Freie Universität Berlin

Diplomarbeit zur Erlangung des akademischen Grades

Diplom-Informatiker

Iterative, prototype-driven development of a whiteboard feature

for the Distributed Party Programming tool Saros

Michael Jurke
Matrikelnummer: 3981778
jurke@inf.fu-berlin.de

Eingereicht bei:
Prof. Dr. Lutz Prechelt

Betreuer:
Karl Beecher
Stephan Salinger

Berlin, 28. December 2010

Table of Contents

1 Introduction and basic information

With this work, a whiteboard feature for Saros, an Eclipse plug-in for Distributed Pair Programming (DPP) is to be provided. Thus in this chapter, first there is a brief introduction to agile software development whereof Pair Programming (PP) is one of its methods, latter about PP in a distributed context, current research and existing tools, to finally present Saros that defines the context of how and where to plug the whiteboard in.

1.1 Agile software development

The term agile software development was introduced in 2001 in the Agile Manifesto. This manifest describes 12 principles, among they state that their highest priority is to “satisfy the customer through early and continuous delivery” and further methods and techniques for iterative and incremental development.

“Agile”, a replacement for “lightweight” methods, is chosen in contrast to the existing “heavyweight” waterfall model that is seen by agile advocates as heavily regulated and regimented. The main idea is that small self-organized teams work closely together with the customer, incrementally adding new functionality at a constant pace. Specifications may be omitted, requirements are seen as subject to change from the very beginning and tasks are broken into small increments with minimal planning to maintain the development process adaptable and reduce over-all risks.

The team size is kept small as face-to-face communication and teamwork is essential to agile methods. Most agile methods use a routine and formal daily meeting so that in a brief session the members report to each other what they did the previous day, what they are working on and which problems they are forcing.

If a team works in different locations, they should maintain daily contact through video conferencing, voice, email or they might even meet to have a session with a too for distributed programming.

There are a lot of well-known implementations of agile methods, examples are Extreme Programming, Feature Driven Development and Test Driven Development.

1.2 Pair programming

Pair programming is an approach out of the agile software development, especially a central part of extreme programming. It implies that two developers attend their work on the computer together. Traditionally, one of them, called the "driver", will use the mouse and keyboard to produce code, the other, the "observer", keeps a watch on the produced results, reconsidering to find errors, solve problems or giving advises. The roles are meant to be changed if suitable, i. e. if one programmer is more capable for certain issues.

This method is supposed to improve programming discipline, to produce better code and to distribute knowledge beyond different programmers. However, there are also critical voices [XPRef03].

1.3 Distributed pair programming

If seriously following the principles in the Agile Manifest

(4) Business people and developers must work together daily throughout the project.

(5) The most efficient and effective method of conveying information to and within a development team is face-to-face conversation.

(12) At regular intervals, the team reflects on how

to become more effective, then tunes and adjusts
its behavior accordingly.

Making pair programming distributed in the agile context seems to be a paradox. However, already the first international conference for Extreme Programming presented a notable publication related to distributed collaboration with the Team Streams system [XPExam01] that provided support for asynchronous interaction.

Over the years, several tools and plug-ins were developed for DPP, see beyond.

1.3.1 Efficiency

In 2004 there was an article “Support for Distributed Pair Programming in the Transparent Video Facetop” [DPPFacetop04] that describes an experiment that analyzes if distributed pair programming is as efficient as collocated pair programming. The breadboard construction consisted of a user interface named “Facetop” that enabled the programmers to see their peers in the background on the screen, furthermore they were using voice and text chat and a program to share applications. The authors arrive to the conclusion that the programmers worked in this construction as effective as if they had sat next to each other.

1.3.2 Related tools

When looking around one might be surprised at the amount of tools for DPP. I will name a short selection, some of these tools are referenced later in this work as they had influence on some design decisions.

On the one hand, there are screen sharing applications like VNC (Virtual Network Computing) or Microsoft NetMeeting that replicate the screen and take control of remote user's mouse and keyboard. They are less interesting for this work as we desire a whiteboard that enables concurrent drawing.

A very handy tool is SubEthaEdit for Mac OS X, originally designed for coding [SubEt], it's collaboration feature also got widely used apart from this for teaching or just composing text altogether.

Next, there is a plug-in for Visual Studio called Wave-net [WAVE_VS], a client/server architecture using TLS encryption for communication and implementing the Google Wave Protocol. It allows multiple drivers and it integrates version control systems. It is a commercial product and not Open Source.

Respective Java, especially for Eclipse, just a short search revealed six plug-ins, XecliP, Sangam, PEP (Pair Eclipse Programming), XPairtise, ECF Cola and of course Saros. However, Sangam is not publicly available, Xeclip and PEP are quite basic and the latest releases were in 2007.

XPairtise is a project from the German University FernUniversität in Hagen, developed in 2006. It is client/server based in the way that a server has to be started separately. It's a pure pair programming tool in the way that multiple drivers are not possible. It offers automatic synchronization if a navigator's file got changed accidentally and offers integration of version control systems. Among it's features there is also a distributed whiteboard I will describe a little further in section 2.2. Unfortunately, also XPairtise had its last release in 2008 only.

1.3.2.1 ECF

The Eclipse Communication Framework, is a “framework for building distributed servers, applications, and tools” [ECF]. It is modularized and standard based (OSGi 4.2 Remote Service standard, former RFC 119). Like this, the ECF, or parts of it can be used to build other plug-ins, tools or even full RCP applications.

For this work, features that must be mention are real-time shared editing, multi-protocol presence and instant messenger integration with multiple accounts and contact list.

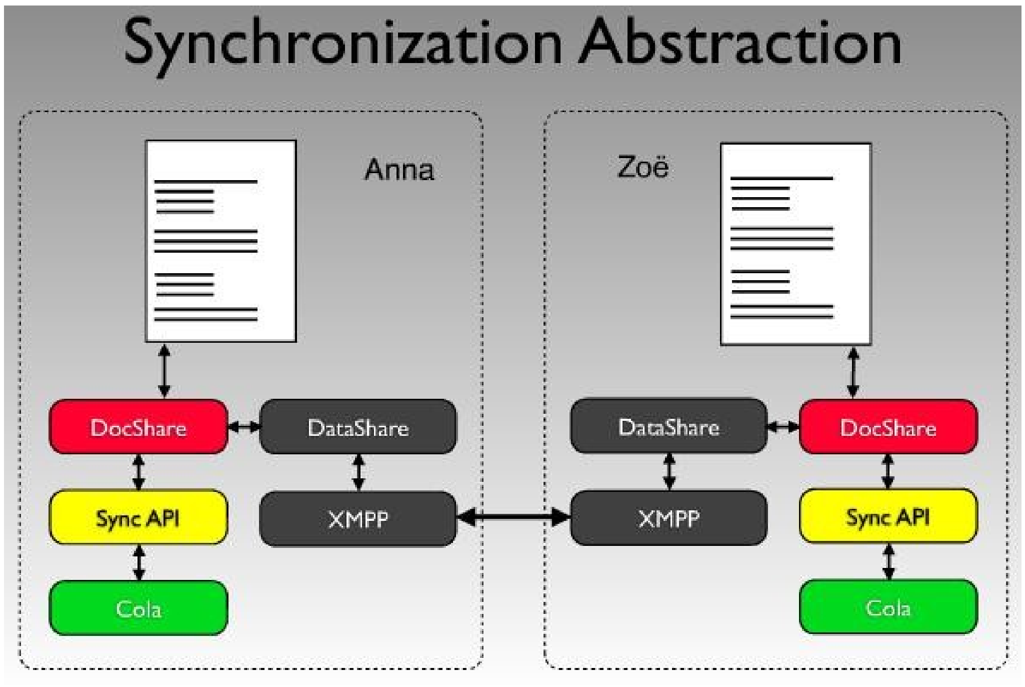


Figure 1 - ECF DocShare and Sync API[1]

Shared editing in ECF is separated in three parts (see above). Cola, from Collaboration, was the first step, a project for ECF that started out of the Google Summer of Code 2006. It had its first major release in Eclipse Ganymede in 2008. Cola is the synchronizer for shared text, following the Jupiter approach ([RTSE]) described in [Nichols et al. 1995], furthermore the DocShare API was developed that contains additional infrastructure like the connection manager. Finally, in 2009 the Sync API was introduced that provides an abstraction layer for a concrete synchronization implementation like Cola, exposed as an OSGi service attempting to spit DocShare in core and user interface [ECF30]. Further to mention is the DataShare API that is used by the DocShare API to transmit messages in abstraction to XMPP.

For the future much more implementations of the Sync API are planned. So for EMF/GMF editors (real-time graphical modelling), or even SubEthaEdit might be adopted. [ECON09]

Practically, apart from additional features, it enables the user to have a roster view integrated in Eclipse. Right click on a editor allows him to share a file with any peer in his contact list. Thus sharing is document, not project based.

1 http://www.eclipsecon.org/2009/sessions?id=429

There was the agreement that a whiteboard feature would also be a useful extension for ECF, too. Furthermore, at the latter state of my work, the team decided to aim at port Saros to ECF in the near future. As a result, developing the whiteboard I tried to keep some parts separated from Saros if this did not require additional effort.

1.4 Saros

As mentioned, Saros is a DPP plug-in for the IDE Eclipse, and actively developed by the department of Software Engineering, computer science faculty of the FU Berlin. Its beginning was in 2006 with a diploma thesis of Riad Djemili [Dje06] and is under continuous enhancement.

Saros does not need a specialized server, instead it uses the Extensible Messaging and Presence Protocol (XMPP) so a conventional XMPP server like Openfire can be used to contact and invite peers to establish a session.

Like in XPairtise, usually whole projects are shared in a session, however, there exists experimental support to share a project partially.

It offers different views for instant messaging like a roster- and session view, but also a chat. Furthermore, in 2009 it was enhanced by multiple driver support that means several users can write concurrently in the same file [Zi09].

We will address technical details again in section 2.3.2.

1.5 Prototyping

Fred Brook's book “The Mythical Man-Month” [Bro75], famous for Brook's law "adding manpower to a late software project makes it later", also highly encourages prototyping. The provocative assumption is “Plan to throw one away; you will, anyhow". Following Brook, there is no question whether you should build the first version as prototype, you will do it anyway – the only question is if you throw it away learning from your experiences or if you just deliver it to the customer (or commit it to trunk). This work tries to make the prototyping process explicitly.

Prototypes in the scope of software engineering are incomplete versions of an application that are developed to quickly gain knowledge or user feedback. A use case may be to create only the user interface and to let the client evaluate whether it looks like he wants it to look and whether it contains the functionality he would like to have.

Literature ([Pom96], [Bisch92]) sometimes distinguishes between three different types: explorative, experimental and evolutional prototyping.

Explorative prototyping aims to find out more about the requirements of the product to be developed. It is typically applied in during requirement gathering and is a useful solution if the programmers are no domain experts or if it is unclear what the customer really wants and needs. The prototype is used as a mean of communication.

Experimental prototyping in contrast is used to find out or to show that suggested solutions are feasible. Although it can be used during analysis, it's main focus is the design phase. It is to prove the suitability of specifications, architecture models or general ideas for system components or external libraries to use. Another point may be to find out whether the chosen architecture is sufficiently flexible concerning extensibility.

Finally, **evolutional prototyping** actually is a kind of incremental software development. In most cases the first step here is to develop the user interface with minimal additional effort to make it presentable while it stays incomplete and not really usable. This shows how the system may look like to quickly achieve user feedback and fill open questions respective to its requirements. With each following iteration functionality is added until the final product. Note that in the case of evolutional prototyping there is no clear separation between prototype and product, so that architectural and respective implementation details developers have work with more care.

However, other sources like [Cri91] and the English wikipedia[2] mention other types, especially Throwaway (or Rapid) prototyping only regarding the difference to the evolutional approach whether the prototypes are discarded or incrementally enhanced up to the final program. The goal of a throwaway prototype will be a system specification (explorative prototyping) while the evolutionary aims to become the final system.

1.6 Prototyping - in the scope of this work

The whole dimension of requirements for distributed whiteboarding in the context of distributed programming is rather ambiguous. While freehand drawing belongs to the must-have features, other tools like rectangles and other shapes, features like selection, awareness and command stack with re- and undo might still have to be verified.

1.6.1 Requirement analysis and verification

It is thinkable to provide simple throwaway prototypes, user interfaces only - that are to be reviewed by customers. But it would be difficult to experience a distributed whiteboard if the important difference to a common paint program – distribution – did not work. Another point is that the whiteboard should be evaluated in the context of DPP so it has to be provided and tested in conjunction with Saros, meaning it has to be integrated in Saros anyway that might not be a trivial task. On the other hand, developing prototypes that incorporate all these features, the overhead will be much too big.

This leads to the conclusion that throwaway prototyping is the not appropriate way to verify feature requirement and assumptions relative to the user interface. The better way will be to follow the evolutionary approach, incrementally extending an incomplete version.

2 http://en.wikipedia.org/wiki/Software_prototyping

1.6.2 Evaluation of design decisions and system components

Disparately, I am not a domain expert for whiteboarding, graphic libraries or distributed editing as well as project design. Nevertheless I had to come to some principal decision how to start to solve the task and which libraries to use without knowing them. This in turn is a typical case in which Brook's assumption is applied – the first version will be a prototype anyway, very likely to be thrown away. Respective the above distinction this can be seen as experimental prototyping to evaluate if external components are feasible.

1.6.3 Hybrid prototyping approach

[Over91] compares evolutionary and throwaway (in the paper referenced as revolutionary) prototyping. Several published case stories are reviewed trying to estimate the impact on Human-Computer Interface design (HCI) and software development. In a conclusion a hybrid approach which balances the two concerns is encouraged. Throwaway try-outs in the beginning to slowly solidify a evolutionary prototype. Behind this there is the observation that after finishing considerable parts of a project even small changes may require large effort but if these changes are applied in the beginning they will be cheap as there is nothing to redo.

Although the mentioned work mainly focuses on customer communication for changes in the requirements, especially the last statement can also apply in a technical manner. In particular the Saros team members alter frequently, as a consequence there may not only be a larger effort for changes in a latter state of a module but besides a new student might have to become familiar with the whole subject-matter. Thus changes in a latter state should be minimized and ideally a design should help to keep them local.

As a conclusion, in a first phase, this work will try to evaluate design decision, system components and external libraries in an experimental prototyping. In a second phase an evolutional prototype is to be developed exercising more architectural care.

After presenting the basic background information and general concepts, in the next chapter you find concrete planing considerations corresponding to this thesis's subject area.

2 Distributed whiteboarding

About the necessity of a whiteboard for DPP, several studies are available. First of all, another result of [DPPFacetop04] was actually that programmers would like to have a whiteboard to support their work while working distributed.

Moreover, in the first steps of Saros [Dje06], in 2006, Riad Djemili accomplished a survey about different aspects of DPP with 44 attendees. One part targeted towards the requirements of an optimal tool for DPP, one question explicitly to a whiteboard feature (“You have a whiteboard editor which allows you to draw geometric shapes and diagrams which your partner can see”). Only 11% answered with “unnecessary”, but 45% said it would be “nice to have” while 27% saw the whiteboard as an “important” feature and 9% even as “essential”.

Also mentioned is an interesting aspect from [SWN*03]: They discovered a higher formalism regarding exposure with documents when comparing DPP to collocated pair programming. In general, from PP frequently arise spontaneous documents that help to improve idea transferring or facilitate explanations. Just written on a paper or traditional whiteboard they are subject to get lost while DPP requires a digital form that easily would allow saving, localizing and recovering.

Ultimately, R. Djemili already proposes a possible solution for Saros: A graphical editor that additionally stores the the graphical model as text (i.e. XML) where the replication in the distributed context is applied.

The evaluation in the context of [Kro08] added up to regard the whiteboard as a “nice-to-have” feature due to the result that it was only used once as an additional explanation medium (beyond 49 sessions). On the other hand it was employed exclusively in three further sessions – explicitly in the end of the work to create sketches and diagrams for documentation and clarification.

2.1 The task examined

Developing a whiteboard is not really a well-defined task. However, before starting this work it was narrowed down a bit. The most important feature was specified to be freehand drawing and a diagram editor was not wanted. This can be justified by the requirement of informality like described in [LaMy95]: ”What designers need are computerized tools that allow them to sketch rough design ideas quickly“.

In contrast to the above proposal to have a text based model by [Dje06], there was the idea to develop a pixel based whiteboard with eraser, using “pixel swapping” to synchronize the picture for everybody. The advantage is the possibility of a pixel eraser tool like known form other paint programs.

In this context I studied the SWT paint example[3]. It implements double buffered painting and rubberbanding (or rather feedback) for line and shape drawing. In the background it creates figures that are immediately painted to the graphical contexts of the background, of the current image or of the rubberband figures. The example is neither small nor simple as it implements every single figure and manual double buffered painting to merge the image with the rubberband without flicker. This example does not have a eraser tool, instead you could paint existing parts over with the background color.

Among all distributed whiteboards mentioned in this work there is only one pixel based – the whiteboard of XPairtise. Researching further I could not find any other example nor literature about pixel-based distributed painting. The problems are quite obvious: on start synchronization, binary data would have to be transmitted if anybody joins later in a session as well as on inconsistencies. To synchronize concurrent edits, the command information would have to be saved to apply any kind of handling. That leads to the question why not using this data directly to draw the image instead of storing respective regions of the paint surface that quickly might degenerate and need

[3] part of SWT example projects, CVS: dev.eclipse.org; module: org.eclipse.swt.examples.paint

an enormous amount of memory.

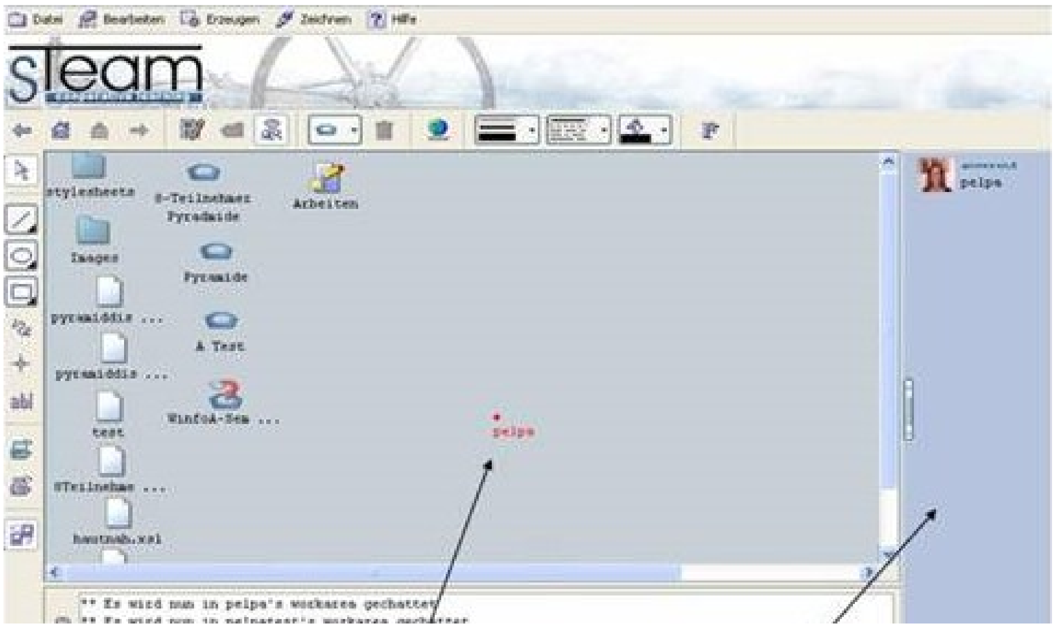
XPairtise actually uses a linear data structure as the underlying model, thus the whiteboard only appears to be pixel based while actually, for instance, eraser points are placed on top of other drawing.

