Mehra, A. Grundy, J.C. and Hosking, J.G. Supporting Collaborative Software Design with a Plug-in, Web Services-based Architecture, In Proceedings of the ICSE 2004 Workshop on Directions in Software Engineering Environments, May 25 2004, Edingurgh, Scotland, IEE Press

Supporting Collaborative Software Design with a Plug-in, Web Services-based Architecture

Akhil Mehra[1], John Grundy[1, 2] and John Hosking[1]

Department of Computer Science[1], and Department of Electrical and Computer Engineering[2]
University of Auckland, Auckland, New Zealand
{ameh010,john-g, john}@cs.auckland.ac.nz

Abstract

Collaborative editing enables one or more users to edit artifacts simultaneously over a network. Collaborative editing is important in many kinds of editing tools such as Computer-Aided Design tools, Computer-Aided Software Engineering (CASE) tools, drawing tools, and document editors. We describe a new approach for realising collaborative editing applications using Web Services. We have added this collaborative editing functionality as a component-based plug-in to Pounamu, a meta-CASE tool. We have used this plug-in, web services-based system to provide collaborative editing for a range of visual design environments. We describe the architecture of our approach, key design and implementation issues, and illustrate the feasibility of our approach by implementing it as plug-in Pounamu components and evaluating its effectiveness.

1. Introduction

The emergence of networked computing applications has led to workers increasingly being geographically distributed. Geographically distributed teams have fueled the need for collaborative computing applications [2, 3, 7]. The use of computer based applications for coordination and cooperation among groups of people who attempt to perform tasks or solve problems together is the endeavor of collaborative computing [16]. Collaborative editing is a major research topic within the collaborative computing filed. Collaborative editing of artifacts is important in many kinds of editing tools, including Computer-Aided Software Engineering (CASE) tools [7, 9, 19].

CASE tool users often desire a range of facilities to support collaborative editing of shared diagrams [2]. These include a range of facilities to support asynchronous work, such as version control, configuration management, merging capabilities, and synchronous collaborative editing. Synchronous editing capabilities allow users to closely collaborate while evolving or reviewing system designs. Editing changes made by one user are shown as soon as possible in collaborating users' tools.

There are a large number of CASE tools available that provide collaborative editing facilities [2, 7, 9]. One of the major drawbacks of most current collaborative editing systems is their reliance on proprietary protocols and technologies, thus restricting a system to a particular platform and limiting the ability to collaboratively work with others using different tools [2, 7, 8]. None of the current collaborative editing systems address the need for systems to be interoperable across autonomous and heterogeneous systems [19]. Most lack support for dynamic incorporation of collaboration facilities at run-time, forcing environments to have fixed collaboration support [7].

We have developed a proof-of-concept approach to support the development of collaborative editing applications using Web Services-based technology. The aim of this work is to explore the support provided by web services to realize a range of collaborative editing facilities, dynamic plug-in of collaboration support and ultimately dynamic discovery and integration of heterogeneous collaborative tools. We have extended the Pounamu Meta Tool [18] to provide collaborative editing functionality using Web Services. The main aim of this was to enable any Pounamu design environment to support both synchronous and asynchronous collaborative editing.

We first present the motivation for our work and discuss related research. We then outline our approach of using Web Services to provide collaborative editing in Pounamu. We describe key design and implementation issues and provide an evaluation of our approach. We conclude with a summary of the contributions of our work and directions for future research.

2. Motivation

Collaborative editing entails both writing activities and communication between authors [11]. Co authors may work concurrently on the same document (synchronous) or on replicas of the document (asynchronous), or at different times on the original or copied document. The document types may include text documents, UML diagrams, graphic objects, images, etc [7].

