming component to Marama. We did this by providing a set of services within Marama allowing a remote client to query diagram specifications (diagram meta-descriptions) and state (diagram model data) via XML messages. The remote clients can also update the state of these diagram models i.e. modify the diagrams. We then implemented a RIA component using the OpenLaszlo framework [9]. We chose OpenLaszlo as it provides support for highly interactive RIAs; compiles to Flash or DHTL/JavaScript, thus running on a variety of web platforms; and lends itself to building collaborative applications using a web architecture. We host the OpenLaszlo Marama/Thin components using a Tomcat web server. Client browsers request the Marama/Thin web diagramming URL and OpenLaszlo provides a compiled RIA diagramming component to the client browser. This uses browser capabilities (DHTL/JavaScript) or a Flash plug-in to provide the Marama/Thin diagramming client. The Marama/Thin client communicates with web server pages to query diagram specifications (meta-data) and diagram state and to update diagram state i.e. when the user edits the diagram in the browser. Queries and updates are sent to an Eclipse instance hosting the Marama plug-ins which acts as a shared application server. Multiple users can be editing the same diagram at the same time. A feature of our approach is that for every different diagram type no new code is written – the Marama/Thin client loads the tool specification remotely from Marama and interprets it to provide a RIA diagramming tool in the browser.

Example Usage

Consider user "Jerry" who wants to do some Entity-Relationship diagramming for a database project. Jerry may choose to create a new diagram with an existing tool (diagram) specification e.g. an ER modelling tool previously specified. Jerry may chose to edit an existing ER model diagram (by herself or concurrently with others). Or Jerry may chose to create a new tool (diagram) specification using Marama's meta-tools – either with Marama's desktop Eclipse client or using Marama/Thin itself.

Jerry choses to open an existing ER model diagram. Jerry logs onto Marama/Thin and then browses the available tool (diagram) types available through a web form, selecting "ER modeller" and diagram "ER model 1". Jerry then begins to edit the diagram. Figure 2 shows an example of the Marama/Thin user interface during this editing process. The Marama/Thin user interface provides a tool palette (a), basic editing command toolbar (b), an ER diagram made up of shapes and connectors that can be manipulated via drag and drop in browser (c), and pop-up property and other view windows (d). In this example, Jerry is editing the properties of an Entity "UniName".



AttributeName
UniName
KeyAttribute
true

Figure 2. Marama/Thin user interface.

Compare the Marama/Thin interface, as shown in Figure 3 (left), with the conventional desktop Marama interface hosted by Eclipse, Figure 3 (right). Save, load, undo etc are supported via a toolbar (a). Drag and drop addition of shapes (b) and connectors between shapes (c) are supported in an identical way to in the Eclipse GEF editor. For example, Jerry selects the "ERConnector" palette item, selects the starting RelationshipShape, click-drags and then releases the mouse over the ending AttributeShape to create a new connector between the two. Similarly, Jerry selects the default Selector tool and clicks on a shape, drags the mouse and releases the mouse button to move the shape. As in the Eclipse desk-top application, behaviours like auto-resize of connectors or auto-move of related shapes are actioned. In Marama/Thin most property sheets and related detail views pop-up automatically when needed or the user asks for them to be displayed e.g. via the Properties button/pop-up menu item. In Eclipse a number of these may be seen juxtaposed beside the diagram editor.