Respective to this question I also got feedback from other Saros team members. The main conclusion was that they prefer to draw shapes because it is difficult to draw lines with the mouse. Furthermore it would be great to rearranged or extend drawn sketches. But if based on pixels only the whiteboard would be unaware of semantics like shapes and text. It would be impossible to select, rearrange and compose on a node level.

2.2 State of the art

This chapter will present a short selection of open source distributed whiteboards or graphical editors with distributional support. The palette of solutions contains simple distributed sketch extensions up to integrated distributed learning environments.

open-sTeam is a open source collaborative system from the University Paderborn to establish and maintain virtual knowledge spaces.



** Es wird nun in pelpa's workarea gechattet

Figure 1 - sTeam Whiteboard

(http://www.open-steam.org/Dokumente/docs/steamhandbuecher/Handbuch_Whiteboard.htm)

The start of the first sTeam Whiteboard was formed in line with a diploma thesis 2001 [Bü01]. It is a Java application with a Swing interface and communicates with the server by COAL (Client Object Access Layer), a protocol developed by sTeam to scan and execute functions of the server. Events of the sTeam server are transmitted immediately so it is a synchronous client.

As can be seen in Figure 1 it integrates components of the sTeam knowledge platform that can be rearranged and extended by drawings in real-time with others in a session.

This whiteboard knows a certain amount of objects, among lines, arrows, pictures and text, it also offers selection and modification, in continuation in this chapter referred as standard shape tools. A special tool to mention is the *Telepointer* – a pointer visible by all users as helper in presentations and discussions.

Although the above whiteboard seems to be a quite complete solution, in 2006 a complete new version was created, based on Eclipse to facilitate the plug-in concept, called Medi@rena. It is an Eclipse RCP application using the Graphical Modeling Project ([GMP]) but does not differ much in functionality (nesting is allowed and connections are added).

Note that both these tools do not have freehand drawing support and are rather not thought to sketch but to develop, share and present learning materials and diagrams.

Already referred several times in this work, the DPP plug-in for Eclipse **XPairtise**, offers a whiteboard feature. It does not support any standard shape tools but free hand drawing with an eraser and export to JPG. The underlying linear points model is are synchronized respective their order.

There are several XMPP based instant messengers that also offer a whiteboard feature. Namely **Coccinella**[4], **Psi**[5] and **Gajim**[6]. They all support standard shape features, however, they do not allow nesting of elements. Their underlying concurrency control structure is related and will be referred again in section 3.1.

Later in this work I will exemplify to use SVG as base for the whiteboard model representation so there is to mention Inkscape[7].

Inkscape is open source vector graphics editing application and aims to be fully compliant to SVG (Scalable Vector Graphics) standard, including XML and CSS. Even more it offers extensions to the standard like connections and more.
In previous versions it used to have distributed whiteboard feature, formerly known as Inkboard, allowing the user to collaboratively create and manipulate an SVG. It worked synchronous and the users where encouraged to use a local server to minimize delay. However, it was removed because it was cause of a couple of bugs and finally lost support so this step saved “a lot of 'user support' time” [Inkwb09].

Another university project to mention is the **MOBILIS platform** [MOBILIS] based on Android and Java using XMPP. It is a platform for mobile collaborative social applications and also has a collaborative drawing feature. The underlying synchronization framework will be later referred in section 3.3.

4 thecoccinella.org
5 psi-im.org
6 www.gajim.org
7 http://inkscape.org

Of course there are plenty of commercial products, too. One of the most famous might be Microsoft NetMeeting, but there is a huge amount more like the whiteboard feature of Adobe Connect Pro, Teamboard, Visualtek Rendezvous, SMART whiteboard, or Edraw Mindmap just to mention a few.

2.3 Preconditions

Incorporating an extension to a plug-in based on another platform involves a couple of preconditions and restrictions.

2.3.1 Eclipse

It originated from IBM VisualAge – a family of computer integrated development environments (IDE) – and was made open source in 2001 [EclFAQ]. Now it is a free software development environment with an extendible plug-in system. The plug-in mechanic can be used for instance by Saros to extend the IDE by the collaboration feature but also offers to create standalone Rich Client Platforms (RCP) that only use central parts of the framework, with any other plug-in needed, of course.

When extending Eclipse, first of all concerning graphical contribution, extender will have to use Java, SWT and JFace. The Standard Widget Toolkit (SWT) is graphics library for Java that in contrast to Swing and equally to AWT accesses native operating system GUI libraries. Thus the appearance is similar to programs written in operating system specific APIs.

JFACE is defined as “a UI toolkit that provides helper classes for developing UI features that can be tedious to implement” [JFACE], implementing common UI programming tasks from Viewers up to dialogues and wizards.

2.3.2 Saros architecture and concurrency control

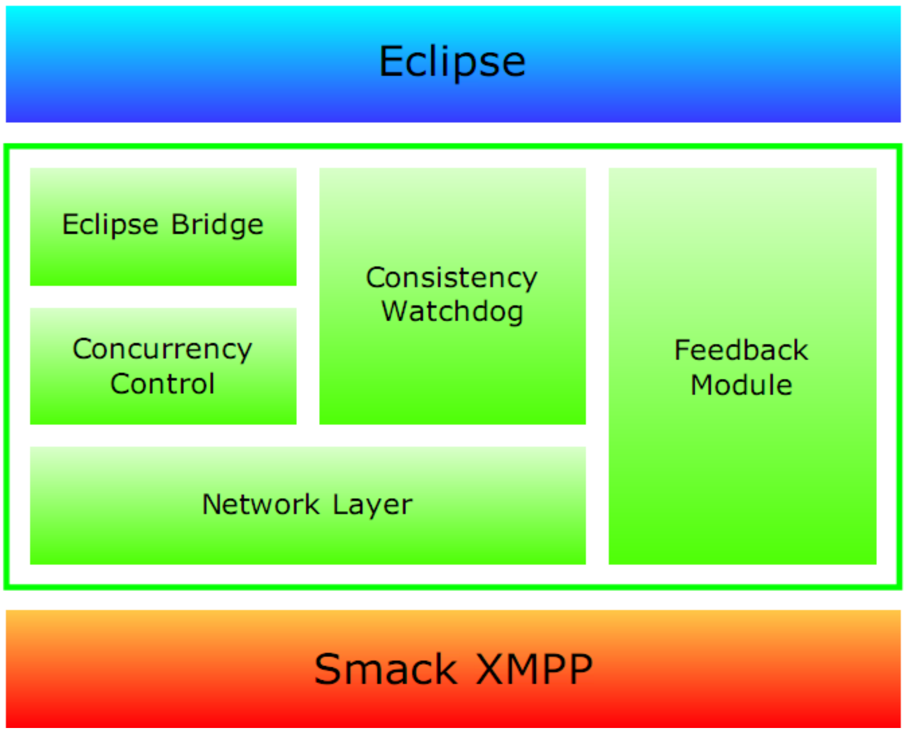


Figure 2 - High-level architecture of Saros [Saros10]

In the above figure you find the abstract high-level architecture of Saros to enable DPP inside of Eclipse based on the Smack API (status November 2010).

Considering a distributed whiteboard, it could require to be integrated in all these architectural parts. As alternative, it had to be provided a custom implementation respectively. Please note, a bridge level does not make sense for the whiteboard as it does not change default Eclipse editors but has to provide a custom view.

On the one hand, Saros offers some abstraction to the network layer for in-session communication in form of the activities infrastructure, also implementing the lower level of concurrency control, namely the Jupiter architecture with the operational transformation algorithm on top of the network layer (see beyond). Activities offer user identification, SWT context switching, sequencing and payload. They are based on XStream, a Java library to serialize Java class objects to XML.

On the other hand, invitation and start synchronization are partly on top, partly hard-coded inside the network layer, furthermore there are different approaches for different parts. While project synchronization is an integral part of the invitation, user list synchronization has its own handler on the invitee side.

The Consistency Watchdog – built on top of the activities infrastructure – exchanges regularly checksum activities used to detect inconsistencies. Originally they were meant to be repaired automatically and resynchronized with the host, however, currently (November 2010) this feature is deactivated. Without forcing the repair, after detecting issues the user is informed by a button, on click the synchronization is performed.

Plugging the whiteboard in Saros is no trivial task, especially due to some inconveniences. These issues are addressed more precisely in chapter 6.3 with conclusions respectively.

2.3.2.1 Jupiter

To achieve convergence in collaborative editors there are two possible communication topologies: replicated and centralized.

Following the centralized approach, edits are first sent to a central server that maintains the shared object, applied and transformed there and sent back. In such a way, they are not immediately visible which interrupts local work flow and decreases user experience, undesirable for Saros.

Having a replicated infrastructure, every client maintains his own version of the shared object where he applies local edits. Afterward they are sent to the peers where they have to be integrated in the context of concurrent operations to maintain consistency.

The Jupiter approach describes a hybrid of a replicated and centralized architecture. Although having a local copy, after applying, operations are sent to a central server where they are sequenced and integrated in a first step before being forwarded to the peers that do a final transformation and apply it to their version. Thus the complexity is reduced from a n-way communication to a 2-way messaging from peer to central server.

Let me at this stage point out that the term Jupiter is used in a quite confusing manner in the context of Saros. Sometimes it was referred as Jupiter-algorithm, sometimes as Jupiter-architecture and sometimes as Jupiter as thus.

The idea (so the term) was developed as part of the Jupiter Collaboration System, developed in 1995 ([Nichols et al. 1995]) as a solution for the dOPT puzzle problem with a high latency and low bandwidth (dOPT is short for distributed Operational Transformation, see beyond). Hence, the Jupiter architecture could correspond to the above mentioned hybrid approach while the Jupiter algorithm describes the modified dOPT algorithm that ensures convergence in this topology (that is actually not implemented in Saros).

Note, that other protocols mentioned in this work use a similar hybrid topology. CEFX ([Ger07]) and SXE ([XEP-0284]) do not even refer the above work and one may assume that they did not know it. ECF Cola ([RTSE], references the hybrid architecture as an approach that could also apply in their context to reduce the complexity (exactly the way Saros uses it), while Google Wave accredits the above work as starting point ([GWave]).

In my case, the statement Saros implements the Jupiter algorithm ([Saros10]) let to confusion. Also very vague statements like the description that Jupiter is not for linear data structures, but for interactions of users with the GUI, that can be represented by arrays of operations. Even if this was true in the case of Saros (it is not), it does not justify this term.

The problem could only be clarified by reading the related works ([Nichols et al. 1995], [Rie08] and [Zi09]) and exchanging e-mails with Sebastian Ziller.

For some team members, Jupiter became a synonym for “something very complicated to ensure convergence in a distributed text editing environment that currently does not work (status November 2010)”. To avoid this I would advise to stop to use this term generously. When talking about communication topologies, it could be referred as (the Jupiter approach to use) *a hybrid out of a centralized and replicated architecture*. Respective the server, it could be described as a *specialized server component* (at the host site for a first operational transformation and global sequencing). And the algorithm should be referred as a *modified GOTO-algorithm* that uses this server component to reduce complexity.

2.3.2.2 Operational transformation

In a replicated architecture, already collaborative editing of plain text files may lead to inconsistencies without concurrency control, this observation leads to the concept of Operational Transformation (OT). The idea was pioneered by in 1989 by C. Ellis and S. Gibbs (dOPT, [Ellis89]) and capabilities were continuously extended including locking, group undo, awareness up to application sharing and more [OTFAQ], especially to mention for this work is tree-structured editing within the treeOPT algorithm [Ignat02] that with some modifications could apply to XML, too. We will return to this topic in context of the concurrency control implementation of the whiteboard model and in 3.3.

Saros text concurrency control is based on GOTO, an optimized version of the GOT (Generic Operational Transformation) algorithm, developed in [Sun et al. 1996] as a consequence of dOPT not ensuring convergence in every case. Although GOTO actually allows peer-to-peer communication, the usage of the Jupiter architecture reduces the complexity a lot ([Nichols et al. 1995] and [RTSE]).

The hybrid topology in combination with GOTO as concurrency control mechanism was introduced to Saros in 2008, for more information I refer to the related works of Oliver Rieger and Sebastian Ziller ([Rie08] and [Zi09]).

2.3.2.3 XMPP and Smack API

Jabber was announced in 1999 as an open technology for instant messaging and presence, together with the release of the first version of jabberd server. In 2001 the Jabber Software Foundation (later XMPP Software Foundation, XSF, see below) was formed to coordinate the growing number of projects using Jabber technologies. During the formalization process of the IETF, in 2004 it was subsequently named Extensible Messaging and Presence Protocol (XMPP). [XMPPHistory]

Among the strength of XMPP belongs decentralization, the usage of open standards, security via SASL and TSL but also due to the possibility to isolate XMPP servers from public and flexible extensibility by extension protocols. Last but not least, XMPP already has a long history and is widely used in open source projects as well as commercial products and has and had the backing of large companies such as Sun Microsystems or Google. For more details and reasons why it was used in Saros I refer to the work [Dje06].

In practice, Saros uses the open source XMPP Java library Smack.
As a shortcoming of XMPP there is to mention the inefficient in-band binary data transfer: binary data has to be base64 encoded and sent as a XML document. To address this several extension protocols were proposed to transfer binary data out-of-band using XMPP like the Jingle File Transfer XMPP Extension Protocol [XEP-0234]. Unfortunately the Smack Jingle file transfer implementation was not feasible enough for Saros, thus that in line with the work of H. Staib [Sta10] a custom Smack extension implementation was developed that enables SOCKS5 bytestreams. For more reasons and details of the change and bytestreams in general I refer to the mentioned work, however, this topic will be addressed again in chapter 9.1 because part of my work was a small network refactoring to incorporate SOCKS5 in the Saros network layer.

3 Whiteboarding over XMPP

Towards collaborative whiteboarding over XMPP (and XML over XMPP accordingly, see below) there have been different proposals, developments and researches in the recent years.

3.1 XMPP/Jabber standardization efforts

As it was mentioned before, XMPP offers flexible extensibility by extension protocols (XMPP Extension Protocol, XEP), the standard process for proposing and providing XEPs is given by the very first, XEP-0001, that apart from the processes describes different types and states. After a proposal has been submitted without being accepted, it is first available in extension inbox[8] as a “ProtoXEP”.

Already in the very beginning of Jabber there had been several proposals for a whiteboard feature extension; the very first already in 2001 as XEP-0010 that quickly got obsolete as part of the Special Interest Groups ([XEP-0002]) cleanup ([XEP-0019]) because of lack of activity in these groups. The next one, in form of [XEP-0113], was sent by developers of the Tkabber whiteboard plug-in but did not get updated or extended subsequently and it also got obsolete. Other submissions like [WBX06], [WBX07] or [SXDE06] even did not get beyond the ProtoXEP status.

In the beginning of my work there was one more ProtoXEP but under active development, with some attendance of Peter Saint-Andre (executive director of the XSF), namely Shared XML Editing (SXE). In June 2010 it became approved and is now available as an experimental XEP [XEP-0284]. It originates in the whiteboard of Coccinella, a Tcl/Tk based instant messenger, and was continuously improved, even implemented by the other instant messengers Gajim and Psi but their versions are obsolete at current state.

3.2 Scalable Vector Graphics (SVG)

Having an XML based network protocol together with the idea of a textual presentation of a graphical model like proposed in chapter 2, one directly gets to the Scalable Vector Graphics standard (SVG).

SVG, first published in 2001, was the result of the W3C SVG Workgroup that in fact developed the new XML-based language on base of six different submissions from different companies [SVGOrig]. Thus it is an open standard for two dimensional vector graphics. Apart from a standard way to represent graphical data, it also specifies dynamic behavior, for instance for interactive applications or animations, for the whiteboard feature of less interest.

8 http://xmpp.org/extensions/inbox/

Note that all the before mentioned instant messengers use a subset of SVG as underlying model representation.

3.3 Collaborative XML editing

Considering the network and concurrency control, the usage of SVG leads to the generalization to just have to synchronize an XML document in a collaborative editing session. Advantageously, there is different current scientific research available.

Concurrency control for collaborative editing of hierarchical tree structures is not the central part of this work, but to develop a collaborative whiteboard feature for an existing DPP infrastructure. Thus, for a very detailed introduction about the current state of art I refer to the work of Dirk Hering “Entwicklung eines Dienstes für Real-time Collaborative Editing für die Mobilis-Plattform” ([Her09]). Here, I only give a short overview about current developments related later in this work.

Most notable to mention are the works to adopt the treeOPT algorithm for XML ([Ign06]) and the development of the Collaborative Editing Framework for XML (CEFX, [Ger07]) by Ansgar Gerlicher that subsequently was extended to use a XMPP Multi-User Chatroom (MUC) by Michael Voigt ([Voi09]) and adopted in the Mobilis platform[9] by Dirk Hering ([Her09]).

Both solutions try to cover the whole area of application of collaborative editing of tree-structured documents and thus result in complex algorithms and data structures. While the treeOPT adoptions are based on operational transformation, CEFX uses a different approach to maintain consistency, later on presented in section 6.5.2.
Both concepts are supposed to work peer-to-peer, while CEFX in practice uses a Jupiter-like hybrid architecture as does SXE.

In contrast, SXE tries to simplify the problem to encourage implementations

[9] http://www.inf.tu-dresden.de/index.php?node_id=578&refer_id=573&ID=112&ln=de and http://mobilisplatform.sourceforge.net/

and fast acceptance as a standard. Thus it covers only a subset of operations (i. e. without join or split operations) and introduces mechanics to simplify the problem, for instance it uses a primary/secondary weight mechanic to specify the child order and in the following no synchronization is necessary in this field.

During development, I became aware that SXE actually can work peer-to-peer with only a small modification that even simplified the implementation. On the contrary, there are other aspects connected to the move operation (moving a sub-tree in the hierarchy) where it does not fulfill all assertions. Convergence is not assured in all cases and due to concurrent move operations it may happen that the resulting XML document is not well-formed anymore.

We will come back to this theme when treating the implementation of the concurrency control data structure for the whiteboard. Now for a start, after presenting basic considerations, we come to the prototyping process of the the graphical interface of the whiteboard.

4 First iteration – SVG rendering with the Batik library

The SVG standard is rather complex, even in the way that there does not exist any library with 100% compliance. In August 2010 as a result of a test with the official SVG Test Suite[10], the most compliant tool was Opera, 95.26%, with version 10.61 [SVG2010].