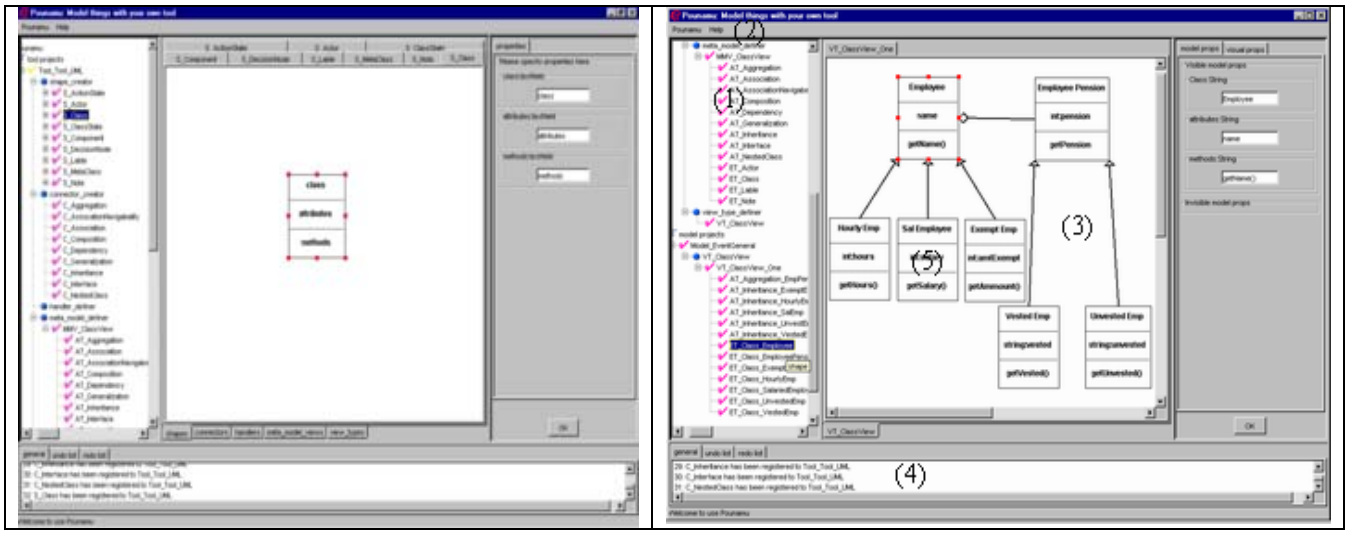

Figure 1. (a) Example of Pounamu shape design view and (b) example of UML tool modeller view.

We choose to provide both synchronous and asynchronous collaboration for the Pounamu Meta Tool based on findings in our previous work [3]. In synchronous editing all changes are immediately visible to all members of the group. Examples of synchronous editors and toolkits for building such systems are GROVE [3], Mule [12], TeamRooms [15] and COAST [17].

Asynchronous editing interaction and modification to a document does not take place in real time. A user may wish to edit a document while another user wishes to work elsewhere. The users or collaborators may request changes made to a document. A user may request or reject changes made to a document. Duplex is a good example of an asynchronous editor [11], as are version control tools like CVS [1]. Some applications support both synchronous and asynchronous work [2, 7, 9], with the ability to move between modes as required.

Most collaborative editing tools have fixed functionality – the collaborative editing support is built in and unchangeable, whether the user wants to use all of its features or not [7]. In addition, most use TCP/IP or remote object technologies like CORBA to achieve data transfer between multiple user distributed applications [2, 6, 12, 14]. These typically have fixed data formats and remote object functionality, making building this infrastructure challenging and difficult to evolve.

We have been developing Pounamu, a new meta-tool to support the specification of multi-user, multi-view visual language-oriented software engineering tools [18]. Pounamu allows software engineers to define new meta-models and meta-views for software tools and to realise tools based on these specifications. Pounamu then provides thick client CASE tools to developers in different parts of software development. Pounamu's design tools include a shape and connector designer, meta-model designer, modelling view designer and event handler designer. Figure 1 (a) shows an example of specifying a UML class icon shape in the Pounamu shape design tool. The Pounamu tool designer specifies a graphical notation for a new tool based on shapes and shape connectors. Shapes can be of arbitrary complexity and may be composed of other sub-shapes.

Figure 1 (b) 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. Each shape can be added to one or more modelling views, with each shape connected to a meta-model entity instance (model instance). Changes to shape properties in a modelling view change model instance properties and vice-versa. Each model instance can associate with shapes in multiple views at the same time.

Our aim in this work was to provide a plug-in, component-based facility supporting both synchronous and asynchronous editing for all visual design tools developed with Pounamu. Although many collaborative editing systems exist, these all use proprietary protocols or technologies and most are static i.e. must be built into a tool. Our approach was to use web services technology to implement a collaborative editing system for Pounamu that could be plugged in at run-time, and could be more easily extended as new Pounamu tool features were developed.

Web services specifications are based on existing Internet standards or specifications that are widely used, thus facilitating seamless interaction between

heterogeneous and autonomous systems [4, 19, 8]. XML-based messages and data formats allow flexible extensions to services, and dynamic discovery and integration support of services at run time offer the possibility of both plug-in services and more dynamic, user-specified features. Some initial research has been carried out using web services to co-ordinate work [8] and support limited forms of fixed-feature, synchronous collaborative editing [19]. Software tool integration and plug-in of collaborative work and other features via web services are in still in initial phases of research.

3. Our Approach

We have been investigating a new approach to support collaborative editing for the Pounamu meta-CASE tool using Web Services. Our main aims have been to (1) provide a dynamic plug-in mechanism for collaborative editing for all Pounamu-developed tools; and (2) to investigate the viability and suitability of Web Services as a technology for the development of such collaborative editing applications.