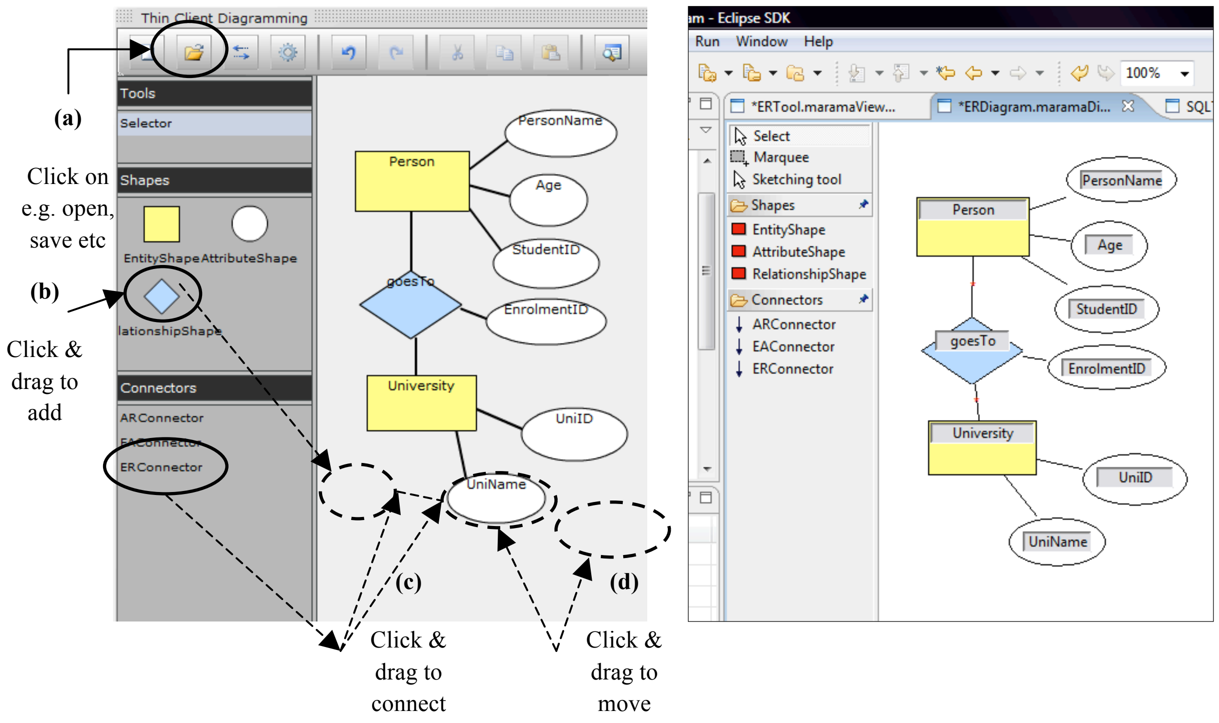


Figure 3. Comparing Marama/Thin (left) vs Marama Eclipse-hosted desktop (right) user interfaces.

As Marama is a meta-tool it allows users to modify their tool (diagram) specifications including shape and connector types, appearance, properties and layout constraints. Users can even specify completely new tool types each with one or more diagram types using Marama's range of meta-tools [5]. These meta-tools include a meta-model editor, specifying tool elements, associations, properties, and constraints; shape designer, specifying shape and connector look and feel; view types, specifying each different kind of diagram available; and constraint and event handlers, specifying complex behaviours of diagram editors and tools. We have provided each of these meta-tools as Marama/Thin web-based RIA diagram editors. This allows Marama/Thin users to modify their diagram specifications – or even create whole new diagramming tools – using their web browser instead of the Marama Eclipse desktop client. In Figure 4 Jerry is viewing the meta-model for the ER Modelling tool. She may add some new constraints to the tool using the Formula meta-model shape, modify existing constraints, or add a new property, element or association.