Consequently, it would be completely misplaced to try to achieve SVG compliance by manually implementing all of its features in the scope of this work and it suggests itself to use an external library.

For Java there does exist two major SVG libraries, Batik ([Batik]) and SVG Salamander[11], both are based on Swing thus they would have to be integrated

10 http://www.w3.org/Graphics/SVG/WG/wiki/Test_Suite_Overview
11 https://svgsalamander.dev.java.net/

in SWT using AWT-SWT bridge[12].

Batik, originally supported by IBM and then donated to the Apache Software Foundation[13], formerly was the most complete SVG 1.1 implementation. Now it is on the second place after the Opera browser ([SVG2010]). Thus, its SVG support is much better than of SVG Salamander and was chosen to be used in the first whiteboard prototype.

A special point to notice is the animation support. Although the whiteboard is not supposed to support animations, real-time SVG editing actually requires animations. It would be a performance killer if a library had to recreate the DOM tree and the rendering tree anew on every change.

Investigating a little bit more I found applications that use Batik in the context of Eclipse, namely the already mentioned GMP, the SVG Eclipse Plugin[14], that renders SVGs only, and an Eclipse plug-in to modify SVG documents called SVGCompost[15]. Furthermore in 2008 on the SVGOpen there was a project using SWT and Batik [SVG08]. Finally, the holongate project even tried to develop a Java2D for SWT library. Unfortunately the last release was 2006. These references motivated me to follow the proposed solution using batik.

Besides I found an Eclipse corner article about Swing integration that mentioned problems that could occur and their solutions ([SwingSWT]).

12 http://help.eclipse.org/helios/index.jsp?topic=/org.eclipse.platform.doc.isv/reference/api/org/eclipse/swt/awt/SWT_AWT.html
13 www.apache.org
14 http://sourceforge.net/projects/svgplugin/
15 http://developer.berlios.de/projects/svgcompost

4.1 Batik and SWT integration

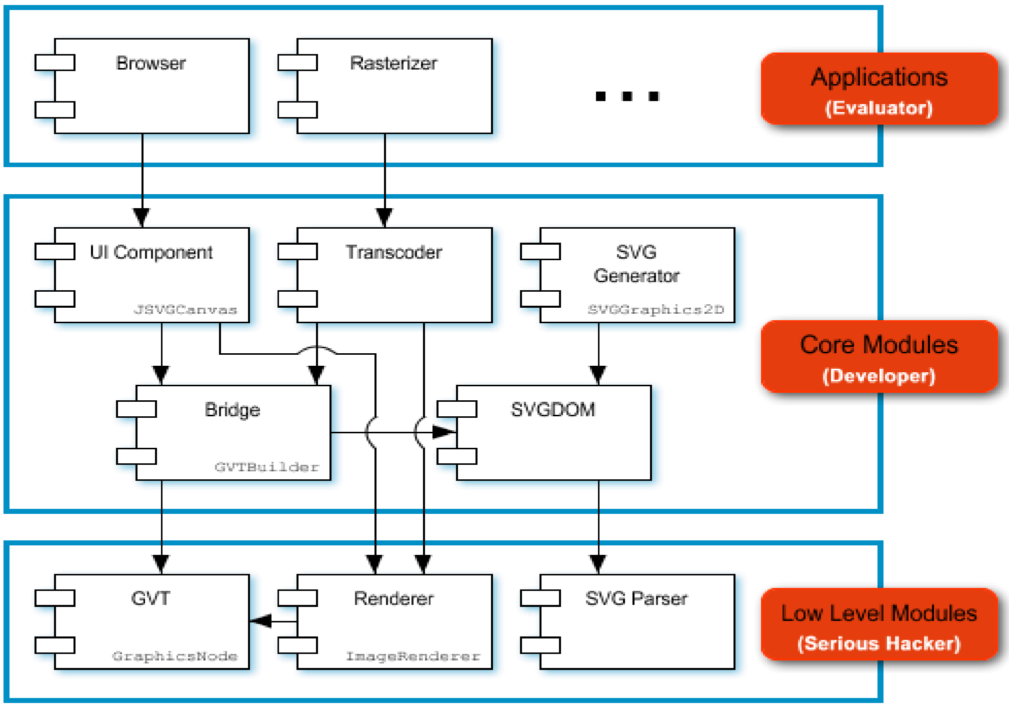


Figure 3 - batik modules
http://xmlgraphics.apache.org/batik/using/architecture.html

For the beginning I only needed the UIComponent part[16] that offers a method to add an SVG document that will be rendered immediately. This was plugged in an embedded SWT Composite using an SWT-AWT Frame and all together initialized in an Eclipse view. Thus, the user could open a view that visualized a static SVG document.

The JSVGCanvas already comes along with a set of so-called “interactors”. For instance by pressing CTRL plus the left mouse button you have a panning tool, Shift plus the left mouse button can be used for zoom. In this manner, if the document is bigger than the provided area of the Eclipse view a user could use panning or zoom to see the whole area.

Of course, this is not really intuitive and customers would expect a scrollbar that was quickly added, leading to the first problem with AWT-SWT integration. Indeed Batik contains also a JSVGScrollPane but the scrollbars would have the Swing library's look and feel, in most cases an easily solvable problem by setting the system look and feel to the Swing UIManager. However, this might

[16] in particular the JSVGCanvas

not be enough on all platforms ([SwingSWT]) and hence would require a platform specific fork in the implementation. Furthermore even without being integrated in SWT the Batik scroll pane did not work like expected. In the following, I added scrollbars to the SWT Composite with listeners to update the contained frame and vice versa. A scrollbar change in batik can be caused by quite a few components, this can be the UI Component, the tree-builder (after creating the tree for the first time), the tree-renderer (using interactors) but also the update manager (dynamic changes), thus the solution was not so easily distinguished.

A problem occurred with respect to resizing the canvas and the scrollbars. This was the first time that the double-threaded nature of a AWT-SWT embedding caused problems with non-trivial solutions, for instance to avoid deadlocks it is heavily discouraged to run any mutual calls (AWT to SWT and the other way around) in a synchronized manner but if running asynchronously it might happen that for instance a call gets a ahead of a previous one or gets ignored completely. In this case this only caused a minor issue leaving the embedded frame a little bit bigger or smaller than the outer SWT composite. As the problem also occurred using the simplest embedded frame – and as it only was a matter of one or two pixels after quick subsequent resizing – it was regarded as acceptable for the moment.

4.2 Dynamic manipulation - Batik UpdateManager

In the next step there was to add a drawing feature. This was quickly done by adding a mouse listener to the JSVGCanvas. On click, a polyline element was added to the SVG document and dragging extended it by additional points.

But to enable Batik to immediately paint those newly created lines, they have to be added to the document in a specialized thread, namely the runnable queue of the UpdateManager.

At this point we should have a closer look to the SVG polyline element. The tag only has one points-attribute containing the list of points. Thus, when adding a point to a polyline in the context of a DOM document the whole attribute has to be set again forcing batik to recalculate the whole rendering rectangle of this polyline that led to bad performance. Note that due to freehand drawing a polyline quickly may have several hundred points.

4.3 Feedback layer

When drawing a line it is still thinkable to directly add it to the document while drawing. However, at the latest when talking about shapes a feedback becomes a basic requirement to visualize to the user how his shape will look like or at least how big it is.

Of course, Batik does not have a build-in feedback support but it could be easily added by a last child element of the root. This element could use its own namespace and in this vein provide abstraction to clients. Likewise concerning freehand drawing the user might want to interrupt his action after having begun and thus it should be added to the feedback layer primary, too.

This approach also solved parts of the above mentioned performance problems with extending polylines. While drawing, the tool adds simple lines to the feedback layer, one after the other, only on completion a polyline is created and added to the “real” document.

4.4 Tools

I already implemented the mouse listener that is able to draw lines. Now the manager should know different tools. In an instant the view was extended by a palette composite on the left (without icons though) where the user could chose between freehand or rectangle drawing.

4.5 Saros integration

To quickly achieve a working prototype, I skipped parts like start synchronization and decided to just make use of Saros' activities infrastructure. Newly created DOM elements are sent to peers without a possibility to manipulate existing elements.

Activities implement the double dispatch pattern in Java having the

AbstractActivityReceiver implementing the visitor pattern. However, there somehow existed a second “consumer” structure and its purpose was not completely clear to me, especially as it caused some overhead without knowing how to use it. It was meant to give the possibility to intercept an activity and I was told that it all makes sense.

It seemed that the expected way to add a new activity was just to do it like other activities do without really understanding everything. I was not alone with this feeling. A team member of the Software project that extended the Saros user interface before, admitted in fairness that they did not understand anything, mentioning “just copy and paste and wow, it works”. Other team members that were about to finish their thesis when I begun, neither could explain it.

4.5.1 Cleaning up Saros' double dispatch implementation

Only at the end of my work I found out what actually had happened: the whole doubled consumer part with converters to the receiver implementation was unused and more than 80% of the whole abstract infrastructure could simply be deleted leaving a clear implementation of double dispatch in Java.

Personally, I still see this pattern as discouraging. It will always force students to change the central business infrastructure by extending the visitor and cause them to use copy and paste maybe with, maybe without understanding. But the worst thing is that it interconnects all parts in Saros making proper modularization impossible. It might be a nice solution for an enclosed part like session management not meant to be touched by everybody but for other modules I see a filter architecture as a preferable solution like can be seen in Smack packet extensions.

4.5.2 W3C DOM element serialization

Back to the whiteboard, I added a new activity containing the mentioned DOM element and the serializable complement (ActivityDataObjects) that converted it to a string.
ActivityDataObjects in Saros are serialized to XML using XStream and

respective annotations, after all, it seems to be indicated to send the DOM element directly over the XML stream. Unfortunately XStream has no basic support for this as it is thought to stream plain Java classes. Of course, it would be possible to write an own converter for W3C DOM elements but due to the lack of mixed mode, comments and processing instructions in XStream this can be ambiguous and an answer from the mailing list convinced me that actually an encoded string will be the better solution[17].

Finally the first whiteboard prototype based on the Batik library was ready, allowing to open a whiteboard view to draw pencil lines and rectangles.

4.6 Problems

During implementation after and after I got aware of non-trivial problems due to the AWT-SWT integration.

4.6.1 Flicker

A minor issue was flickering in two cases. In the very beginning the whole composite flickered during scrolling and sometimes during drawing. A deeper look to the mentioned article ([SwingSWT]) provided a solution by setting the system property “sun.awt.noerasebackground” to true. The other flicker problem only occurred once in a while during drawing and was bounded by the dirty region (bounding rectangle) of currently drawn element. This was especially annoying while I still used a polyline from the start to enable freehand drawing – the dirty region of the polyline quickly covered main parts of the drawing area (for instance draw a line from up-left to down-right). The usage of the feedback layer reduced the flicker to the last two points. Other team members confirmed that they could live with this.

A look in the paint process with the debugger revealed more about the cause for the flicker. In general, Batik and SWT as well as Swing actually use double buffered drawing that is exactly meant to avoid flicker. However, before redrawing the dirty region, batik cleans the background buffer that for unknown reasons forced a repaint of this rectangle (supposedly by SWT) that thus

17 http://comments.gmane.org/gmane.comp.java.xstream.user/6664

becomes visible to the user for a short moment.

An answer from the batik user list where I provided the later mentioned sample implementation for the here presented issues confirmed that the above redrawing can only be cause by a broken connection between SWT and Swing[18].

4.6.2 Performance

In short words, the used constellation performed intolerably bad in the beginning, for instance, while drawing a line with a considerable length the line slowly started to appeared only ways after the mouse pointer. Frequently there were interruptions by background processes where the mouse movement was not tracked causing rough edges and choppy curves.

A solution was found in using OpenGL[19] as rendering engine (in opposite to DirectX[20]) using an argument for the Java Virtual Machine[21]. Drawing now proceeded fluently with only pretty short interruptions, thus it would not heavily effect user experience.

4.6.3 Background errors due to resizing

In combination with OpenGL another problem was solved. With DirectX, on small composite sizes suddenly the image was rendered wrong and doubled. Although the upper left part was correct, it reappeared in the lower part, sometimes overlaying, sometimes hiding the real background.

With OpenGL this problem was changed to a slight vertical "shaking" of the background image.
This problem might be connected to the amount of resize-events that are exchanged when embedding Swing in SWT as mentioned in the respective Eclipse corner article but I was not able to solve the problem by trying to reduce these events like described. A look in the debugger did not help as it stopped at the underlying rendering engine, OpenGL or DirectX respectively.

[18] http://comments.gmane.org/gmane.text.xml.batik.user/14967
[19] www.opengl.org
[20] http://en.wikipedia.org/wiki/Directx
[21] -Dsun.java2d.opengl=true

4.6.4 Rendering got stuck

Unfortunately only in the very end while testing the Saros integration I got aware that after some time, suddenly rendering stopped to work. I noticed that I can trigger this by subsequently resizing very fast for a while.

In the batik user mailing list[22], the author made the assumption that the blocking call (AWT EventQueue.invokeAndWait()) in the UpdateManager's method updateComplete() to repaint the JSVGCanvas might cause a deadlock. Researching more it revealed that actually the Batik UpdateManager still worked correctly and also paint-methods were executed properly. My assumption would be that somehow the underlying rendering engine got an error or an deadlock due to the double threaded behavior of SWT-AWT integration.

I repeatedly provided a very short sample implementation causing this problem to the batik user and developer mailing list as well as to the eclipse platform without gaining a solution. Finally I had to accept it as non-resolvable.

4.6.5 More

However, while searching for solutions I even encountered more issues. For example it seems that not all events are properly communicated from batik to Eclipse[23] inducing the people to use the JavaScript-Java interface LiveConnect to manually transmit the events to Eclipse.

4.7 Alternatives

In the following I had a closer look to other usages of batik in Eclipse like the mentioned GMP and SVGCompost project. Both actually abandon the Batik-Swing components. GMP uses a Java2D drawing adapter to enable the export to SVG facility. The SVG plug-in on the other hand uses Batik to render an SVG element to an rectangular image only that then can be manipulated using the Graphical Editing Framework ([GEF]).

[22] http://comments.gmane.org/gmane.text.xml.batik.user/14967
[23] http://www.mail-archive.com/batik-users@xmlgraphics.apache.org/msg08518.html

The last approach also could be used for the whiteboard. However, it does not implement to create any kind of element and instead of rectangular figures one of the basic whiteboard requirements is freehand drawing. Furthermore, it is to expect that fluent drawing by re-rendering an SVG element to an rectangle on every mouse move using batik is not given. Thus, one would have to implement the drawing feedback using GEF and in the following transform it to an SVG figure with the only advantage to be able to visualize and modify “unknown” elements from peers, too. But for the start there are no compatible peers to receive those elements from, thus it is a very complex solution without benefit.

4.8 Conclusion

The result of this first prototype was that in the given context in respect to the requirements of usability the employment of batik is unsuitable. It is little likely that the situation with SVG Salamander would be different. Thereby we also say goodbye to create a whiteboard with nearly full SVG rendering capabilities because of the necessity of an external library like presented in the introduction of this chapter.

The whole process took about five weeks, in my opinion a little bit too much for a throw-away prototype for a whiteboard. The reasons are simple. Although the implementation as such lasted not even two weeks it remained unclear whether problems would get out of control (else it could have become a an evolutionary prototype). Answers from support mailing lists encouraged to continue to force them and it took longer until it was clear that the integration of batik would always continue to cause trouble.

5 Second iteration – GEF The eclipse Graphical Editing Framework

When talking about graphically editing with Eclipse, it is inevitable to come across the Graphical Editing Framework ([GEF]). This framework actually was already considered in my inaugural presentation but some points caused me to foremost give a try to Batik.

First of all, it would go completely out of range to reach SVG compliance just using GEF. The situation changed now as the Batik prototype (chapter 4) revealed that external Swing libraries, the only available, are not feasible to be used for a whiteboard in the Eclipse environment.

Having a closer look to the instant messengers from section 2.2, namely Coccinella, Psi and Gajim, lays bare that non of them uses an external SVG rendering library (a graphical library though) and thus they only support a small subset of the SVG standard to provide compatibility and proper extensibility. Other reasons might have been that it could be quite time consuming to develop a vector graphics representation anew.
These points also influenced the decision to generally maintain SVG as model representation, or more accurately a subset of SVG to be extended as it comes.

Another point that argued against GEF was its missing freehand drawing support, certainly, its main purpose is to edit models and diagrams. On the other hand, there is the Eclipse Sketch project[24], an API for freehand drawing on Eclipse graphical editors based on GEF. Unfortunately having a look to the code, the freehand drawing feedback is painted on the graphical context object only, that means in a distributed environment where repaints can be caused by remote peers the feedback could be painted over while draing. Just like this, it was not given that GEF works without problems in a distributed environment.

As a conclusion, first of all there was to distinguish whether GEF is feasible for freehand collaborative drawing, or rather freehand drawing with concurrent

[24] http://www.eclipse.org/sketch/

edits.

Yet, there also are a lot of arguments to use GEF. It offers views and editors with quite some basic functionality, beyond drag and drop, zooming, printing, outline and overview pages, hierarchical tree layout or connections as well as basic editing features like copy, paste or undo and redo.
Last but not least GEF is well integrated in the Eclipse environment and won't have problems with SWT integration.

For a start, the next section will explain the GEF architecture and design patterns to be used in the implementation. It is left general and can be skipped if being proficient in the framework.

5.1 GEF background, architecture and design patterns

Its origin is a foundation for GUI builders and more (now the Eclipse Visual Editor Project[25]). It has a history of 10 years active development and is a stable and actively supported open source feature for eclipse ([GEF04]).

It is widely used in open source projects as well as commercial products and base for UML diagram editors, organization charts, state machines, GUI builders and much more.

GEF is part of the Eclipse Tools project and can be integrated as editor or view in the IDE or can be a stand alone editor using the Eclipse Rich-Client-Platform feature.

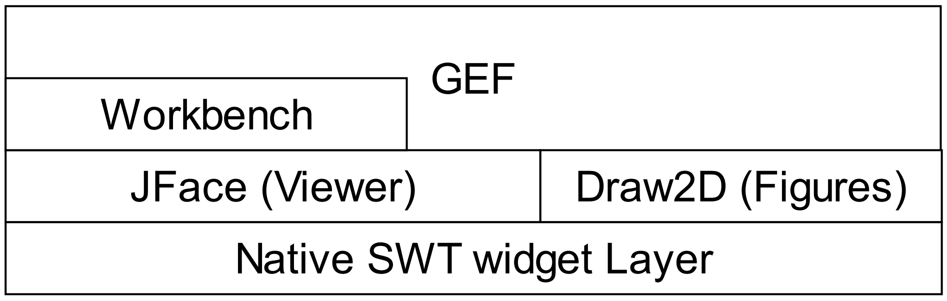


Figure 4 - GEF Eclipse integration

Draw2D is used for rendering, layout and scaling and GEF is the integration

25 http://www.eclipse.org/vep/

layer for Model-to-View mapping and the workbench integration.