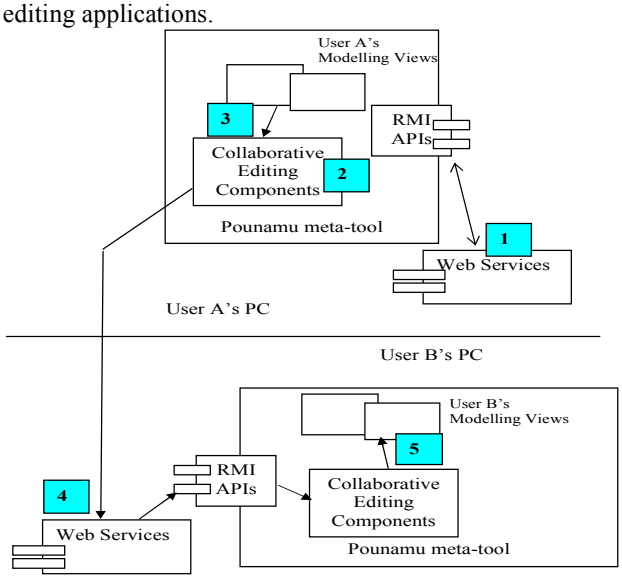

Figure 2. Overview of our approach

Web Services has emerged as the technology of choice when developing remote or distributed components but it still unclear how suitable the technology is for software tool infrastructure development. The main reason so far for Web services success is its ability to be interoperable between diverse languages and platforms [8]. A Web service can theoretically be produced by or consume information from any language/platform and uses standard internet protocols to exchange information [5]. In addition, the ability to dynamically discover and integrate with remote services at run-time has many potential applications in a wide variety of domains, possibly including for software tools.

Figure 2 summarizes the way we went about building our collaborative editing components for Pounamu. We firstly developed a system that exposes existing, remote Java RMI Pounamu APIs as web services (1), utilizing the principles of a Service-Oriented Architecture. This API allows all Pounamu model and project management and editing facilities to be accessed via web service messages, and provides a subscribe-notify structure to handle events. We then went on to build plug-in collaborative editing component that uses these exposed services to achieve collaborative editing in Pounamu (2). This component is plugged into a Pounamu environment to extend it to provide collaborative editing facilities. To date these have included synchronous diagram editing, capture and replay of diagram edits, request and transmission of edits, and diagram merging support. When a user edits modeling views in Pounamu, the collaborative editing component is informed of the edit events (3). Connecting two or more Pounamu environments with collaborative editing components produces a network of peer-to-peer collaborative editing tools. When an edit occurs to a modeling view being shared, the collaborative editing component sends this to the remote Pounamu applications via their web services APIs (4). Received events are translated into update operations on the receiving user's modeling views.

4. Architecture

The Pounamu meta-CASE environment can be broadly broken up into two main parts, Pounamu Tool Projects, and Pounamu Model Projects. The Pounamu Tool Project lets a user specify their own modelling language by defining a meta-model made up of entities and associations. The Pounamu Tool Project enables a user to define and describe various visual and textual notational constructs with respect to the tool meta-model. For example, the user may define an "Actor" meta-model element, and then may specify a visual icon to create and view Actors in use case diagrams, and textual property sheets to be able to view and update each Actor's properties. Once a user has created a tool project, they use this tool specification to create various models based on the meta-model described by his tool. The Pounamu Model Projects manage model instances and view shapes created by users when using the modelling views of a tool.

We required the collaborative editing plug-in to provide synchronous and asynchronous collaborative editing support for any Pounamu-specified tool while needing no code changes to Pounamu itself. We also required that Pounamu tool users be able to decide if they needed collaborative editing functionality and choose to load the plug-in (or not) at run-time. For this to work, each application must have the same Pounamu tool project open, enabling them to share the same tool definition

(meta-models, views, shapes, property sheets, event handlers etc). Collaborative editing is then undertaken by sending modelling view editing messages between Pounamu desktop applications running on each user's PC. The editing messages are replayed by the target Pounamu application to effect a change made by another user. This can be done synchronously – changes sent as they are generated to other users, or asynchronously – changes are requested periodically and applied as a group to another user's modelling view.

We chose to adopt a peer-to-peer architecture for our collaboration components. This was based on findings from our previous research into collaborative editing in CASE tools [6, 7]. Pounamu being a desktop application presented an interesting challenge - integration of a desktop application with a web server in order to effectively use web services to support peer-to-peer transmission of editing events.