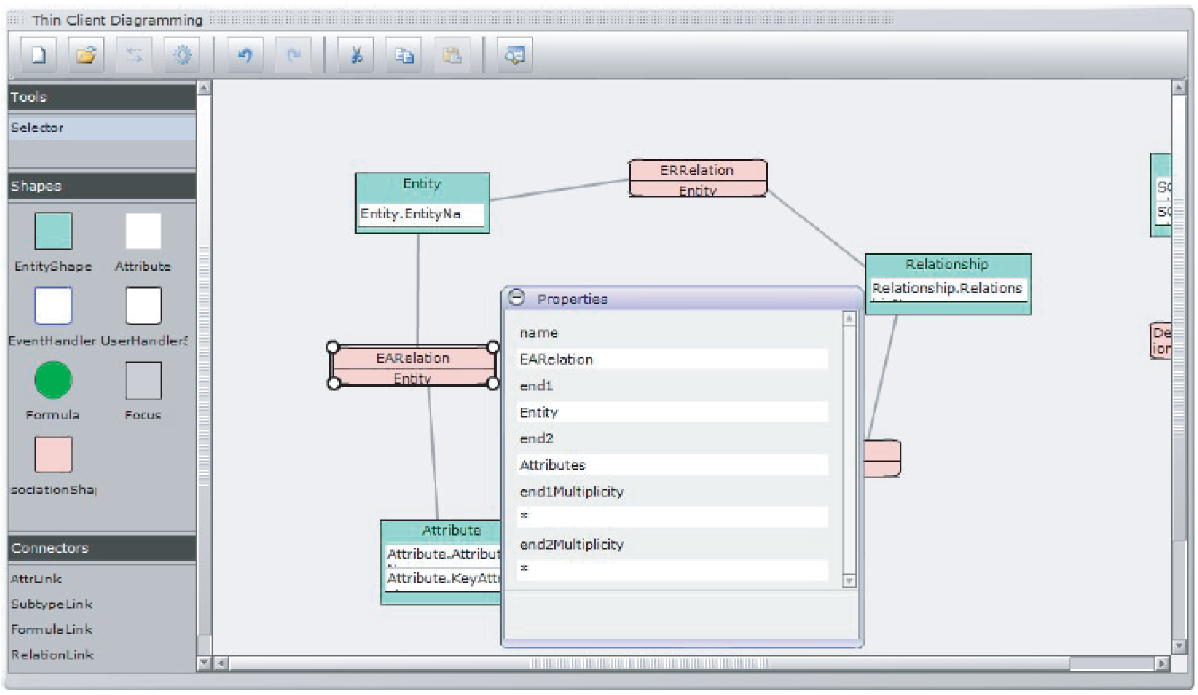


Figure 4. Marama meta-tool (meta-modeller) itself being used in Marama/Thin to design new tool.

We wanted to allow multiple users to collaboratively edit their Marama/Thin diagrams simultaneously to support collaborative design and review. We added infrastructure to provide edit caching, element and connector level locking, and simple animation of other user edits on the same diagram. In Figure 5 (a) Jerry moves the shape "Parent" then connects it to the shape "Brother". In Tom's Marama/Thin client a list of edits made by Jerry is shown and Tom may indicate he automatically wants these applied to his diagram when received, or may chose to review and have these actioned later. Similarly Tom may add a new shape and change its label e.g. to Sister. These edits appear in Jerry's list of other user edits on her diagram and she may have these applied or wish to discuss them via e.g. text or audio chat with Tom.

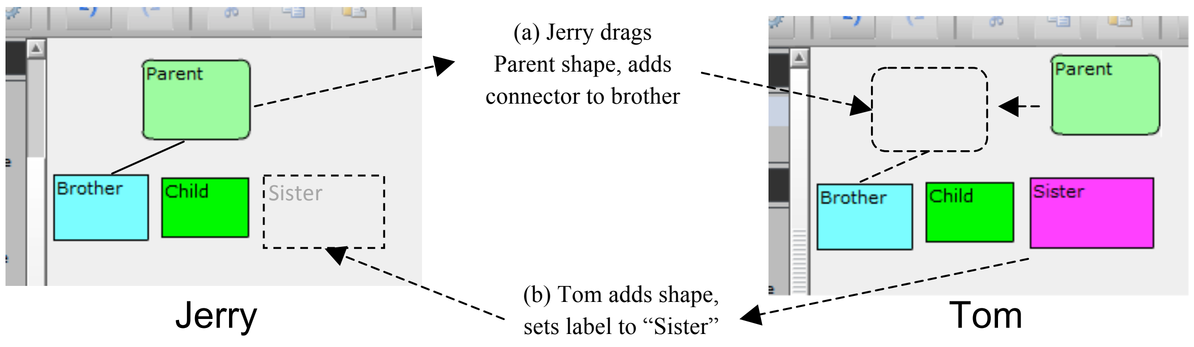

Parent
Brother
Child
Sister
Jerry
(a) Jerry drags Parent shape, adds connector to brother
(b) Tom adds shape, sets label to "Sister"
Parent
Brother
Child
Sister
Tom

Figure 5. Simple collaboration example.