As mentioned, it allows modifying models graphically so that the user can directly interact with the model by a figure. GEF handles the mouse and keyboard events, interprets the inputs, actions, menus and key bindings.

The goal is that one does not have to re-invent the wheel for generally necessary functions.

5.1.1 Draw2d

To understand the structure of GEF let's first present the underlying Draw2D API the counterpart for Java2D built on SWT. It is standalone lightweight GUI toolkit that shares design patterns with java Swing and was build to support GEF functions with optimized layout and double-buffered painting as well as clipping on a figure's bounds. Furthermore, Draw2D takes care of the state of the graphics object and freeing resources.

5.1.1.1 Structure and User interaction

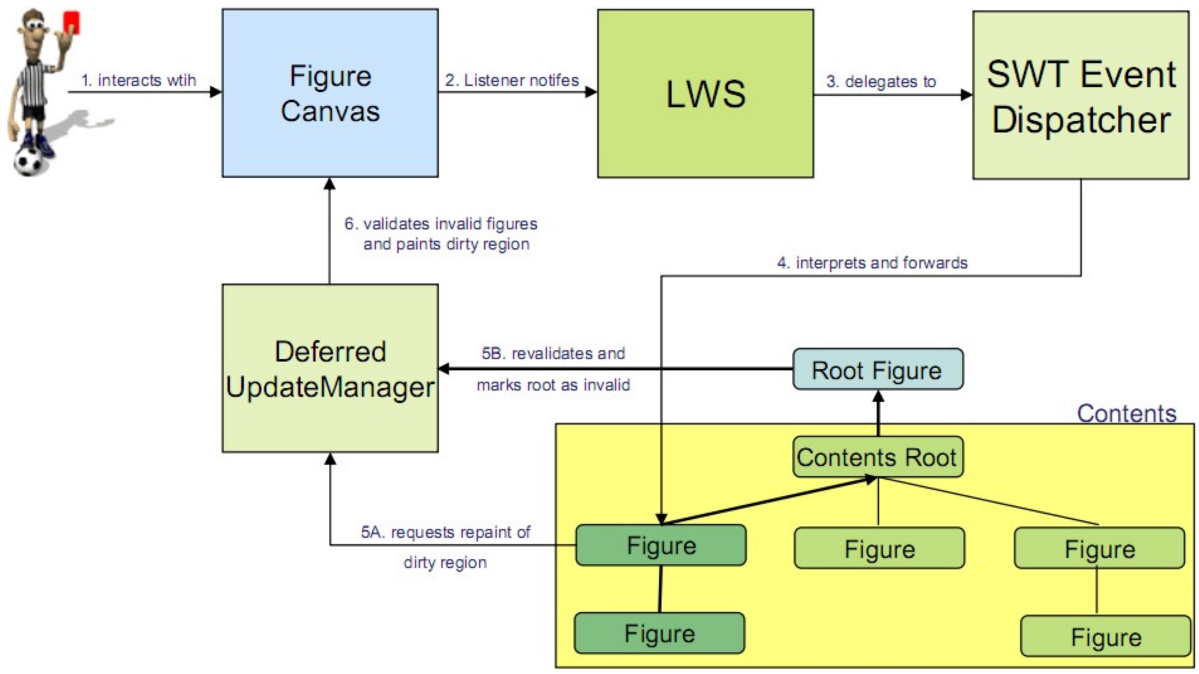


Figure 5 - Interaction scenario in Draw2d[26]

[26] source http://www.eclipse.org/gef/reference/GEF%20Tutorial%202005.ppt

The FigureCanvas as a subclass of an SWT canvas is the native layer and allows painting draw2d contents. The LightweightSystem (LWS) contains the EventDispatcher and UpdateManager. The first translates the SWT events and forwards the draw2d events and handles focus. The second schedules Layouts and calculates damage.

5.1.1.2 Figures

Figures are the basic lightweight component permitting colors, transparency, visibility, borders and much more. Figures can be composed without limits, for instance as Class-chart and can have any kind of shape although the clipping (respective child figures) will be rectangular only. For other purposes clients have to provide a custom clip-path themselves.

There are connection figures, anchors, labels supporting fonts and animations. The root is organized in layers so that selection or additional information can be visualized without conflicts.

5.1.1.3 Coordinate systems

By default, absolute coordinates are used but to use relative coordinates are offered, too. The coordinate system can be zoomed, scrolled and several utility functions make it easy to convert coordinates respectively other figures.

Unfortunately, Draw2D only offers flow and fixed layouts so that frequently there is to implement an own layout manager. For a whiteboard on the contrary, a simple coordinate layout will be enough.

5.1.2 GEF

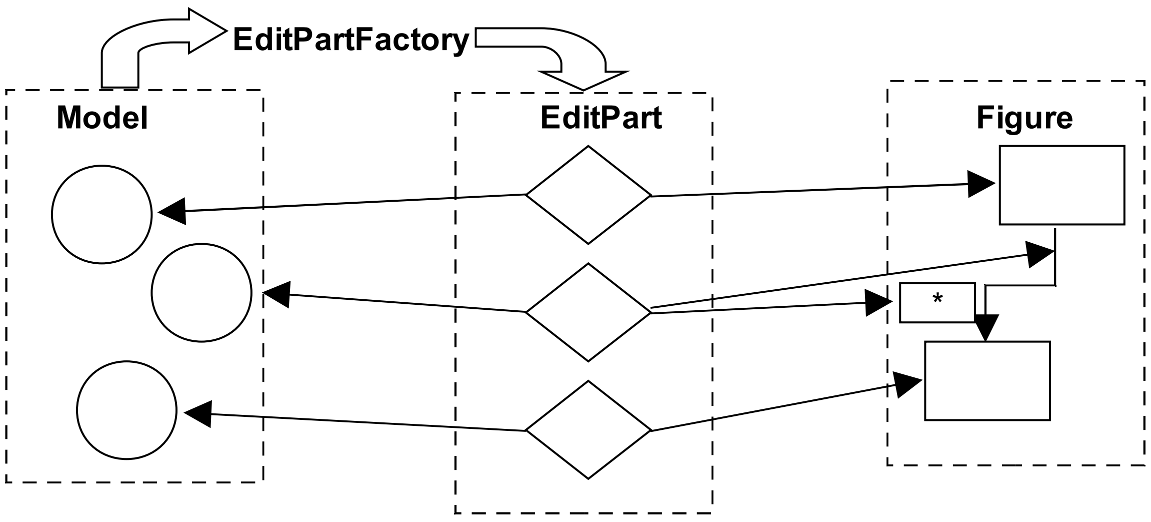


Figure 6 - GEF's Model-View-Controller structure

Like Swing, GEF uses the Model-View-Controller (MVC) architecture pattern and advertises a strict separation of the three layers. Thus offers flexible mappings between model and view.

5.1.2.1 Model

The model represents the persistent data layer and corresponds to the publisher in relation to the view implementing the observer design pattern. GEF is strict so the data container does not know other parts of the program. It promotes changes through listeners.

5.1.2.2 View

The visualization of the data is made in the view layer (remember, the GEF term is figure). It does not contain persistent data, equally is not supposed to know other parts of the program. It is programmed accordingly the composite pattern.

5.1.2.3 Controller

Finally, the controller tier directs the communication and uses the same named design pattern. The GEF term is EditPart that forwards changes in its refreshVisuals() method. Thus it contains a model and a figure.

The connection between model and edit part is made by EditPartFactory and on creation or change of a model edit parts are (re)created. They thereon create the figures.

5.1.2.4 Drawbacks

As mentioned, the MVC structure offers flexible mappings between model and view. This can be an advantage but as a drawback type checking during runtime is inevitable because type errors are only recognized then. Especially the frequent use of runtime type check (*instanceof*) has to be kept in mind while creating inheritance hierarchies – even the order of type checking might lead to different results (as a child always is instance of its parent), for instance this commonly applies to the root element that may inherit from a more general model class.

Another drawback of MVC is, that frequently also view data has do be stored in the model, thus, it will be maintained twice: ones in the model and another time in the figure. Typically this applies to diagrams with a layout based on coordinates that have to be loaded from the persistence layer and it will apply to a whiteboard as well.

A more general recognition of MVC also allows widgets that are model and controller at the same time. But GEF's strict separation of the layers does not allow them leading to duplicated data in frequent cases.

5.1.3 Editing

Usually while working locally, an edit will be done by a command. As the EditPart is an observer, the API also works if the model change is caused in another way (like by remote peers). **Commands** can be provided by the Eclipse workbench as so-called **Actions** from the menu or a toolbar – or by **edit policies** that define the direct editing in the graphical editor.

Edit policies establish the duties and responsibilities of edit parts (pool or chain of responsibility pattern). They are used to contribute feedback, commands and targeting. They are freely plug-able helpers and handle specific features to modify the model allowing dynamic modification and hinder restrictions due to single inheritance.

They receive requests containing the necessary information from the system to create corresponding commands.

Commands at the end, in combination with GEF's own command stack offer the infrastructure be to executed, undone or redone.

5.1.4 Miscellaneous

Apart from the Eclipse workbench integration, GEF offers quite some useful interface extension like a customizable palette, drag and drop support, properties view, or a variety of tools for selection or creation. There are interchangeable and freely arrangeable edit part viewers like an outline or tree view.

Expectedly, as framework it offers infrastructure that still lacks a working implementation. This also explains the existence of a couple of predefined role names for edit policies, requests and properties to facilitate the maintenance of the place-holder IDs although any unique string would do it.

5.2 Prototype for GEF feasibility

I already presented in the introduction that the approach of the Eclipse Sketch project is not feasible for Saros. However, analyzing GEF's edit policies reveals that actually there is a proper built-in infrastructure to provide user feedback in form of the feedback layer.

The following prototype consisted of an Eclipse RCP[27] platform with a GEF GraphicalEditorWithPalette, a creation tool and a simple rectangle model with edit part and a custom edit policy. The RCP approach was chosen because it is much less effort to use the pre-built editor than to integrate it to Saros as a view, besides it is much faster to start a small standalone application instead of the whole Eclipse IDE with plug-ins. The edit policy visualized a custom feedback in form of a GEF PolylineFigure, tracking the location points of the request, however, it did not yet create a respective model neither a command

[27] Rich Client Platform, http://wiki.eclipse.org/index.php/Rich_Client_Platform

though (this was impossible because a creation request in GEF only transmits the position and maybe the size).

In a second state, a timer thread randomly created rectangle models and added them in the SWT thread to the root to distinguish properly collaborative work.

5.2.1 Results and further steps

Both, freehand drawing and concurrent editing worked without any problem and there outcropped several further aspects.

Above all, it made clear how a proper freehand drawing feature has to be implemented in GEF. We need a request that sends a point list instead of a layout rectangle, a tool to create it and an edit policy that can receive point list requests for the feedback and to return a corresponding creation command.

Another aspect are GEF's facilities like the mentioned editor. Eclipse maintains separate hierarchies for views and editors. This can be experienced in the way that editors can only be added to the editor area (usually in the center) while views can be arranged freely all around. As an editor, the whiteboard would hide other editors tabbed in the same editor part – this is not what we want. On the other hand, it still has editor features. Apart from editing, it maintains its own command stack, allows undo and should allow saving and reloading. Hence, it would be nice to be able to hook the editor in a view.

In the following I used the created prototype to give this a try. With some adjustments it worked perfectly. The only difference is that actions should be hooked directly to the view's toolbar and not to the workbench one's.

The whole evaluation was done in few days and it resulted that the Graphical Editing framework is a perfectly feasible for a Saros whiteboard feature. As well it clarified the further steps how to implement it.

5.2.1.1 For the start - the whiteboard as Saros plug-in

Finally, I experienced it as much more comfortable to work with a standalone application while developing the graphical feature. To do this without messing up Saros too much I decided to leave the whiteboard as a plug-in at first as this could be done without extra effort.

In Saros, new features are usually implemented in a separate branch. After time, this branch may slowly differ more and more from the Saros trunk version. A problem I already experienced while creating the batik prototype that used an own branch. After some time I could not apply useful patches anymore to my version without adjusting them. Another, even worse, problem usually is to merge these separated versions to trunk at the end of the work. Keeping the whiteboard in a separate module would always allow to use the current Saros trunk and merging at the end can be done without problems.

Actually this was my plan from the beginning because the whiteboard does not interfere with most Saros parts except the network layer. Keeping it separate, satisfied the Separation of Concerns pattern (see 7.13). But I was told this is unwanted. If so, the packages can be copied to Saros at the end omitting the standalone package.

5.3 Evolutionary prototype for the graphical editor

As described in section 1.5 for the evolutionary prototype I have to exercise more care, however, still the editor and basic infrastructure can be used from the previous prototype.

One thing I noticed in GEF was that it requires quite some understanding of its architecture, besides deep hierarchy and pattern understanding in general as well as to know what actually the framework already does for you to be extended in a proper way. A common case was that I quickly resolved the problem with big and confusing classes but in continuation I could refactor it to become shorter and easier to understand.

Thus, the following description does not correspond to my chronological steps in time but logical steps after achieving more knowledge about the framework facilities and the common way to extend it. Furthermore it is more suitable to describe the whiteboard in architectural parts.

Figures, edit parts and other basic requirements are omitted because all used figures are given by GEF and edit parts as well as the edit part factory correspond to the model (the general functions are explained in section 5.1.2).

5.3.1 The editor/view

The editor at first has to define all actions and editor facilities like the palette, the tools and status bar by creating, configuring and initializing corresponding parts.

In the beginning, we can almost use the GEF editor with a palette[28] as it is. The only thing to do is to populate the palette with tools. The final editor consisted of a selection, marquee and panning tool in a group to manipulate the editor or current selection and the pencil tool as well as creation tools for rectangles and ellipses.
New graphical elements can be added by drag and drop, too, thus, as a further step the palette has to be configured as drag-source while target edit parts or rather their edit policies will take care of dropping. Those items have a fixed size that I adjusted to the current zoom. Logically, polylines cannot be created like this, however, for convenience a drag and drop of the pencil will add a vertical line (instead of doing nothing or throwing an exception).

To enable zoom there was to set a zoom enabled root edit part as well as to configure the zoom including zoom actions with keybindings (“+”, “-”) and mouse wheel zoom support.
Note that the editor uses a zoom multiplier. The point is that GEF uses integer based layouts, however, on high zoom levels this would result in choppy curves (on ten fold zoom one fix point for a line would be every 10 pixels only) that is especially annoying for freehand drawing, for shapes this more likely feels like a raster. Another advantageous aspect is that for example SVG allows float values that thus would not be represented properly on higher zoom levels. Instead the coordinates could be translated by the zoom multiplier to provide correct presentation respective a number of decimal places.

[28] GraphicalEditorWithPalette

Finally, the editor has to create and register the available workbench actions. GEF offers ready-made undo, redo and select-all actions, likewise I created a copy, paste and delete action.

5.3.2 Freehand drawing

In a first step after creating the polyline model, I needed a **request** that caches points instead of creating a layout rectangle from start and current point.

Afterward, I could implement a **creation tool** that maintains the current tool state to track points while dragging using this request.

Finally, there was to implement an **edit policy** that extends the GEF XYLayoutEditPolicy but also understands this new request. It will add or erase a line feedback and forces extending subclasses to provide a respective command.
Features to mention are that this class translates the points from the absolute editor coordinates respective current zoom, scroll and target model. Furthermore it offers utilities to remove redundant points. During mouse drag the request tracks an arbitrary amount of points (respective drag speed and computer performance). Some of them can be omitted without changing the resulting line because they are collinear.

5.3.3 Commands

Commands in GEF have to provide an execute, undo and redo functionality. Thus, for example for creation, the undo had to delete the model again. For a move, the command has to store the previous layout and the previous parent if applicable to be able to execute an undo. Furthermore they take care to check requirements, for example a deletion cannot be undone if the previous parent does not exist anymore in the hierarchy.

The editor has commands for all kinds of possible editing by edit policies and/or actions, including creation and manipulation, special to mention:

The copy command can never be undone. Following tutorials it adds references to the selected items to the clipboard while the paste command clones these elements. In this case, if the user moves the elements after the copy, pasting them would clone their current state but not the state during the copy. Thus, already the copy has to create clones which – on the other hand – cannot be added directly by a paste because multiple paste is possible.

Another aspect is that by just cloning a model in an XY-layout, it would appear at the same position like the original hiding this and leaving unclear whether the execution was successful until moving the top element away. To avoid this, the paste command shifts the elements' positions a certain amount to the down-right.
However, a second execution of the paste action would then hide the just previously inserted figures. To solve this the paste action maintains a counter of previous executions. This integer is passed to the command so that it multiplies the shift respectively and finally non of the executions will be hiding another one.

5.3.4 The model

Although incrementally adding features worked well respective most parts of GEF, the persistent representation required more knowledge and evaluation again. For the start in this prototype, I used the common way in GEF, creating simple custom model classes for each object and incorporating the Eclipse PropertyChangeSupport with the edit parts as PropertyChangeListeners. As I still had the XML infrastructure from before, these models were wrappers of an underlying XML document were the view data was stored in an SVG like manner. Consequently the most general abstract model class was named ElementModel that understands GEF layouts to resize and relocate itself and may have child models, thus nesting is general allowed, although a method is used to distinguish whether really a child can be added (for instance a polyline cannot have children). All other models were subclasses of this one.

5.3.5 Completion

After providing the infrastructure, I added the application specific edit policy[29] that provided the commands to create a polyline out of a point list or any kind

[29] ElementRecordLayoutEditPolicy

of ElementModel by a layout rectangle, the command to change a model's size and position as well as the command to change a parent.

The mentioned edit policy also proves if the target is a composite and not given it will change the target as well as translate the layout to the next element that is a composite.
Other features are that it ensures that no element becomes smaller than a minimum size.

5.4 Adjusted network prototype

The created activity in chapter 4 cannot be enough anymore: elements can be modified now. To quickly achieve a running version, I used the defined properties from the eclipse property change support to send the changes to peers, in other words the network manager of the whiteboard became a listener to the model. Unfortunately, at first this caused a cyclic behavior and I had to manually set and check the source of a property change event (Eclipse did not do this like I expected).

Another issue was that in this way an event was send over the network for every property change and not for every command. A better solution seemed to be to solve sending on a command level.

Finally, I noticed that the application actually did non access the underlying XML document anymore so it became questionable why to maintain it. Even without it would be possible to convert the models to an SVG for saving and loading.

However, the basic collaborative drawing feature worked, mind that without synchronization only.

5.5 Feedback by team members and conclusions

At this stage there was the first proper evaluation by team members. First of all, this prototype confirmed assumptions from the inaugural presentation: all current members saw it as preferable to draw shapes and to have the possibility to rearrange them, copy and paste as well as undo/redo also attracted interest. Furthermore to mention is that nobody missed a pixel eraser tool.

It was mentioned that a bigger variety of shapes would be nice as well as colors but the most notable missing feature was synchronization. Being built on the Saros activities structured, messages arrived with more than half a second delay in average so it was easy to create inconsistencies. Thus, the next step of the work became the synchronization of distributed hierarchical data, like already introduced in chapter 3.