One of the major decisions taken during the design process was establishing how various Web Services were to be hosted. It is envisaged that in the future lightweight Web Serve components wil be available that will easily integrate with existing desktop applications and be able to host Web Services. Currently no such Web Sever exists thus making it necessary for us to integrate the Pounamu Meta tool application with a standard web server using RMI technology. Figure 3 describes the architecture of the system.

The main components of the system are the Pounamu Meta Tool, the Collaborative Editing Component, the Web Server and the RMI Registry. As User A edits modelling views, their Pounamu tool generates various editing events. These events are propagated to the collaborative editing component of User A's Pounamu tool using Pounamu's subscribe/notify event propagation mechanism (1). User A's collaborative editing component forwards the editing events to its peer(s) hosted on other users' PCs using remote web service calls (2), in this example to User B's web server hosted by User B's PC. This target Web Service looks up User B's collaborative editing component using a locally hosted RMI registry on User B's PC (3). It then forwards the message it receives, containing the editing events generated by user A's Pounamu tool, to User B's collaborative editing component (4). This in turn maps the editing events generated by User A into API calls for User B's Pounamu tool. These API calls result in corresponding modelling view updates being achieved in User B's Pounamu modelling tool, duplicating the effect of the edits made by User A on User B's modelling views (5). Asynchronous editing is supported by caching of edit events by the sender (User A) or receiver (User B) collaborative editing components.

An alternative client-server based architecture is presented in Figure 4. In this architecture, we have a central Web Server and RMI Registry used by a number of Pounamu applications. This has the advantage of not requiring locally hosted web servers and RMI servers on each user's PC, enabling use of a high-end server machine to host servers, and enabling caching of edits by the centralised web service. The disadvantage is a loss of robustness – if a shared server fails all collaborative editing is halted until it is restarted.

5. Example Usage

In this section we describe an example usage of our Pounamu plug-in collaborative editing component. To understand how the collaborative editing component works consider the following scenario. Two colleagues John and Mark are working on an ERD (Entity Relationship Diagram) based model project created in Pounamu from geographically disparate locations. By enabling synchronous collaborative editing, they can collaborate in real time. If they desire, they can choose to use asynchronous editing and can merge sets of diagram changes made by others at their own convenience.

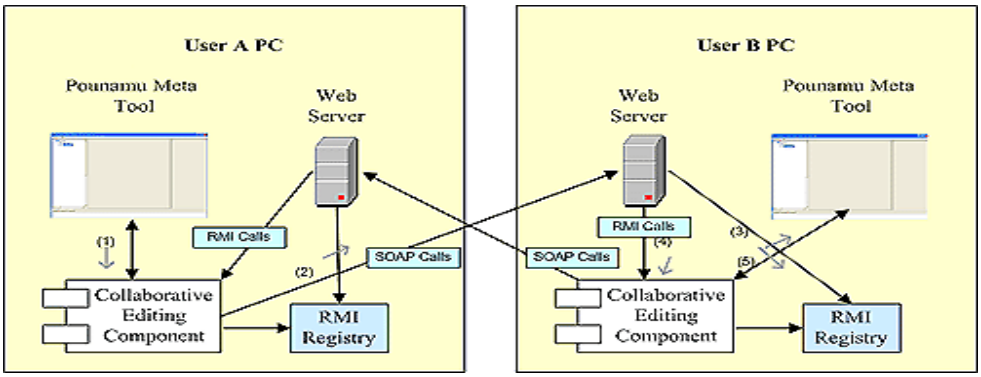

Figure 3. Overview of our web services-based collaborative editing system architecture.

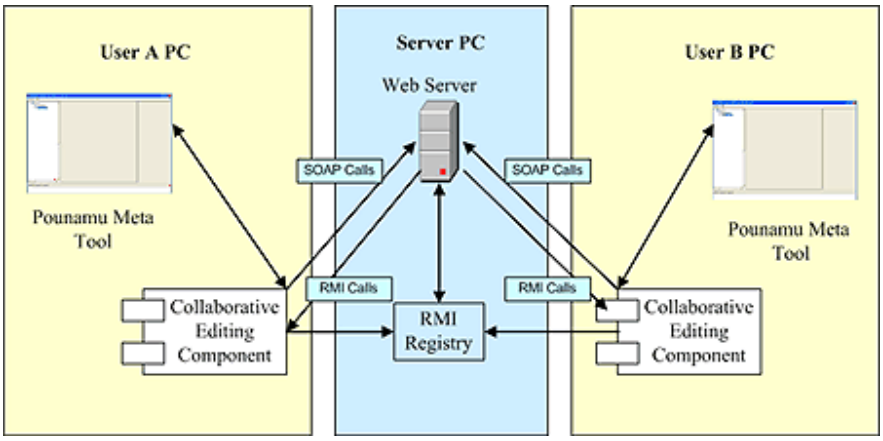

Figure 4. A shared web server enterprise scenario system architecture

Consider the scenario of John and Mark deciding to collaborate in synchronous mode. Figure 6 shows a sample scenario – all entity (square) or relation (oval) shapes surrounded by a box-highlight have been remotely added. The arrow annotation between screen dumps shows the direction in which messages are being passed.