Architecture and Implementation

Figure 6 shows the architecture of our Marama/Thin RIA diagramming tool. We use our existing Marama meta-tool, a set of Eclipse plug-ins, as the “application server” for multiple users (a). This is because Marama provides meta-tool specification capabilities, save/load to XML, complex diagram editing behaviour management, and extensible capabilities to integrate with other Eclipse tools. We developed a set of API services to allow querying of meta-tool specifications in an XML format, querying of diagram state and update of diagram state (b). These interfaces are Marama-independent i.e. Marama could be replaced by another thick-client tool e.g. Visio without the rest of the architecture – or users - being aware of this. We used OpenLaszlo to specify and generate the Marama/Thin diagramming application and implemented the API services using a set of Java Servlets used by this application (c). These all run on a Tomcat web server. The OpenLaszlo-generated RIA application queries the API services to obtain diagram specifications, diagram state and updates diagram state using a set of XML messages and XML Schema. Each user has a browser hosting the RIA diagramming client generated by the OpenLaszlo Marama/Thin specification (d). For this project we used a Flash implementation as we found this provided the “nicest” diagramming client, but a DHTML/JavaScript client can also be generated by OpenLaszlo. Multiple users can be viewing and editing the same diagram at the same time. Each user gets their own client RIA application copy and their own web server thread. Each web server thread communicates with the single instance Marama application which centralises the diagram and tool data and manages synchronisation of updates. This approach supports synchronous editing. Diagrams can also be versioned for asynchronous editing and changes later merged. A “command log” is managed by the Marama application to support collaborative editing (e), along with some basic Command serialisation, grouping and locking support.