Apart from this, the prototype confirmed GEF as external underlying rendering library. The editor and editing features were approved and doing the same only with SWT and JFace would be out of range. I want to remind the SWT paint example from section 2.1 that revealed that fast and flicker free painting with feedback already acquires a notable effort, selection and modification not to mention.

6 Concurrency control for collaborative XML editing

In chapter 3 I gave a general introduction to concurrency control (CC) algorithms for hierarchical data structures like XML. During my work there also raised the question why I do not transform the XML in a simple linear text file using the existing CC system for linear data structures implemented in Saros.

6.1 Reasons for a specialized concurrency control approach

6.1.1 Well-formedness of an XML document

Actually the XML specification[30] defines an XML document as a text, but a text that is well-formed, meaning it satisfies a list of syntactical rules. If these rules are not satisfied, the text is simply no XML and a parser has to cease

[30] http://www.w3.org/TR/REC-xml/#sec-well-formed

processing. These rules, for example, define that every tag has to be delimited by “<” and “>” (or “</” and “>” respectively for closing tags) and that tags delimiting elements have to be correctly nested. Thus, adding a start-tag only would already break the XML.

However, it is thinkable that as long as an application only produces well-formed changes the before mentioned failure could also be avoided when using a algorithm for a linear data structure like GOTO (see section 2.3.2.2). The application would have to apply all changes of an operation before parsing the text again. If moving an element is realized by deleting and inserting again, even moving an element could result in a well-formed document. But as soon as two peers concurrently add an attribute with the same name the procedure would fail. The CC algorithms would not even be aware that the clients changed an attribute. Another problem is that attribute order is not specified in an XML and thus can be platform dependent and differ from client to client.

6.1.2 The DOM tree in the application

If an XML is part of the application's logic (for example the model representation of a whiteboard), it will be available as parsed DOM tree to use the advantages of a hierarchical document. To make a CC algorithm for linear data structures applicable, the application would have to serialize the tree to text, apply the algorithm and parse it again. First of all this may induce some problems with concrete parsers – a parser might change the combination of start and closing tags to a degenerated one if the element has no child elements anymore or change the attribute order breaking the stored indexes of the operation history. Apart from that it could cause a severe performance problem making the application unusable.

More problems and considerations can be found in the works to CEFX ([Ger07]) and treeOPT ([Ignat et al.03]), more precisely they are consequences of the fact, that CC algorithms for linear data structures can not simply apply to hierarchical data structures.

At the early stage, there was the agreement with a team leader to follow the Shared XML Editing (SXE, [XEP-0284]) approach to implement concurrency control (CC) in the newly created whiteboard.

The reason was simple: the two other above mentioned approaches are scientific works with a much wider scope than this work could implement, taking in count operation that are not supported by the whiteboard anyway (for example split and join of nodes). Furthermore, in the beginning I did not have access to the respective documents, but the most important argument was that there did not exist any confirmation or verification that these algorithms ensure convergence.

SXE on the other hand seemed to be approved by the mentioned instant messengers (section 3.1) and on its way to standardization. Thereon, let's first of all present the principles of this experimental XMPP extensions.

6.2 Shared XML Editing – XEP-0284

The base principle of SXE can be described by the term “record”. Records correspond to the data structure of the shared artifact and are also means of communication. There is no way to change the shared artifact but records. There are three different types that are explained in the following as well as their relation to other approaches.

For the start, have a look at the following sample edit sent by the whiteboard implementation to create a rectangle. More precisely it describes the XMPP packet extension, the message tag is omitted.

```
<sxe id="2" session="7963" xmlns="urn:xmpp:sxe:0">
     <new name="rect" rid="749146970812179611"
          parent="-903211645218651747" type="element"
          primary-weight="0.16" version="0">
     </new>
     <new name="x" rid="7208659102388296971"
          parent="749146970812179611" type="attr" chdata="580"
          primary-weight="0.92" version="0">
     </new>
     <new name="y" rid="4435997654315217511"
          parent="749146970812179611" type="attr" chdata="2150"
          primary-weight="11.38" version="0">
     </new>
```

```
          <new name="width" rid="-7966461684004231930"
               parent="7491469708121796ll" type="attr" chdata="590"
               primary-weight="21.58" version="0">
          </new>
          <new name="height" rid="-1525783545821880161"
               parent="7491469708121796ll" type="attr" chdata="710"
               primary-weight="31.73" version="0">
          </new>
</sxe>
```

As may be noticed, a record has to be created for every node of the DOM document, whether an element or an attribute (or others). Having only an empty SVG root with the ID `rid="749146970812179611"`, the edit would result in the following XML document:

```
<svg xmlns="http://www.w3.org/2000/svg">
     <rect x="580" y="2150" width="590" height="710"/>
</svg>
```

Please note that XML does not specify attribute order. Thus, the order they are actually printed out may be implementation dependent and not connected to the internal order due to the weight mechanic (see section 6.2.4).

6.2.1 The new-record

Field Name	Mutable	Applies to nodes of type	Description
rid	no	all	The ID of the record.
type	no	all	The type of DOM node (element, attr, etc.).
version	Indirectly	all	The current version of the record.
parent	yes	all	The record id of the record corresponding to the parent node.
primary-weight	yes	all	The primary weight used to determine the order of sibling nodes corresponding to the records.
ns	yes (1)	element, attr	The namespace of the element or attribute
name	yes (1)	element, attr	The name of the element or attribute.
chdata	yes	text(2), attr, comment(2)	The content of a text node or a comment or the value of the attribute.
pitarget(2)	yes	proccessinginstruction	The target of the processing instruction.
pidata(2)	yes	proccessinginstruction	The data of the processing instruction.

Figure 7 - The fields of a new-record[31]

Notes: (1) The XEP labels these fields with a TODO: “is this reasonable?” In the context of an SVG whiteboard it is actually not. If you imagine an SVG rect element, it uses completely different attributes for its position as for example the ellipse tag. A conversion would not make sense and require a specialized validation. Furthermore it does not feel intuitive to change one shape to another.
The same applies for the namespace that in the context of SVG should never change.
If still willing to change these fields a record can be deleted and recreated.

(2) These elements are omitted in the whiteboard network protocol because processing instructions like CSS style sheets will not be supported as well as any other processing instruction. Only a small subset of the SVG standard is to be implemented, see section 4.8.

In the context of the Document Object Model, the new-records correspond to DOM nodes and their specializations respective the type field (element, attribute, processinginstruction, comment and text). If a client receives a new-record for a parent that does not exists, this new-record has to be discarded.

The version field is used for synchronization that will be described later. As mentioned in the notes, the application specific new-record will only have three mutable fields:

- parent – to move a record and its child nodes
- primary-weight – to specify the index of a child node
- chdata – the data of a node

These fields can be changed by a set-record.

6.2.2 The set-record

Attribute	Description
target	REQUIRED and MUST be the record id of the node being modified.
version	REQUIRED and MUST be the current version of the node incremented by one.
parent	OPTIONAL to change the parent of the existing record with rid=target

[31] http://xmpp.org/extensions/xep-0284.html

primary-weight	OPTIONAL to change the primary-weight
chdata	OPTIONAL but only allowed if the target node is of type 'attr', 'text', or 'comment'.

Figure 8 - the fields of a set-record and their scope of application

Notes: All fiels that are non-mutable in the scope of this work are omitted.

SXE also specifies the fields replacefrom and replacen to partly change the chdata value of an existing record. This is omitted in the implementation as modifying an existing chdata field is neither required nor supported at the current state.

set-records in the context of shared editing correspond to the local history of a DOM node. On conflicting edits, they can be used to discard changes and revert the state to a previous non-conflicting version. Similar concepts are found in the treeOPT algorithm.

6.2.3 The remove-record

Attribute	Description
target	REQUIRED and MUST be the record id of the node to be removed.

Figure 9 - the field of a remove record

This record actually goes beyond a remove operation in the other approaches. Due to the lack of an undo-feature in SXE, not only the target record can be removed completely but also the histories. On the other hand, it comes along with a requirement: first clients have to send remove-records for all children before removing the desired record.
The following figure explains why this is necessary. Convergence is not ensured if a remove record is not sent for children, too. The numbers in the shapes correspond to their record IDs (RIDs).

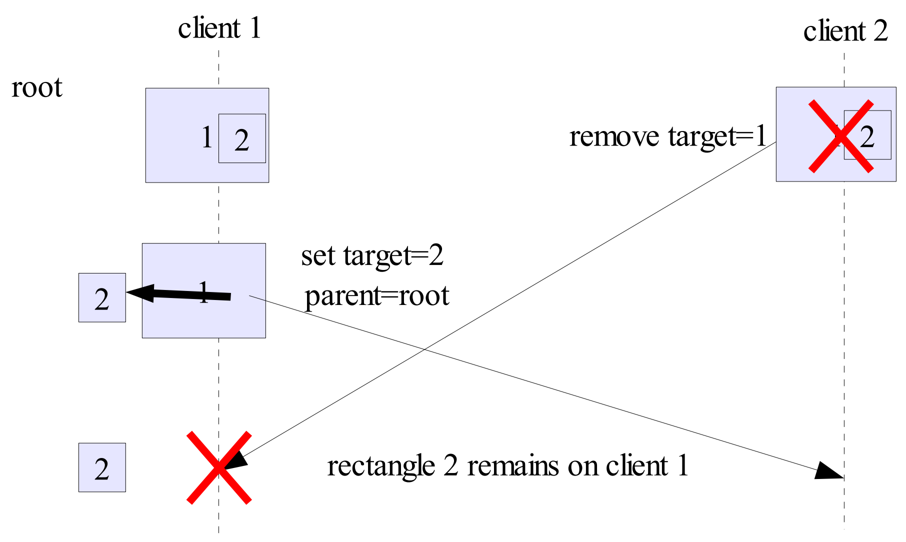


Figure 10 - divergence if omitting remove-records for children

Note, that due to the SXE solution to send remove-records for children first, the rectangle 2 would be deleted on client 1 together with the other rectangle. One may argue that this does not preserve the user intention but this is questionable (maybe client 2 wants to delete all children explicitly).

Finally, the protocol specifies, if a client receives a remove for a record that has children (for example concurrently newly created ones), the client has to send remove records for these children, too.

6.2.4 Weight of a record

In terms of SXE, the weight of a record defines its position in respect to its siblings. The value is primarily defined by the float of the primary-weight field (see figure 7), secondarily – if these values are equal – by the lexicographic ordering of their record IDs as Unicode value (in Java and thus in the provided implementation this can be done by the string compare method). As a consequence, concurrent creation of new nodes or concurrent child order changes of different nodes to the same parent does not need any concurrency control effort because the position of nodes is given in a global manner. Due to the (theoretical) infinite range of float numbers, reordering of children can be done without limit.

If a client wants to move a node N between two nodes with the same primary-weight (PW), first he has to change the PW of one of these nodes, thus that he can change the PW of N to reside between the two desired nodes. Having more nodes with the same PW, consequently may require subsequent PW changes until achieving the required gap. Of course, one or more of these additional set-records could conflict with other edits from peers leading to an unintended result, but convergence is preserved in all cases.

Another drawback respective user intention can be seen in the following figure:

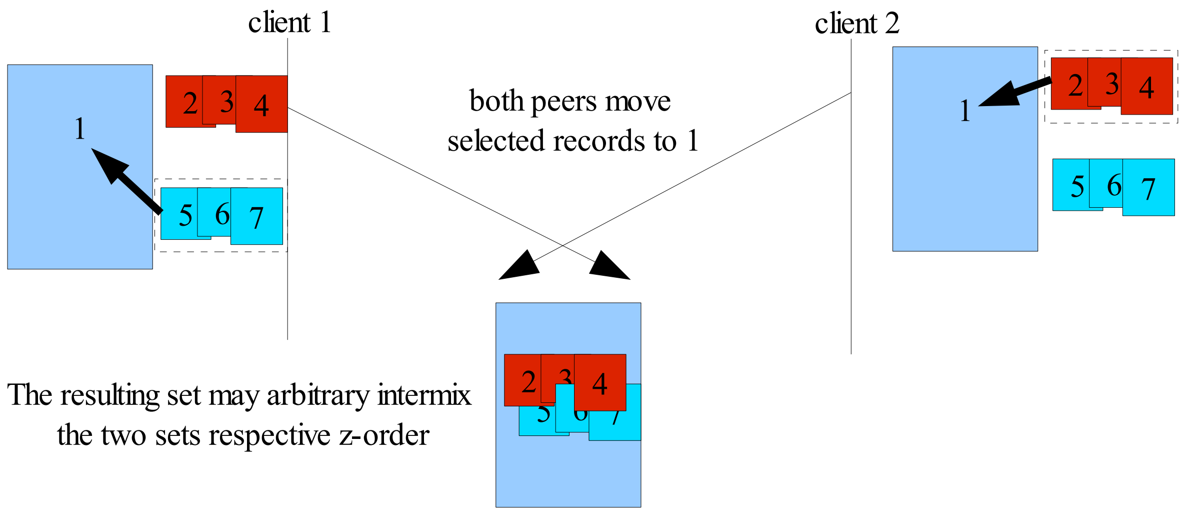

client 1
client 2
both peers move
selected records to 1
1
2 3 4
5 6 7
The resulting set may arbitrary intermix
the two sets respective z-order

Figure 11 - intermixing concurrently moved sets due to primary-weight mechanic

A user would expect to add the whole set on top or beyond the concurrently added set without intermixing them. If these operations occur one after the other the problem can be avoided: moved records are added as last children to the new parent and thus appear on top.

The other mentioned algorithms, treeOPT or CMAX (from CEFX) do not have the problem. treeOPT defines the position explicitly, so that it has to be transformed by the operational transformation algorithm, while CMAX defines the new position beyond the siblings by a reference element (the element after) corresponding to the DOM method insertBefore(newChild, refChild).

6.2.5 Commutative and non-commutative edits

SXE divides editing actions in two categories: commutative and non-commutative.

Commutative records cannot conflict with any other record and thus they are never undone to a previous version and a history is not required. New- and remove-records are commutative. Respective new records, the weight helps to define their order. If the parent is missing they must be discarded as well as must remove- and set-records if their target is missing (the record was already deleted by a peer).

Set-records on the other hand are labeled non-commutative to each other because they change the same record and thus can conflict. On a conflict they have to be undone and therefore a history of set-records is required.

The following table shows possible pairs of concurrent operations towards a same target or parent RID respectively for new-records.

locally performed	**received**	**consequence**
new	new	add both new records respective their weight
set chdata=”a”	remove	record is removed
remove	set chdata=”a”	set-record is discarded, record stays removed
remove	remove	the second (remote) remove-record can be ignored
set chdata=”a”	set chdata=”b”	conflict resolution (revert to previous version)
new	remove	the parent is removed and the client has to send a remove record for the newly created record (see 6.2.3)
remove	new	the remote new-record can be discarded

Figure 12 - possible concurrent records to same target/parent

6.2.6 Conflict resolution in SXE

As demonstrated, the SXE protocol acts on the assumption that conflicts can only occur due to concurrent set-records. The conflict resolution is rather simple using the version field:
Every set-record will first increment the target version field by one. Afterward, if the version of the received record equals the local one, apply the set record, else discard the conflicting set-records and revert every field but the version field to a previous version. The previous version in the protocol description ([XEP-0284]) is defined to be the set-record's version minus one (v - 1), however, this is actually not very precise. The fields have to be reverted to the last non-conflicting version smaller than v. This is given indirectly by specifying

to discard conflicting set-records – thus, it might be that there is no set-record in the history for v – 1 anymore because it was discarded, too, but it should be specified explicitly.

A simple example can be seen in the following figure, where two clients try to change the “width” attribute-record of a rectangle concurrently. Note, as abbreviation “new type=attr” is replaced by “attribute” and other attribute-records than “width” as well as the element record are omitted.

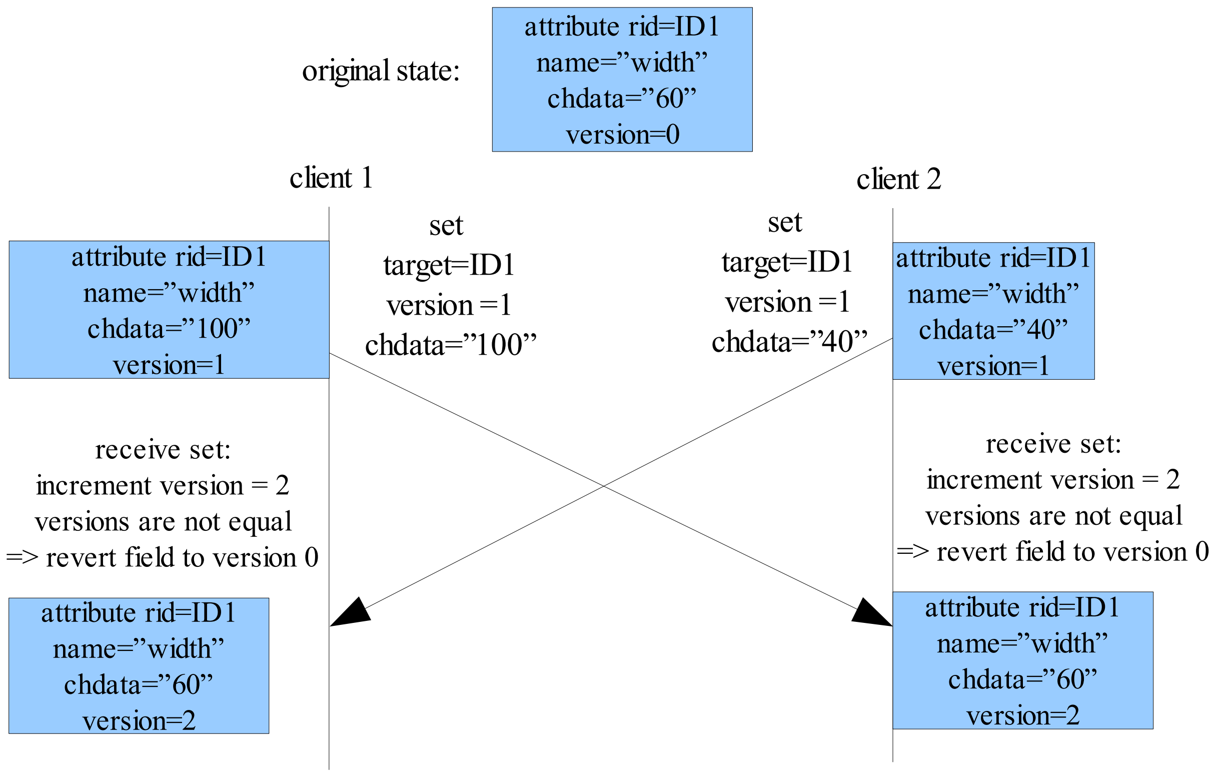

original state:
attribute rid=ID1 name=”width” chdata=”60” version=0
client 1
client 2
attribute rid=ID1 name=”width” chdata=”100” version=1
set target=ID1 version =1 chdata=”100”
set target=ID1 version =1 chdata=”40”
attribute rid=ID1 name=”width” chdata=”40” version=1
receive set: increment version = 2 versions are not equal => revert field to version 0
attribute rid=ID1 name=”width” chdata=”60” version=2
receive set: increment version = 2 versions are not equal => revert field to version 0
attribute rid=ID1 name=”width” chdata=”60” version=2

Figure 13 - conflict resolution in SXE

6.2.6.1 *Undoing conflicting edits*

As noticed, conflicting set-records are reverted, that is actually a very feasible approach. Most existing operational transformation algorithms do not ensure semantic consistency. This fact is worked out with more details in [Ign02].