To start with John is presented with a configuration dialogue box (Figure 5) that enables him to choose the users he wishes to collaborate with and the mode of collaboration (synchronous or asynchronous).

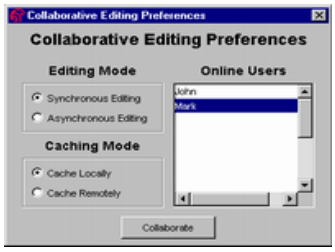

Figure 5 Configuration Dialogue Box

To begin with John adds an entity "book" in his Pounamu ER modeling view Figure 6 (1). The view edit events generated by Pounamu are sent to John's collaborative editing component and in turn forwarded to Mark's collaborative editing component via the web services infrastructure described in the previous section. Mark's collaborative editing tool generates API calls to Mark's Pounamu tool which results in ER diagram view updates that make it appear to Mark as if the new book entity has been immediately added in his tool.

Now Mark decides to add an attribute author to this new entity (2). As soon as Mark does so the attribute is added in John's environment by the same mechanism as the creation of the entity in Mark's Pounamu tool – editing events are sent back to John's environment via web services and immediately actioned to create the new book attribute. Now finally John decides to add an attribute publisher and create an association between the Book entity and attribute Author (3). These changes are immediately reflected in Marks environment as John completes each of these diagram edits.

Asynchronous editing can be enabled at any time and edits are cached either at the sender end or receiver end. John and Mark can choose when to have changes made by the other merged with their ER diagram updates. Changes to an ER diagram component in another view result in the meta-model instance generating view updates, which are propagated to other users in the same way as described above. The entire contents of a modeling view can be sent to new users on request, being recreated by their Pounamu tool so they can join the collaborative editing session. Network failure will default view editing to asynchronous mode with local edit event caching. Any changes made are then sent to other users when communication is re-established. Concurrent synchronous edits by two users on the same modeling view item are currently inter-leaved by a simple locking mechanism. A multiple edit multi-user undo/redo is also supported. Occurrences of conflicts are resolved by presenting the user with a dialogue box and prompting the user to take appropriate action.

John and Mark are able to collaborate quite easily in the Pounamu software environment using this plug-in synchronous editing support. Simple synchronous work co-ordination is achieved by highlighting shapes temporarily using pre-existing Pounamu diagram facilities. Further co-ordination can be achieved via the help of a chat application, audio link or other third-party communication support tool. The collaborative component added to Pounamu tools enables both Mark and John to work simultaneously from anywhere in the world as long as they have access to the Internet. Since we are using Web Services there are no problems associated with firewalls and other network security features to contend with.