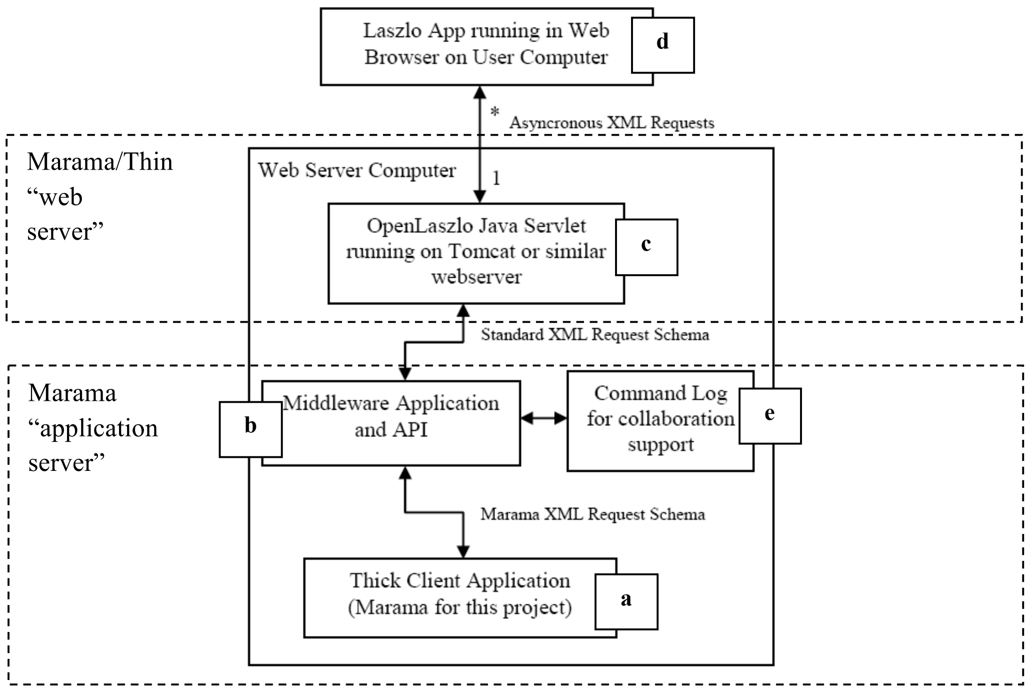

Laszlo App running in Web Browser on User Computer
d
*
Asyncronous XML Requests
Marama/Thin “web server”
Web Server Computer
1
OpenLaszlo Java Servlet running on Tomcat or similar webserver
c
Standard XML Request Schema
Marama “application server”
b
Middleware Application and API
Command Log for collaboration support
e
Marama XML Request Schema
Thick Client Application (Marama for this project)
a

Figure 6. Overview of Marama/Thin architecture.

For this project we chose Marama as the desktop meta-tool diagramming application for the central application server, due to its powerful extensibility and meta-tool facilities. We implemented the middleware application APIs using Java RMI and web services and developed XML schema to represent generic diagram specifications, state and edit Commands intended to be independent of Marama's particular internal representations. We store the Commands as XML documents to facilitate collaborative editing. We used OpenLaszlo to build the RIA diagram editor due to its expressive power and multiple target platform compilatioj support. We chose to compile the OpenLaszlo to a Flash SWF which is loaded and run by the Flash plug-in in the client browser. We found this produced a more responsive and better user experience RIA diagram editor than the DHTL/JavaScript version generated by OpenLaszlo. We used Java Servlets to implement a set of server-side tool and diagram browser, diagram specification query, diagram state query, edit command, and collaboration functions used by the OpenLaszlo RIA diagram client. Most of these are called asynchronously using AJAX to ensure high user response time in the browser client.

Discussion

We have used Marama/Thin to realise several complex Marama diagramming tools and to build many large diagrams using these applications. Marama/Thin provides a RIA diagramming client for any Marama-specififed diagramming tool, including all of the Marama meta-tool editors themselves. The Marama/Thin RIA diagram editor looks and feels very similar to the desktop client Eclipse editors for the same diagram types. Diagram element creation, deletion, movement, resize, property editing all behaviour in a very similar manner and response times are similar to the Eclipse application. This is a significant achievement. Some complex diagram behaviours e.g. automatic layout or automatic hide/show of elements, complex constraint handling and import/export, must all be done by the backend Marama Eclipse application server. Despite this often the user is unaware that many of these require asynchronous calls to the API web services by the Flash web browser client and then subsequent update of the diagram.

Our approach provides the benefit that users have a diagram editing experience in their web browser very close to the one they experience using the desktop Eclipse-based client. Users may also use the backend Marama meta-tools to modify their diagramming tool or even create whole new diagramming tools themselves. The generated Flash client provides a good user experience and a diagramming platform comparable to most desktop diagramming applications. No code is written to realise vastly different diagramming applications as Marama tool specifications are interpreted by the OpenLaszlo-generated client to realise each tool. Collaborative diagram viewing and editing is supported in natural fashion by each user opening the same diagram and seeing edits made by others shortly after they occur.

As some complex tool behaviours require backend processing by the Marama application server e.g. complex diagram layout, automatic layout and inter-diagram updates using model behaviour processing, a delay between these being processed and diagram impact may occur. In the Eclipse desktop client the user waits for any such diagram updates (typically taking much less than second) before continuing editing. In the Marama/Thin client these updates may take up to several seconds as they depend on latency between the browser, web services and (single instance) Marama application server. We use asynchronous calls from the Flash web browser client to the API web services allowing the user to continue editing, but then conflicts need to be resolved between subsequent user edits and "automatic" updates. A similar issue occurs when multiple users make concurrent conflicting edits to a shared diagram. We resolve both situations by showing the edit history and conflicting edits to the user(s). However, this can be a less than ideal editing experience for the user when they have to undo and redo edits due to conflicts. Currently only Marama is supported as a backend editing application server. Seamless web-based user interface integration with other web-based applications is not supported.