In a text file, the clients see the effect and it is easier traceable what actually had happened when the semantic changed involuntarily. But if the XML is part of the application logic this might lead to quite confusing results. Have a look at the following figure. Actually both clients want to change the x-position of an element to 20.

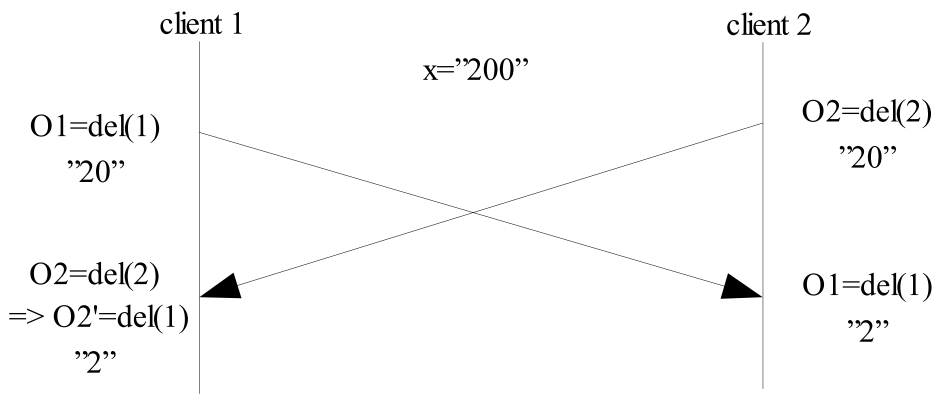


Figure 14 - Semantic inconsistency problem with OT

Still it would be possible to defer to one peer but even this may be seen as more confusing than to discard the change forcing the users to communicate their ideas. A similar idea can be seen in the Action-Constrains Framework ([Shapiro et al.]), on antagonistic operations clients have to give votes for one of the operations based on a so-called multilog that is means of communication. In SXE, however, this functionality is not protocol immanent and thus only the application can provide a feasible mechanic. To make this possible the provided implementation informs the model about conflicts transmitting the previous, the remote and the reverted state. At the end of my work the GUI part will still be missing though.

6.2.7 Start synchronization

The session negotiation part of SXE uses Jingle and is similar to other XMPP Extensions. However, it will be skipped in this work as session negotiation is part of Saros. This way we come to start synchronization directly.

The defined data flow starts with the client sending an SXE connect message to the host, that will answer with a state-offer that can be accepted (accept-state message) or refused (refuse-state respectively). Note, that this protocol does not manage multiple documents in one session, instead a new session has to be negotiated for every shared object. This would also apply to the whiteboard if it is supposed to support multiple pages.

After accepting the state, the host answers with a state message that in principle consists of the set of records currently stored for the shared artifact, including new-records and the respective set-record histories. A client is supposed to apply these received records one after the other with a small difference respective set-records. In a session, they are applied like described above expecting consecutive version numbers, during synchronization in contrast, when applying a set-record, the target's version has to be set to the version field of the set-record to avoid a conflict. The reason is that a state message will not contain records that were discarded or reverted due to previous conflicts, so that it is expected that the version numbers are not consecutive anymore.

6.2.8 Hybrid architecture, MUC and peer-to-peer communication

Actually, SXE is specified using a similar approach to Jupiter, thus a hybrid architecture of a centralized and replicated system, where the host corresponds to the central server. However, the server in this case does not have any functionality but relaying messages to peers to avoid that a latter message gets ahead a previous one. The host can be the initiator – thus a peer – of a session or a multi-user chat ([XEP-0045], in the following referred as MUC). If using a peer, this induces a high load on the host as is visualized in the following figure.

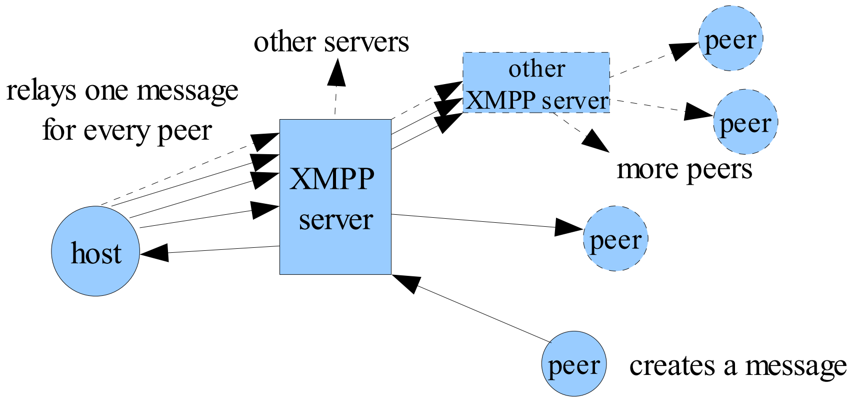


Figure 15 - relaying one message by the host

This problem is not trivial. If 10 peers concurrently do an edit, the host will have to relay 100 messages.

Therefore, SXE mentions that a MUC should be used for sessions with a large number of participants.

In Saros, a timer was introduced to the activities architecture, triggering every second. Every peer collects activities so that he will send only one message to one peer in a second (or 10 messages to 10 peers respectively). Thus, the host in Saros would collect the edits of the 10 peers (or rather all edits received in one second) and send only 10 messages (one to every peer) encapsulating the 10 edits. As long as the host is editing (in PP jargon being the "driver"), the delay will be from 0 to 1 second plus the network delay. If any other peer is a driver and editing, the delay would add up to 2 seconds.
We will talk about this Saros-specific implementation later in section 9.7.

In [Voi09], the Collaborative Editing Framework for XML ([Ger07]) was extended and evaluated. During his work he noticed a significant performance and network load problem because of the operational transformation algorithms (CMAX) and the hybrid architecture where a host had to relay messages. This issue already took place with a comparatively small number of participants (less than 10). As a consequence he introduced a MUC as distributor to reduce the load on the host – afterward it only had to relay one message to the MUC that will distribute it further.

On the contrary, using SXE the MUC can be the host like presented in the following figure.

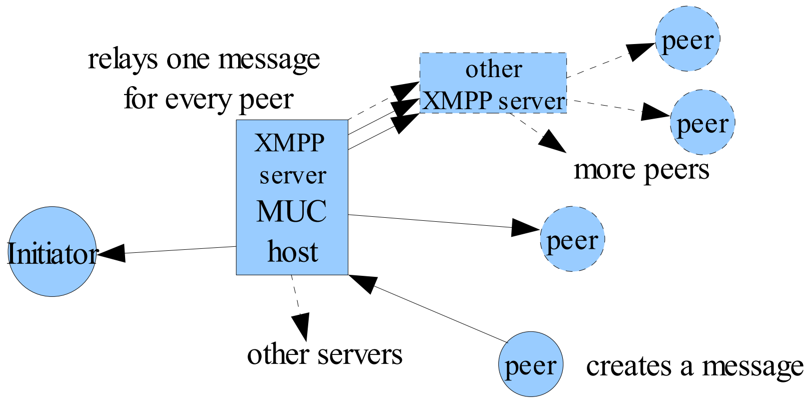


Figure 16 - relaying a message using a MUC

This is possible because relaying does not require any kind of specialized action (neither transformation nor reordering or delaying of messages).

In continuation, having a closer look to the protocol it makes it questionable why it should not work peer-to-peer (P2P). Please note, that in an XMPP network, P2P communication as thus does not make sense because every message has to be relayed by one or two servers anyway. But in an hybrid architecture having a peer as server the ways are doubled and the amount of messages multiplied at the host site that could be avoided. Furthermore Saros implements SOCKS5 bytestreams, in other words it is possible to have a real P2P connection to the other clients so that this issue gets more interesting.

As SXE does not depend on the central component for any transformation the only problem in P2P communication would be out-of-order messages. Hence, it could happen that a child node is received before its parent (or rather the new-records respectively).
For an example have a look at figure 17.

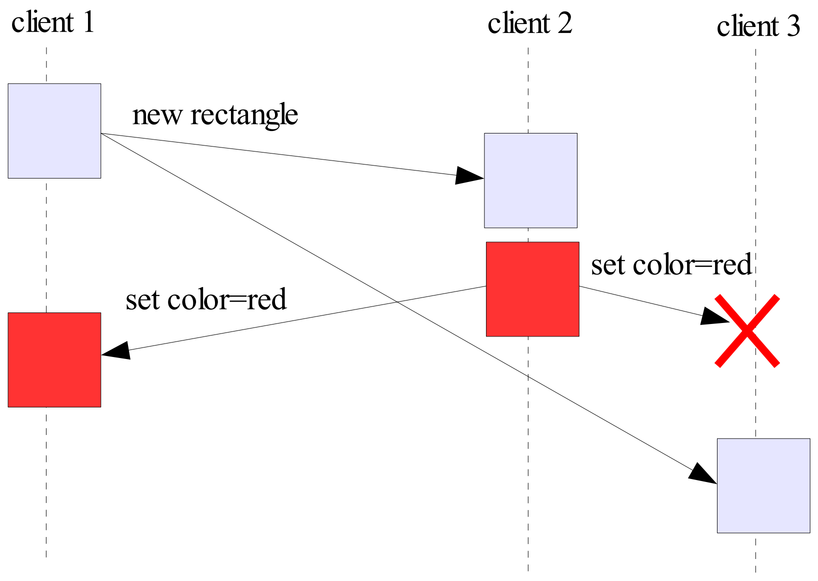


Figure 17 - out-of-order message due to P2P communication

Client 2 and 3 may reside in a local area network and due to some reason (i. e. traffic shaping or high load), client 3 receives the newly created rectangle with a higher delay than client 2. As a consequence he would discard the set-record to change the color while it was applied on the other peers, convergence is not assured.

6.2.8.1 *Causal readiness*

This problem is deeper analyzed in the paper [Ignat et al.03] that introduces the term of causal readiness and its solution can also apply to SXE. Upon receiving a remote record, first it is tested if it is causal ready. If so, it is applied, else it is queued. Additionally, after applying any other record it is to check whether we can apply a queued record now.

Non causal ready are:

- new-records if their parent record is neither deleted nor available locally.
- set-records and remove-records if their target is neither deleted nor available locally.
- a set-record if it has a version higher than the correct one.

The last point requires some closer inspection. Having a record with version 1, imagine a set-record to this target arriving out-of-order with the version 3 (before version 2). In accordance to the protocol the target would have to be reverted to version 2 (v-1). This case is not explicitly specified as there is no set-record with version 2 available but it is most likely that the record remains unchanged. Afterward only, the set-record with version 2 arrives resulting in a second revert to version 1. Hence, on the peer where these records arrive in wrong order, the target remains unchanged while on other peers both records are applied.
This scenario shows that actually it is very useful not to apply any set-record with a version higher than the correct one. Even if not working P2P this would help to detect an error as early as possible, for example a message in XMPP could get lost[32]. Due to the version mechanic this is extra severe: If a version is missing at one peer, every subsequent set-record would fail to be applied.

As a drawback, enabling P2P, the peers now have to track the RIDs of deleted records. Without this, it would be impossible to decide whether a record is not yet ready or already deleted. Also without out-of-order messages it would be good idea to keep a history of deleted RIDs because clients could distinguish errors (fail fast).

6.2.8.2 A note for simplicity

Several times during my work I was told that a solution should be simple at first place. This chapter might give the impression to cause some overhead. Please note in this context, that it is much less effort to use the existing Saros

[32] for example see user discussion: http://comments.gmane.org/gmane.network.jabber.psi.devel/8564

P2P network facilities and to add two tests for causal readiness (tests that have to be done anyway for error checking), than to implement a MUC specific structure.

6.3 Considerations - How to integrate SXE in Saros

At this stage, I came to analyze the existing Saros infrastructure more profoundly that was already briefly described in section 2.3.2. Unfortunately, it revealed that due to quite different premises there are no specified points in Saros where it could be integrated. On some parts it would cause a lot of overhead or it would mess up existing code in an exaggerated way.

In the following you find considerations to selected Saros parts and their relation to the proposed whiteboard concurrency control protocol SXE.

6.3.1 GOTO algorithm and operational transformation

In section 6.1, I pointed out that a specialized approach is necessary for collaborative editing of hierarchical data and the GOTO algorithm cannot do it. Now, after having had a closer look to SXE, it is obvious that it does not define an algorithm for operational transformation at all, rather it defines a data structure to ensure convergence. Neither records are transformed at any stage nor this is necessary.

6.3.2 Jupiter and relaying messages to peers

We already talked about Jupiter, the hybrid approach of a centralized and replicated communication topology in section 2.3.2.1.

Saros editing activities (“Jupiter activities”) are sent to the host, where they are received by the Jupiter class, sequenced and transformed a first time. Then they are forwarded to other peers that apply the final sequencing and transformation.

SXE on the other, only depends on the host to relay messages and could even work P2P with a small modification as explained above. The relaying could be done in three lines in an SXE controller class and it did not seem convenient to change an existing concurrency control infrastructure for text editing by this relay mechanic. Furthermore, it seemed more advisable to use a multi-user chat ([XEP-0045]), thus, the host of a Saros session would not need a specialized component at all.

6.3.3 Consistency Watchdog

The Consistency Watchdog is a Saros component to verify consistency beyond the peers. Therefore, on the host site it calculates check-sums for open files while it uses received check-sums to verify the files on non-host sites.

First of all, there is an issue when applying check-sum algorithms to DOM trees or XML files. We already discussed that attribute order is not specified in an XML. As a consequence, it could happen to calculate two different check-sums although the XML documents are actually same. This problem could be solved by using the same parser implementation. As an alternative, a custom check-sum algorithm could be provided. Note, that in SXE attribute order is specified by the primary-weight and there wouldn't be a problem.

However, in Saros, the consistency watchdog is necessary because files could be changed outside the mechanisms it uses to capture edits, for example the file is edited from outside Eclipse or by another Eclipse plug-in Saros cannot access easily.
With the whiteboard on the contrary, the situation is different. The only way to change the shared artifact is the embedded editor and no external modification is possible. Furthermore, the only way to change the artifact is given by records and the implementation has to ensure that a record is always applied in the same way. If a remote record cannot be applied due to some reason, the client has to take respective actions to correct the situation or accomplish a resynchronize or reconnect.

As a consequence a parallel mechanism to ensure a synchronized shared object would only hide protocol or implementation errors and as thus it is discouraging.

6.3.4 Initial synchronization processes

We already pointed out in 2.3.2, that Saros does not define a standard way to perform start synchronization. However, project synchronization is done by files and was thereby not applicable for the whiteboard in my opinion. Furthermore the SXE protocol gives a clear description on how it is to be performed.

Another aspect is that the Saros classes for start synchronization already had grown a lot over time. They interconnected almost every single part. That this is a bad concept, was brought up at the end of my work by a team member that wanted to add a new feature to Saros: to add a second (or more) project to the session. He tried to refactor the invitation process to make a dialog reusable. It resulted in a non-trivial task due to plenty of interconnections. If I had added the whiteboard communication chain, too, he would have had even more work extracting this code. Especially to mention as the whiteboard is session specific and not project specific, thus, the code had nothing to do anymore with the dialog he actually needed.

6.3.5 Activities

For the same reason, the activities structure confronted me with problems as it did not allow to modify the protocol. Activities will be encapsulated in a <timedactivities> tag, hence the protocol would not conform to the one presented in section 6.2.

6.3.6 Conclusion

As conclusion I proposed to keep the additional whiteboard concurrency control data structure out of existing classes.
The Jupiter implementation is very feasible for text editing but not applicable for SXE. The invitation process works but would be messed up even more than it already was. The additional initial communication chain of the whiteboard should be kept in a specific package and hence out of the central Saros classes.

Note, that this would not cause additional effort but facilitate maintainability, extensibility and error localization.

The mentioned issues and the proposed alternative was posted to the Saros mailing list to trigger some feedback.

An answer I got from the previous project manager supposed me to use activities because of the offered facilities mentioned in 2.3.2 (user identification, SWT context switching, sequencing and payload). Furthermore I was suggested to use Jupiter and to extend it with my new operational transformation algorithm.

6.4 Arguments against SXE

I misunderstood the above suggestion towards Jupiter and operational transformation (OT) as a “have to” and in the following – although I only had two and a half months left for my practical work – I had to become more familiar with OT approaches for collaborative XML editing. However, because I slowly had understood the principles of SXE I had noticed some issues that motivated me to investigate more anyway.

6.4.1 Intention preservation

An effect of SXE is that conflicts are only discarded and never transformed. Furthermore there are the issues mentioned in 6.2.3 and 6.2.4 where the user intention might not be preserved.

6.4.2 Maintaining a well-form XML document

Furthermore, investigating further I noticed some inconveniences in the protocol respective maintaining a well-formed XML document (well-formed, see section 6.1.1).

6.4.2.1 Attribute names

In general, due to concurrent edits two or more attribute records with a same name could be created and added to the document while the XML specification confines attribute names to be unique.

In the implementation this problem is solved by discarding the record with the smaller weight.

6.4.2.2 Circular hierarchy

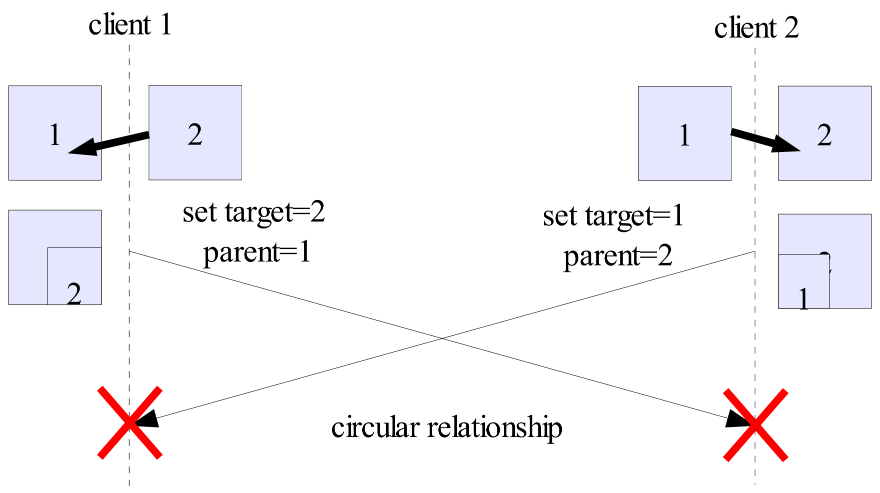


Figure 18 - circular relationship due to concurrent set-records

At best, the records will map to a DOM implementation that will throw an exception or the application tests for circular hierarchies maybe resulting in two differing shared object only. In the worst case, the application logic will get stuck in spinning due to the circular relationship.