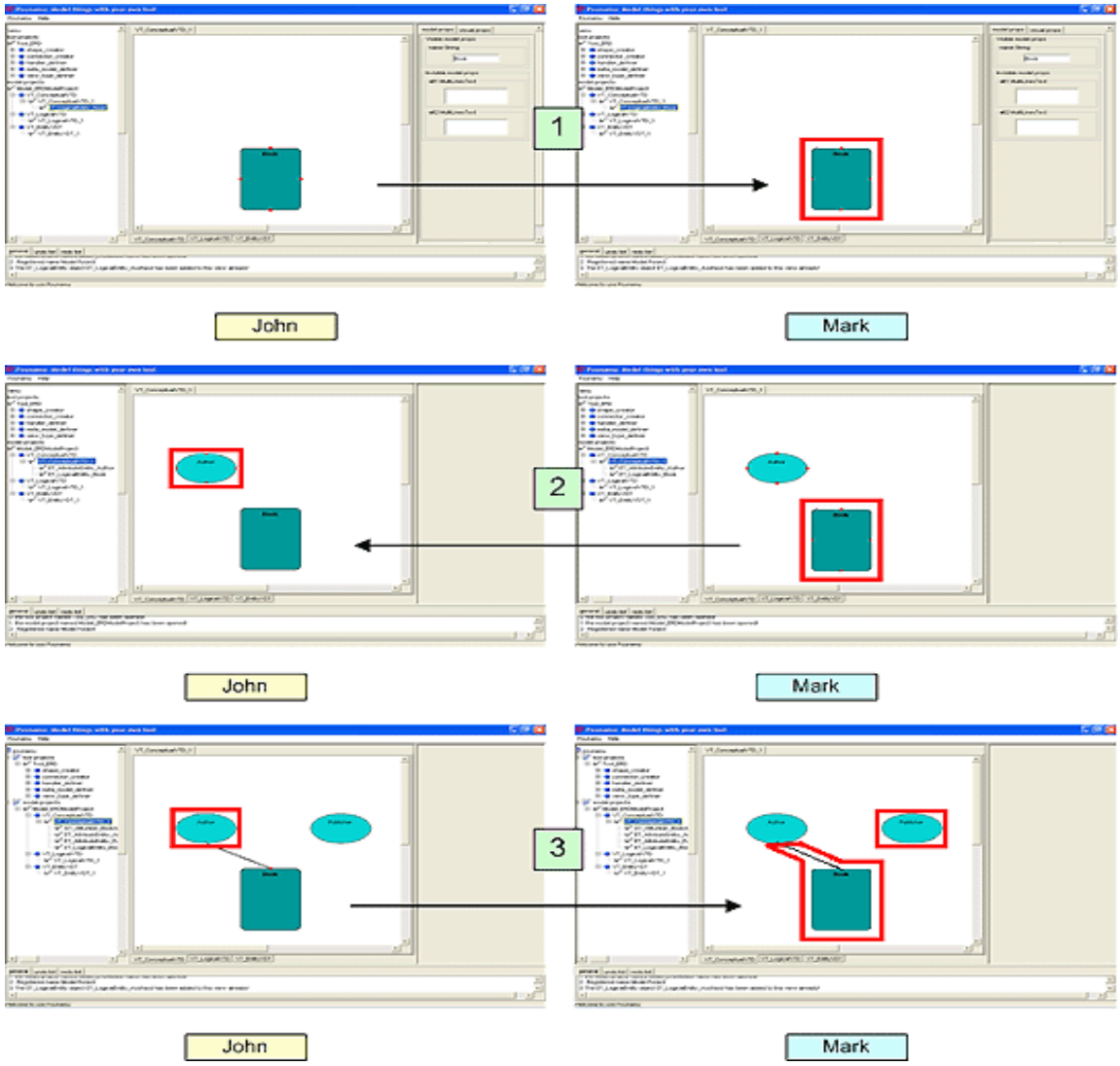

Figure 6. An example of synchronous collaborative editing in Pounamu

6. Design and Implementation

One of our main design objectives was to make minimum or ideally no changes to existing Pounamu code. The collaborative editing component has been implemented as a plug-in [7]. No code associated with any part of the collaborative editing component has been embedded in the original Pounamu code, except for some functionality relating to turning this component on. In our design, each Pounamu application has to have a corresponding collaborative editing component to enable it provide collaborative editing. A Pounamu application will generate a number of modeling view editing event types. Our collaborative editing component subscribes to all of these. Editing events are propagated to the collaborative editing component to be processed immediately after they are generated by Pounamu. Once processed the collaborative editing component calls the appropriate remote web services on the collaborator's side.

Two main features of the Pounamu meta-CASE tool enabled us to design our collaborative editing component as a plug-in. We ensured during Pounamu's core development that it provides extensive subscribe/notify event handling infrastructure. Pounamu also has a number of API's which can be used to execute all of the required editing commands and data structure queries on the tool remotely. In addition, we ensured that all editing event serialisation and deserialisation, remote component communication and co-ordination control for collaborative editing is done by the collaborative editing component.