Two related, key extensions we need to further develop for Marama/Thin are improved conflict resolution for asynchronous diagram updates and pushing more complex behaviour handling to the browser hosted client. Asynchronous updates occur when either collaborating users simultaneously edit the same or inter-dependent diagram elements and/or back-end behaviour processing results in further diagram updates after a user edit. We have basic conflict resolution support in Marama/Thin implemented as server-side dependency-based locking (in the Marama application server) and edit command presentation and undo/redo in the browser-based client. Both could be further enhanced to improve the diagramming user experience. Due to the complexity of Marama meta-tools and the internal Marama APIs, localising the pushing complex behaviours to the browser client is very challenging in general. However, some improvements could be made especially concerning diagram-only automatic layout handling and dependent element updating.

We would like to support a wider range of back-end applications e.g. Visio or other drawing tools but this is challenging both in terms of how open the tool APIs are and – related to the issue of pushing some complex editing behaviours to the browser client – this is likely to be even harder to do than for Marama. User interface integration is another area of possible improvement where Marama/Thin's RIA could be seamlessly extended by other tools similarly to how Eclipse's user interface is designed for seamless multiple plug-in integration via the extension points concept. This would allow our Marama/Thin RIA to be better-integrated with other RIAs for improved user experience.

Summary

We have developed Marama/Thin, a Rich Internet Application for diagram editing. Marama/Thin provides diagram editors in a web browser of similar user experience to desktop diagram editors in the Eclipse platform. It uses an OpenLaszlo-generated Flash implementation running in the browser and communicating with web services, themselves communicating with a single Eclipse-based Marama meta-tool instance as a backend application server. Marama/Thin supports extensible diagram application meta-tool capabilities via the Eclipse-based Marama meta-tool. Collaborative diagram editing is supported by multiple users sharing the same backend Marama application server and their own RIA browser client.

Acknowledgements

Support from the New Zealand Foundation for Research, Science and Technology via the Research for Industry programme is gratefully acknowledged.

References

1. Cao, S. Grundy, J.C., Hosking, J.G., Stoeckle, H. and Tempero, E. An architecture for generating web-based, thin-client diagramming tools, In Proceedings of the 2004 IEEE International Conference on Automated Software Engineering, Linz, Austria, September 20-24, IEEE CS Press, pp. 270-273.
2. Gliffy Inc. Gliffy - Create and share diagrams online. Retrieved on September 1 2009 from http://www.gliffy.com
3. GEF, Eclipse Graphical Editing Framework. Retrieved on September 1, 2009 from http://www.eclipse.org/gef/
4. 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.
5. Grundy, J.C., Hosking, J., Huh, J. and Li, N. Marama: an Eclipse meta-toolset for generating multi-view environments, Formal demonstration paper, 2008 IEEE/ACM International Conference on Software Engineering, Liepzig, Germany, May 2008, ACM Press.
6. Johnson, G,W. LabVIEW graphical programming: practical applications in instrumentation and control, 1997, McGraw-Hill School Education Group.
7. 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
8. 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
9. Laszlo Systems Inc. OpenLaszlo | the premier open-source platform for rich internet applications. Retrieved on September 1 2009 from http://www.openlaszlo.org.
10. Microsoft Corporation. Microsoft Office Visio 2007 product overview. Retrieved on September 1, 2009 from http://office.microsoft.com/enus/visio/HA101656401033.aspx.
11. Minas, M. Specifying Graph-like Diagrams with DiaGen, Electronic Notes in Theoretical Computer Science, vol. 72, no. 2, Nov 2002, pp. 102-111.
12. Sparx Systems, UML tools for software development and modelling. Retreived on September 1 2009 from http://www.sparxsystems.com/.