SXE specifies that if a client changes the document state to an invalid one (for example because of a specified XML schema), the host has to send records to repair the situation. This could also apply here with some modification.

6.4.3 Ensuring convergence

I only notices on case where SXE does not ensure convergence, but there might be more and further research may be necessary.

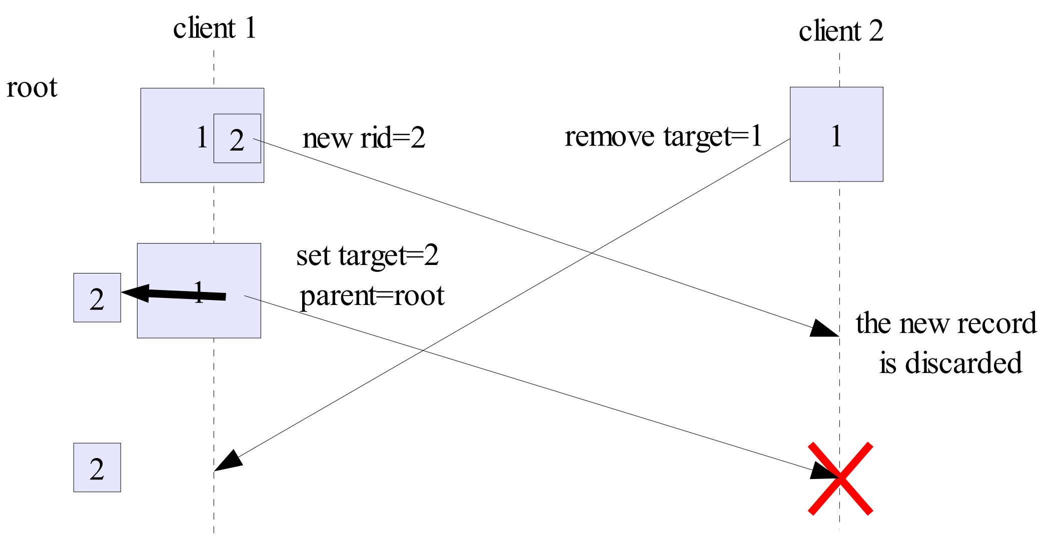


Figure 19 - not ensuring convergence due to concurrent remove, set and new

Without the set-record, client 1 would be obliged to send a remove record for the rectangle 2, too, and no divergence would occur. (see also 6.2.3)

A solution would be that a client has to answer with an remove-record instead of discarding a record that has no parent.

Another way to create an inconsistency is very similar to the case above with the difference, that the child node (like rectangle 2 above) is not moved out of the parent by a set-record but by a conflict. Hence, it cannot be enough to send a remove for a new-record only.

However, it was already mentioned in 6.2.8 that it would be useful to keep a history of deleted records. In this situation a client could answer with a according remove-record if an arrived set-record (due to conflict or not) would result in moving a node away from a deleted parent (or if trying to edit any deleted record in general). Doing so would not require anymore to send remove-records for all children first before being able to remove the parent as shown in 6.2.3.

This is the proposed solution for the provided implementation.

6.4.4 Lack of undo/redo support

SXE does not provide a undo/redo facility, in other words, there is no “undo-record” that would enable the user to undo a previous edit. The whiteboard implemented in chapter 5 on the other hand supports undo and redo, so it would be nice if this could be transmitted by the underlying protocol. However, the problem is not too severe as a undo-create corresponds to a remove (plus vice versa) and a set-record can be “undone” by setting the value to the previous one with a second set-record.

6.5 Further investigation

Remember the suggestion above about how to extend the existing Jupiter class and the mentioned issues that caused me to come back to further investigation. In section 3.3, there is already a brief introduction of the current state of research operational transformation (OT) for collaborative XML editing that had influenced my decisions up to now. In the following, the two mentioned works will have a closer inspection.

6.5.1 treeOPT solutions for concurrency control

The algorithm was already presented in 2002 ([Ign02]), but in 2006 only there was the mentioned adoption for XML ([Ign06]). The abstract data model is a tree of nodes with a history buffer for each level of nodes.

The most general idea is to apply an OT algorithm for linear data structures recursively over the document, in that way the work from 2002 explains a treeOPT-GOT algorithm and [Ignat et al.03] mentions that a treeOPT-dOPT combination can be easily performed.

Given these points, it does not make sense anymore to provide an XML-specific OT to the Jupiter class. But in the contrary, it might be thinkable that a new server component parallel to Jupiter reuses the implemented inclusion transformation of the GOT algorithm. Unfortunately, the transformation is “somewhat more difficult”[33] (assumed to be more efficient though), apart from

[33] [Ignat et al.03]

text-edits the path to a node has to be transformed using history buffers for each level. In other words, a real reuse of existing classes would be tricky.

Editing in a shared project, accordingly editing shared files in a shared directory tree, also corresponds to the collaborative editing of hierarchical structures. As a consequence, another adoption could be to provide an algorithm that covers not only single opened files but the whole shared project. Currently (status December 2010), so-called file activities in Saros are note controlled by an OT algorithms yet – they only reset the state of the GOTO algorithm in the Jupiter class – and concurrent actions may cause divergences. A solution would be great but completely out of the scope of this work though.

Especially it is questionable whether this could help to reuse anything. A closer look to the work of 2006 reveals, that the authors developed three different algorithms for hierarchical text editing (with levels like paragraph, sentence and word), XML documents and drawing. The reason might be that every single application already results in very complex algorithms and it is much easier to maintain three different and separated implementations than to interweave them.

Unfortunately, I had neither the complete documents nor the implementations for the treeOPT-algorithms on hand. Attempts to contact the authors failed, too, making it difficult to immerse myself deeper in the material. Especially, it is not yet completely proved that this algorithms ensures convergence in all cases, so that it does not result advisable to implement it in Saros yet.

6.5.2 Collaborative Editing Framework for XML (CEFX)

The first version was developed in 2007 as part of a doctor thesis by Ansgar R. S. Gerlicher ([Ger07]) and it was continuously extended. Especially, there existed a XMPP based version using a MUC that let it seem to be perfectly feasible for Saros. As mentioned, it is not based on operational transformation though, thus an integration to the Jupiter class would not be advisable. But as it is an implementation for the requirements of the whiteboard concurrency control it required further inspection.

The underlying algorithm is called Consistency Maintenance Algorithm for XML (CMAX). Like SXE, it does not perform operational transformation if clients edit the same attribute concurrently. The reasons mentioned are that the probability for a concurrent edit of the same attribute is very low and that in most cases it does not make sense (see section 6.2.6.1). The base principle behind is a total ordering of operations based on a state vector. If a remote operation arrives after the executions of a local operation which actually is positioned behind respective the total order, a undo/do/redo scheme is applied. In other words, the local operation is undone, then the remote one is executed, and afterward the local will be redone. Of course, this is done transparent to the user. Furthermore, similar to SXE it uses an ID-approach to identify, hence, every element can be identified directly and no path transformation is required like in the treeOPT algorithm.
However, there is more behind. For example, the total ordering may not always correspond to an ordering the operations actually can be performed, as a deduction the state vector may have to be modified or operations may have to be discarded. Furthermore, it provides fine-granular conflict resolution schemes for conflicting edits (corresponding to set-records in SXE). For more issues and features I refer to the related works.

The communication layer at first was based on RMI that resulted as disadvantageous. In a follow-up, the framework was ported to use XMPP by M. Voigt ([Voi09]). He besides resolved several performance issues by providing abbreviated communication tags and using a MUC to relay messages sent by the host (see 6.2.8). For more information have a look at his work.
Especially to mention is that he detected that the described first version of CEFX was erroneous and he fixed these issues. His version was actually supposed to be available at sourceforge but the uploaded version was erroneous and not complete. In November 2010 this version was still lost.

The third version is a porting to android by D. Hering, developed in 2009, as part of the concurrency control mechanism of the Mobilis platform[34] ([Her09]).

Especially to mention at this place is the chapter “7.6. Probleme und offene

[34] http://mobilisplatform.sourceforge.net/

Punkte” (problems and open issues). It states that, some CEFX-features are neither tested nor implemented (for example the update operation and with this everything connected to possible conflicts) and the correct execution of concurrent operations cannot be warranted. Furthermore, it was not possible to use/implement a late-join mechanism and there are uncaught exceptions with non well-formed XML documents.
Finally, he notes that a further development will be difficult. Most of the source code would be encapsulated in big classes without extension points. Besides, at several locations the framework would lack a clear separation between client and server.

As Hering already foreboded, trying to inspect the framework deeper resulted in a bigger challenge than expected. There was neither documentation about how to use it nor where or how to start. A look in the code revealed a some hard-cord debug output to standard-out as well as hard coded XMPP communication based on the Smack API. Thus, it would result a bit difficult to port it to Saros and to integrate it in the existing logging and session mechanic.

On the other hand, in this case it was easy to contact the authors and developers. I quickly gained answers and was pointed to entry points to the framework as well as offered to have a phone call whenever possible.

In a call with M. Voigt, he admitted that he needed more than three weeks to get into CEFX and even more to be able to extend it. He didn't know about the SXE XMPP extension but reading it he described it as an very interesting development. Certainly, the overhead would be high but the protocol will perfectly feasible for a whiteboard feature (maybe more than CEFX).

As a conclusion, the gathered information about this framework and the descriptions of its usage raised doubts whether it is a convenient solution for Saros at current stage. It would need improvement, some refactoring and maybe even more evaluation and scientific research.

6.5.3 Adaption of SXE

We already mentioned the shortcomings of SXE that are related to interconnections of the remove- and set-record. Having a closer inspection of the paper “Data Consistency for P2P Collaborative Editing” ([Molli et al.06]), the WOOT approach reveals a very interesting concept to solve these problems. Instead of removing a record completely of the tree, it instead could be marked as “invisible”. The extra data storage can be justified by the fact that most elements will be referenced by the command stack anyway. Please note, that also other approaches like CEFX do neither discard removed nodes (or rather the respective operation).

Remember the issues mentioned in 6.4.3. Of course, it would be possible to keep a history of deleted records separately, but it would require some exceptions when applying set-records to them. This had to be done because a concurrent edit (or conflict) might move them out of a deleted sub-tree and as a consequence would have to be recreated by sending a new-record instead of a set. Having “invisible” records would allow to apply set-records without extra effort by just changing the visibility.

But the more important point is that this feature would facilitate a undo-feature for SXE a lot. In chapter 7.4.2 you find an explanation that the absence of an undo facility in SXE actually causes more trouble that one might expect and that it also inhibits of keeping a history of deleted records, at least, at in the first prototype.

The problems I had with the proposed way to integrate something in Jupiter, treated problems with SXE, and this basic considerations were provided to the mailing list, triggering some feedback that I would not have expected.

6.5.4 The duct tape programmer

The head of the software engineering research group[35] that develops Saros, sent me an e-mail explaining that he does not want get involved deeply with the topic. He explained, an article he had sent to the mailing list was actually addressed to me: the assay “The Duct Tape Programmer”[36]. He had the impression I would look for the only correct solution and that I should put my energy in developing a solution and not in developing a concept for a solution.

[35] http://www.inf.fu-berlin.de/w/SE/WebHome
[36] http://www.joelonsoftware.com/items/2009/09/23.html

Further, we usually would not develop know-how, but would just apply it. Luckily, I could convince him that I was actually forcing problems that do not have a ready-to-use solution to be applied in one day.

Please also remember the doubts about maintaining the whiteboard as a separate module and finally I got another interesting statement from another adviser: As I once explained him that I frequently change my code to improve it he discouraged me: but if it works? In section 7.13 I give some justification for frequent changes and refactoring.

However, these reactions made me wonder because I had a completely different view of Saros. At this state, I admit, I thought of Saros as a project with a little bit too many parts just duct-taped together. One could suspect that some team members had suffered from a significant lack of time leading to hasty programming. I thought I would do a real favor in spending more time in planning than maybe some others.

To justify my lead impression (that is shared by the majority of the team members) I give some points.

The first impact was given by the immense lack documentation or comments in the central network or business logic like “TODO Add some information what needs to be done to add a new activityDataObject” and “TODO this method is currently not used. Probably they interfere with Jupiter”. Other comments are completely out of sync due to frequent use of the Eclipse refactoring mechanisms.
Further to mention is a central network class[37], messed up by invitation process specific code and doubled (supposedly copied) methods, searching the code revealed unnecessarily doubled code, unused methods and even unused hierarchies (see 4.5.1) and the overall structure of Saros is monolithic and pretty interconnected. Last but not least, it had and has several critical bugs, most to mention, the concurrency control was broken (concurrent edits in the same file lead to divergence, status October to December), although multi-driver support is one of the central Saros features.

Please mind, that quite some parts of Saros are very good though and straight

[37] XMPPTransmitter

forward. It is more likely that some problems in the center spread throughout the application and lead to confusion over and over again.

A special point to mention is that frequent refactoring, separation, modularization and structuring as well as new practical know-how for application sharing is not provided excessively. The above reaction almost gave the feeling that these points are seen as optional or maybe even as undesirable by team leaders because they might require additional effort. I would like to encourage with this paper to change this attitude.

6.5.4.1 Work in Saros

Team members in Saros rather do not work at home but in a room at the university specially provided for this purpose. During my time, agile methods like daily stand-up meetings were introduced and pair programming is highly encouraged. Unfortunately, in practice conducting PP fails on the circumstances: every student has to force his own subject matter that usually requires weeks or months to get a deeper understanding.

On the other hand, it is assessed to spend about 50% of the time in improving Saros and bug fixing, actually perfectly suitable for PP. But also this fails, because the bugs usually also require a significant amount of investigation and time.
In the best case, a second programmer (that didn't research) could just be an observer. Well, of course, it is the idea of PP to have an observer but this observer would be more likely daydreaming like described in [XPRef03]. The worst case for me would be a partner giving nice suggestions like a repeating of keep it simple but serialize it to a file and transmit it with the project, because he did not get too much involved with the problematic.

The point with the refactoring is that actually new students seemed to be highly motivated as they arrived. Recently, some team members have put a lot of time to improve the chat or to disentangle the invitation process but they had to stop at some time. They had to continue with their own thesis. Finally, they mainly had extracted parts that they actually needed.

There is no magic behind. Due to the lack of continuous refactoring like proposed by agile methods, central parts became messed up and a refactoring may seem to be very simple in the beginning but quickly you come to understand that if you change this, you have to do that. After extracting this, you have to get involved to the work of XYZ first and investigate how he added his solution to Saros. After improving his work you may continue with the part of member ABC and so on. At the end you may only get told, balance bug fixing, features, refactoring, and process improvement.

This is a good example of the concept of Extreme Programming (XP, [Beck00]). It defines 12 practices, beyond Pair Programming and Refactoring. Beck states that these practices are interdependent: if you do not follow all of them they may fail. Under the above context, it seems quite logical: if you do not have an understandable infrastructure, PP really may result difficult because too much research time is necessary for every single part. And this is research without gaining real knowledge, let's say, programmer skills – you cannot even write down what you understood.

Let me point out another principle of XP: the 40-hour-week. For some months, we had daily stand-up meetings of the Saros team. I think, this was quite useful as it resulted much easier to keep work work and home home. On the other hand, imagine working every day in a room that will not always offer an atmosphere for quite thinking time and research. When will you actually have time and calm to read a paper and perform deeper investigations?

Finally in your free time, in the evenings and the weekend. I would claim that some students still want to develop a proper scientific work at the end of there studies, but if you are supposed to spend 50% of your time in bug fixing and in working with a code base that does not yet allow very professional solutions this may result a little bit more difficult.

6.5.4.2 A possible reason for a lack of refactoring

Of course, the situation in Saros of course is special. Just take the principle of “Don't repeat yourself”. Even on wikipedia you find the commonly know “Rule of three”[38]:

[38] http://en.wikipedia.org/wiki/Rule_of_three_%28programming%29

- First time just do it.
- Second time, wince at duplication,
- Third time, refactor.

In a conventional project a single team member might experience this and initiate the refactor. The same happened to me and I guess it will apply to every team member working on his own part.

But the issue is slightly different with central Saros classes. Team members will change every half a year and it will be very likely that they only change the central logic once, hence, it may apply “just do it” from above. Naturally, else it might take a week to understand it.

There even is another aspect. If you have to extend one of those central classes, you may want to following the same coding style like everything else is done. This is a very good idea to avoid confusion and to avoid an onion anti-pattern – even if the class consists out of doubled methods. Therefore, the bugfix 9.5 added a new method to the XMPPTransmitter instead of providing a completely new method offering transparency. But I personally was not sure about this.

The first step to improve this situation is to highly encouraged and honor refactoring in Saros. These refactorings should be done in a team, an in my opinion with more than two members. A pair may always tend to waste time due to discussions as I experienced already when I had code reviews with the author of a patch.

Another step is to encourage modularization and abstraction. Abstraction of central parts would exactly offer what every team member needs: the ability to use it without copy and paste and without understanding everything. The time they need now to understand could be applied somewhere else.

Proper modularization, first of all, would allow team members to specialize to some clearly specified part (not interconnected scarily). Secondly, it would make it possible to define parts – in the best case own projects – that should not be modified but with profound reviewing. This could avoid to introduce bugs like 9.8.

I see it as questionable that I get the answer “no, keep it simple” if I explain that I want to keep some part transparent to Saros. First, sometimes this was the simplest and shortest solution. Second, sometimes it was clearly not the fastest solution regarding quite thinking time. But afterward everything was faster – this is what I was actually thinking about.

An experience I made in this work consists exactly in this point: a simple solution requires time and consideration. All-over-the-place programming can quickly provide solutions, but these are neither simple, nor their complexity may be obvious at first place (see chapter 7 and 8). So, I came to the conclusion that it is not convenient to encourage anyone in Saros to perform duct-taping.

Please note, that quickly after the discussion, some team members became proud of their duct tape programming style. This was even connected to the bug in the Saros concurrence control that redounds to my advantage now. Duct tape programming may lead to provide a quick fix for something, disregarding the time necessary to fix the bug that was created by the fix. Even worse, it can degenerate to quick an dirty solutions.

6.5.5 Conclusion

In this moment, I was not yet sure about the above point. Hence, I perceived that further research and consideration was undesirable for the moment. I had to decide to simply implement SXE as it is, disregarding mentioned issues. Please remember, this work follows the prototyping approach and as this, the advise I got actually completely fitted to my task without any doubt.

7 First SXE prototype

The principles of this XMPP Extension were already explained in 6.2 and will not be repeated here in detail. Although the protocol was improved in the latter state, the general architecture presented here corresponds already to the final version at the end of the work.