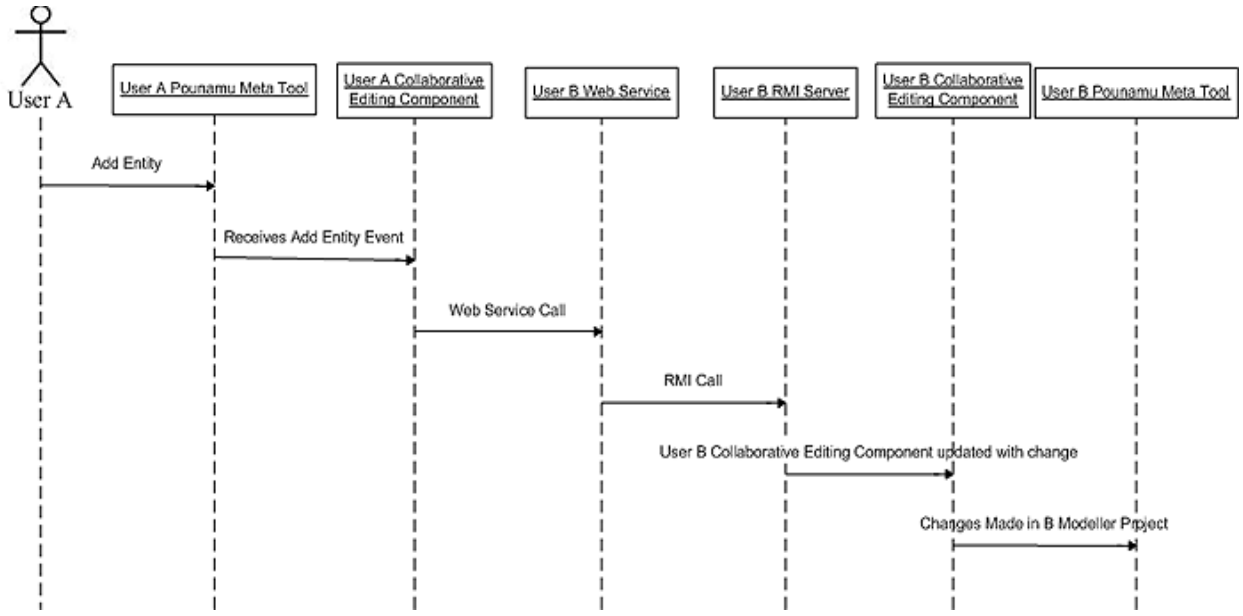

Figure 7. Sequence diagram for a typical interaction in a Collaborative Editing situation

Using web services technology to build this collaborative editing component design turned out to be quite straightforward. We developed a set of "mirror" web service-based APIs for each Pounamu API feature. These allow a remote Pounamu environment to query the state of another Pounamu tool and to run editing "Commands" – objects that encapsulate modelling view change-of-state instructions, remotely. In addition, Pounamu's subscribe/notify event handling was supported by equivalent web services-based infrastructure using SOAP messages to encapsulate editing event information. We developed a set of web service components that call remote Pounmau APIs via RMI, thus providing a translation between web service-based messages and RMI API calls into Pounamu environments. We found this approach necessary in order to host the web service APIs in a separate web server to our Pounamu application.

The sequence diagram in Figure 7 describes the interaction between various objects in a collaborative editing situation. Typically we have two users collaborating, User A and User B. If User A issues an add entity command then User A's Pounamu meta tool will generate an event for the entity received which will propagate to its corresponding Collaborative Editing object. The event will encompass various details of the entity that has been added – type, location, initial property values and so on. User A's Collaborative Editing component then calls the Collaborative Editing Web Services hosted by User B. This informs User B's Pounamu tool of the changes made to the shared modelling view by User A. User B's Collaborative Editing Web Service relays the message to User B's Collaborative Editing component via an RMI call that an entity has been added. The entity is subsequently added to UserB tool.

The Pounamu Meta Tool was developed using the Java platform. This made it straightforward to develop the Collaborative Editing Component using Java RMI and Java web services implementation technologies. Our collaborative editing system includes Web Services and Remote Method Invocation (RMI), and we used JAX-RPC (Java API for XML-based Remote Procedure calls) for building the Web services on the Java platform. JAX-RPC provides a remote procedure call based on the SOAP protocol.

7. Discussion

We have used our collaborative editing plug-in components for Pounamu to provide a basic set of synchronous and asynchronous editing features for any Pounamu-specified visual modeling tool. Examples this has been used on to date include an ER diagramming tool, UML tool (including use case, class, sequence and deployment diagrams), and semantic modeling tool. Synchronous editing currently provides a basic locking mechanism to inter-leave multiple user concurrent edits. Asynchronous editing allows caching of edits made by other users on source or target PCs and selective merging of changes.

Our approach provides similar capabilities to previous work we have done with component-based collaborative editing components for software tools [7, 6]. It has capabilities similar to a number of other researcher's approaches for their frameworks and environments [2, 15, 17]. However, our Pounamu plug-in is more flexible in that it provides synchronous and asynchronous editing for

any tool that can be defined with the Pounamu meta-tool – a potentially enormous range of applications. Many framework-based collaborative editing support approaches have been developed [2, 15, 17], but nearly all hard-code the collaborative editing support into applications built with the framework. This complicates the framework and incurs overheads for both the tool and user interaction, even if the collaborative editing support is not desired or not always required by users [7]. The use of wrapper Web Services around existing Pounamu API's demonstrates an approach that can be easily integrated into any tool including existing commercial tools. Other alternative approaches would be to use an application sharing framework e.g. MS NetMeeting™ for synchronous editing, and version control tool for asynchronous editing support. Disadvantages are that moving between the two can be difficult, diagram-based models are difficult to effectively version and merge, and knowledge of the application events, as used in our approach, allows semantic as well as syntactic consistency management support to be provided [2, 3, 7]. Web services technology has been used for other collaborative work systems, but in the main for supporting workflow-based integration [8]. Our components currently only provide quite rudimentary group awareness facilities compared to other frameworks [15, 17].

Key future research includes improving configurability of the collaborative editing plug-in and supporting integrated semantic as well as syntactic checking during asynchronous merging operations. We would like to develop some group awareness plug-ins that complement the collaborative editing support currently offered. We are interested in using web services dynamic discovery and integration technologies to locate and integrate diverse collaborative work services. We would also like to apply this support to our web-based diagramming component for Pounamu and our zoomable views support, both recently added to the Pounamu meta-tool. We are interested in seeing whether disparate software tools can share these web service-based components i.e. collaborative work between Pounamu and non-Pounamu tools.

8. Summary

We have built a prototype collaborative editing plug-in component for Pounamu, a meta-tool for building visual diagramming applications, using web services technology. This plug-in allows distributed users to collaboratively edit diagrams synchronously or to asynchronously edit diagrams and merge changes. The plug-in uses web services to encode editing events and diagram content in XML, exchanging these between users environments. A plug-in component architecture using web services technology has proved to be feasible for building such a collaborative work infrastructure, with a number of promising future extensions possible.

References

1. Concurrent Versioning System (CVS), http://www.cvshome.org/.
2. Dewan, P. and Choudhary, R. Coupling the User-Interfaces of a Multiuser Program, *ACM Transactions on Computer Human Interaction*, **2** (1), 1995, 1-39.
3. Ellis, C.A., Gibbs, S.J. and Rein, G.L. Groupware – some issues and experiences. Communications of the ACM, 34(1):38–58, January 1991.
4. Gisolfi, D. Is Web services the reincarnation of CORBA?, Web services architect, Part 3, IBM Developer works.
5. Glass, G., The Web services (r)evolution Applying Web services to applications, IBM Developer Works, Nov 2003.
6. Grundy, J.C., Hosking, J.G., Mugridge, W.B., Apperley, M.D. A decentralised architecture for software process modelling and enactment, IEEE Internet Computing, Vol. 2, No. 5, IEEE CS Press, Sept/Oct, 1998, 53-62.
7. Grundy, J.C. and Hosking, J.G. Engineering Component-based, User-configurable Collaborative Editing Systems, Software – Practice & Exp, vol. 32, Wiley, 983-1013, 2002.
8. Kafeza, E., Chiu, D. and Cheung, S.C. Alert-Driven Process Integration in a Web Services Environment, In Proceedings of the 1st International Conference on Web Services, Las Vegas, USA, June 23-26 2003.
9. Kaiser, G.E. Dossick, S.E., Jiang, W., Yang, J.J., Ye, S.X. WWW-Based Collaboration Environments with Distributed Tool Services, World Wide Web, vol. 1, no. 1, 1998, 3-25.
10. Newcomer, E. Understanding Web Services: XML, WSDL, SOAP, and UDDI, Addison-Wesley, June 2002.
11. Pacull, F., Sandoz, A., Schiper, A., Duplex: A Distributed Collaborative Editing Environment in Large Scale. Proc. 1994 ACM Conference on CSCW, October 1994.
12. Pendergast, M.O. and Vogel, D. Design and implementation of a PC/LAN-based multi-user text editor, In Proceedings of IFIP WG 8.4 Conf. on Multi-User Interfaces and Applications, North-Holland, September 1990, 195–206.
13. Posner, I.R. and Backer, R.M. How people write together, Proceedings of the 25th Hawaii International Conference on System Sciences, Vol. IV (January), 1992, 127-138.
14. Reiss, S.P. Connecting Tools Using Message Passing in the Field Environment, IEEE Software, **7** (7), 1990, 57-66.
15. Roseman, M. and Greenberg, S. 1996. Building Real Time Groupware with GroupKit, A Groupware Toolkit, ACM Transactions on Computer-Human Interaction, 3 1, 1-37.
16. Schooler, E.M. Conferencing and Collaborative Computing, Multimedia Systems, vol. 4, 1996, 210-225.
17. Shuckman, C., Kirchner, L., Schummer, J. and Haake, J.M. 1996. Designing object-oriented synchronous groupware with COAST, *Proceedings of the ACM Conference on Computer Supported Cooperative Work*, ACM Press, November 1996, pp. 21-29.
18. Stoeckle, H., Grundy, J.C. and Hosking, J.G. Approaches to Supporting Software Visual Notation Exchange, In Proc. of the 2003 IEEE Conference on Human-Centric Computing, Auckland, New Zealand, October 2003, IEEE CS Press.
19. Younas, M. and Iqbal, R. Developing Collaborative Editing Applications using Web Services , Proc. 5th Int. Workshop on Collaborative Editing, Helsinki, Finland, Sept 15, 2003.