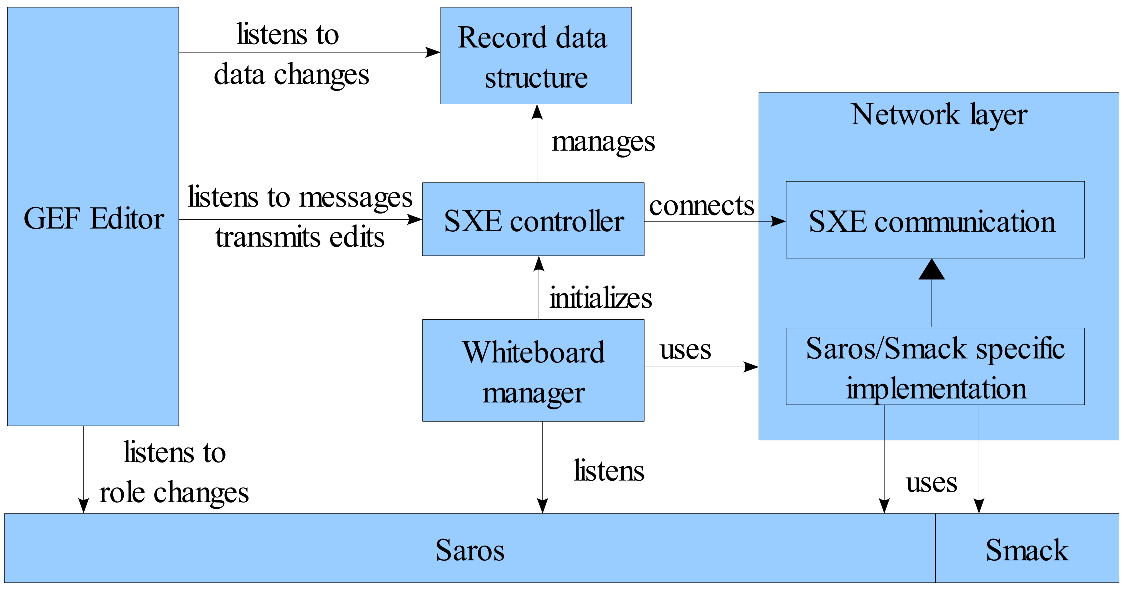


Figure 20 - overall structure of the plug-in

The SXE data structure and its network specific code (start synchronization and messaging) are independent from Saros, network and GUI. The editor is build on top of the protocol, thus, it can be seen as a graphical SXE editor. It is observer of the data structure and also listens to arrived messages, for example to update the editor content after applying the state message as invitee. Besides, it listens to role changes because the SXE protocol does not have any support for this.

The whiteboard manager is the only singleton of the application and the central connection between the concurrency control data structure, Saros and the view. It is loaded on start-up (defined in the plugin.xml) and listens to a Saros session start. Then it initializes the SXE session on the host side and enables invitation on peer sides. Likewise it listens to user joins to actuate the start synchronization and to role changes to relay them further.

7.1 The data structure

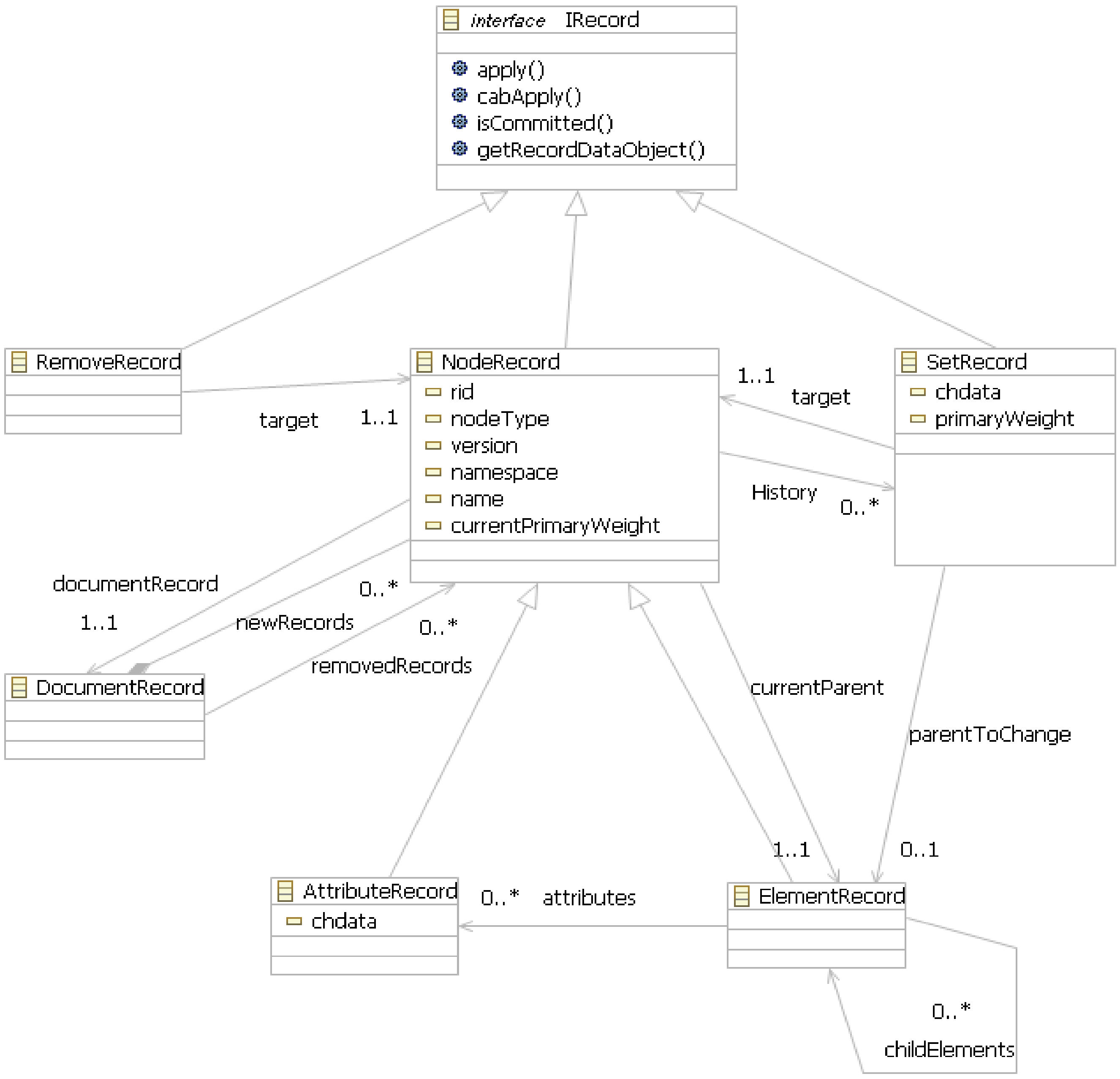


Figure 21 - Record data structure of SXE

Note, this figure omits operations and data types as well as irrelevant variables. The DocumentRecord does not actually keep a history of removed records but of their RIDs only (see 7.4.2).

The Node-, Attribute-, Element- and DocumentRecord correspond to the respective DOM node. The interface IRecord offers abstraction to the GUI and the network layer; remember that SXE messages consist of a set of records.

The record specific logic for applying is encapsulated in the apply method of every record. This keeps the controller class small and well-arranged.

The implementation uses a specialization of the Java TreeSet[39] and nodes implement the comparable interface so that child elements (and attributes respectively) have the correct order respectively their weight. If a record is added locally without setting a primary weight, this specialization will add a primary-weight higher than the last one to append this record to the end (as a consequence on top).

Attributes are maintained in a further subclass that maps names to the records. This way, an attribute can be easily accessed by its name. Besides it helps to solve the issue mentioned in 6.4.2.1.

SetRecords, as mentioned, build the history of a NodeRecord. Although ensured during applying, the history list class makes it impossible to add a record with a lower version than the last one. Local SetRecords do not need to provide a version – subsequently it will set during application. In other words, local edits can always apply.

Finally, the concept from Saros activities to maintain a business object and a serializable object was adopted. Thus, for every records exists a RecordDataObject that replaces references to other records by the respective RIDs and converts every variable to a string. Converting the serializable back to the application class requires some special logic: A parent or target record with the specified RID might not be available in the data structure. In this case, an exception is thrown that is handled by the controller.

7.2 The SXE controller class

The controller is the central part of an SXE session and maintains the document. It takes care about error handling during applying of records and in the following its methods are presented in detail:

- apply a local list of records

If locally successful, records will be sent to peers. Note, that a GEF Command

[39] namely NodeSet<T extends NodeRecord>

could transmit any kind of record to the document, even an invalid one due to an error. In those cases the local document remains unchanged and nothing is sent to peers.

- apply a remote SXE message

This is done by applying one RecordDataObject at a time. The controller proves first if the corresponding record is already contained in the document. If so it will be omitted. This is critical: if a set-record would be applied twice, the local version of the target would be out of sync.

Because of this, the records must store the sender. Only this way it is possible to distinguish whether two set-records are same. Note that different peers could send a same edit with a same version.

- queue non-causal-ready records to apply them when possible

For the background I refer to section 6.2.8.1. As mentioned, the conversion of the RecordDataObject may throw an exception because of a missing record with a specified RID. In this case, this object is queued, mapped to the missing ID. After a new record is inserted into the document, the controller checks if there are any queued records for its ID. If so, they are applied.

For set-records the controller has to check manually if their version overstates and they have to be queued. They are mapped to its target and checked for causal readiness again if the target version has increased.

- maintain state of the controller during session negotiation and therefore queue incoming messages during synchronization and apply them on finish

During the synchronization process, the invitee starts to receive remote edits and sets the controller to connecting state before sending the accept-state message. This way, no edits get lost.

Actually Saros implements a similar idea with the activities but trying it out it did not work, neither with the whiteboard nor with text edits (status October 2010). I tried to fix it but I completely failed to understand the desired flow of

events even after searching several hours and sending questions to the mailing list (see 6.3.6). ActivityDataObjects received during session negotiation are queued by the ActivitySequencer from some certain point and executed on starting the sequencer. But somehow Saros also uses a helper – the stop manager – more or less a locking feature while synchronizing the project. It was claimed it works and I should use it: I could serialize the XML to a file and transmitting it with the project. Maybe this suggestion was not serious.
Having a closer look to the stop manager revealed that it actually manually triggers a subset of managers to block editing from now. Well, first of all I did not understand why to plug my whiteboard manager there if the here provided solution works much better. The other point is that I saw this as critical: we already know that it is not trivial to intercept editing actions in Eclipse (see 6.3.3), so why depending on it.

- intercept set-records whose application would result in a non well-formed XML document

Remember the problem with circular relationships mentioned in 6.4.2.2.

At current state, the situation is checked during applying a set-record. An Exception is thrown and handled by the controller; the version of the target will be correct though. The record cannot be applied to make sure that really never a circular hierarchy is created – this could lead to infinite loops in the graphical editor leading to a StackOverflowException.
Those cases will lead to divergent versions of the shared XML document. The problem is less severe because of the undo feature. Certainly, it is not so nice that clients manually have to revert to a valid state but further research is necessary to provide a convenient solution. Most of my attempts just resulted in relaying the issue to another case or they even caused more severe issues.

7.3 Converting SVG attributes to GEF layout rectangle

I maintained the general principle from section 5.3.4: to have a specialized model classes for every graphical object. The point is that SVG has quite different ways to store the position and size for every element so that these values have to be converted to a GEF layout rectangle. A model class is the best place to do this conversion. The alternative would be to let the edit part create the layout rectangle out of the SVG values, while the commands would have to convert the other way around.

Furthermore these classes offer some figure specific facilities. For example, the polylines from freehand drawing have to be shifted and resized according to a new position and size provided by the user. Besides the model caches the point list of the polyline as this is considerably more efficient than parsing a string on every access.

Please note that using a specialization of the ElementRecord in the application also requires to deserialize remote records to their respective instances. Therefore the controller can be provided by a custom factory making this possible. The factory creates the SVG specific ElementRecords based on the tag-name.
It also has other usages, for example it helps to provide a custom feedback figure for element creation. By default, GEF only visualizes a selection or target rectangle. This factory together with the edit part factory now offers a transparent way to create a custom figure for a tag name – finally, for example the ellipse feedback will be a semi-transparent ellipse.

7.3.1 Remove underlying DOM implementation

As already mentioned in 5.4, the underlying DOM implementation was not accessed anymore by the application, now, switching to SXE this even became more obvious.

Indeed, it would be possible to provide a wrapper-DOM implementation that encapsulates the records and then encapsulating the DOM elements in the models but it is quite evident that there is nothing to gain. Another possibility for custom usage of SXE would be to use the DOM Level 2 Event model in the way that the records listen to DOM changes.

Lastly, it is actually quite useful to have fine-granular access control. Sometimes there are values that are semantically related, for example the x and y value. If one user changes x and another y, the object would appear at a place where nobody wanted it. Therefore an empty set-record can be sent for the complement (instead of leaving it unchanged): if a conflict occurs it will revert both values or none. On the other hand, respective the width and height this is more likely neither necessary nor desirable.

7.4 Commands on top of the data structure

The GEF commands described in 5.3.3 had to be modified to transmit records now instead of performing changes directly on the model. An abstract SXE specific command provides the infrastructure – instead of implementing execute, undo and redo, the subclasses have to provide the records, undo- or redo-records, that will subsequently be passed to the controller.

7.4.1 in a distributed environment

Working distributively – remote clients may change the underlying data structure without depending on the GEF commands – may also have some undesired consequences.

For example, a remote record may cause that a local undo is not possible anymore, in other words, this action should check enabling and in the following be disabled. By default, this is not done (only when removing a selected item) because no local command was executed. As a consequence, the editor listens to applied remote messages (and to view updates in general) to force an update of the provided actions.

The redo-feature required some more investigation. It resulted that this action is enabled by the command stack as long as the last command was an undo, independently whether the respective redo will be executable or not. This is useful if working locally. Consequently, a flush redo is only a private method in the stack, non accessible from outside. But in a distributed environment the data structure may have been changed concurrently to make the redo execution impossible.

The solution consists of a custom command stack that checks if the last view update was caused by a peer. If so, it checks the redo executability and disables redo if required.

7.4.2 undo/redo support

To enable the undo and redo support using SXE, the undo-create was mapped to a remove and vice versa, a undo-set could be mapped to a new set-record that sets the values to the previous ones. However, the undo of deletion (and redo of creation respectively) on the other hand confronted me with quite some difficulties because you cannot simply create a completely new record in this case: this would break the references in the command stack.

The problem was solved by a re-create mechanisms that changes the RID of the record and resets the history. In this way the reference could be maintained locally but the peers received a completely new record.

This idea has some drawbacks though: We already notices in 6.2.8, that a history of deleted records would be quite useful as it could solve some conflicts or even avoid them completely. Sadly, this is now not possible, as the references may have been reused by the command stack. The only thing we can do is to keep a history of the RIDs of deleted records – in the case of a conflict we can distinguish whether it is a non causal ready record to be queued or whether we should answer with a remove record.

Furthermore, it confused the issue of the GEF commands somewhat. Suddenly, the undo-remove had to iterate throughout a deleted sub-tree, recreating and recomposing the tree. To make this possible, additional or modified infrastructure was needed, for example, during a second undo create a previous remove-record still maintained the same reference but it already was executed. In other words, a remove-record whose target was not removed from the DOM tree did not correspond to a non-executed one anymore.

7.4.3 Only modify top-most records

Selection a set of nested elements in GEF will transmit the selection as list in an arbitrary order. Particularly, child elements are selected as well and may be before or after their parent. A feature that may rise some issues:

It is not so nice to insert a child to the document prior to its parent. Although this is locally allowed and does not lead to divergence on the peer sided due to the causal-readiness mechanisms, it is not a proper style. In the context of deletion, removing a parent before removing its child, this behavior could even rise up some issues.

Moreover, selecting parents together with its nested children results in unintended behavior when moving or resizing. One would expect to move the parent but leaving the children on its relative position to its parent but this was not given.

Therefore, I provided a utility list-like class[40], that separates inserted records in top-most (or root) records and records that are children of the the top-most records (sub-tree records). The commands now are only executed on the mentioned root records. This also applies to the remove-records because of the situation mentioned in 6.4.3 and its solution. This was to answer with a remove if another peer concurrently moved anything out of the deleted sub-tree. So it is not necessary anymore to remove children first as specified by SXE and applying a remove was slightly modified to mark all descendants as removed, too.

7.4.4 Disable a set-parent if the result is invalid

An undo or redo of a set-record that changes a parent might end up in a not well-formed XML. Those kind of set-records are only created and maintained by the ElementRecordAddCommand. If the mentioned situation is given, the command is disabled.

7.5 Notification of data structure observers

In the graphical context of SXE, ElementRecords correspond to the models that have to notify observers. Attributes on the other hand are features of these models and thus should notify their parent only. Further to mention, the set-record field primary-weight and parent are actually features of the parent, too, respective contained children and child order. They are the only mutable fields of an ElementRecord. As a consequence, during application of any record, only the parent has to be notified (or the parent of the target respectively).

[40] HierarchicalRecordSet

It resulted, if notifying the GEF edit part about every single change in the data structure, this may have undesirable consequences. Just imaging to first update the view after the x-value change and consecutively again after the y-value. Working on a local copy alone did not raise significant issues yet but together with remote messages it quickly made a figure seem to jump around or to perform an operation in steps only.

As a solution, notifications were cached by the parent ElementRecords. The controller listens to these caches and informs observers of the data structure (the GEF editor) only after applying a whole list of records.

This concept resulted as very useful and it also opened room to some GEF issues. If moving or deleting a selected collection of records, the editor creates one command for each selected item. These commands are composed in a compound command, but internally executed one after the other arising the same “stepping” issue like mentioned above. Another drawback was that in this way the edits were also send to the peers one by one. In other words, modifying 100 selected elements resulted in 100 single messages.

A transparent solution could be provided by extending the custom command stack: SXE specific commands (in our case all commands are SXE specific) are compound in a custom way that transmits there change records in one list to the controller.

7.5.1 Notify about view update in progress

During my work it resulted that the GEF editor could require some notification about the view update in progress (notify about notification). The Background is the following:

After a certain amount of elements GEF performance towards selection and executing actions on selected items slows down a lot. The main reason is that it recalculates action enabling (undo, redo, copy and paste) and child edit parts on every change of the selection (in our case on every notify to an ElementRecord). Let's say, it is especially expensive if the performed edit removes any selected item – but it is impossible to remove anything without selecting it. After creating 5000 objects, trying to delete them took 3 to 15 minutes in the default implementation.

GEF received one remove notify after the other, after recalculating one figure and removing the selection handle, it will count to 4999 to recalculate layout, child figures, actions …, than it handles the next notification counting to 4998 etc.

This feature or issue may have its origin that GEF is thought as a diagram editor with short commands. It again results in a view update performed step by step. Meanwhile action enabling was flickering and the GUI blocked. This is more than undesirable, especially after messing up the whiteboard with 5000 lines (easily achievable when working in a session) you may really want to delete everything more than ever before.

The notification about the view update in progress provided a perfectly feasible solution to let the GEF editor use a custom viewer implementation that ignores selection changes in this state. It forces an update of actions at the end only. Note, that the edit parts still have to be notified one after the other because the only alternative would be to force a complete refresh of the editor. But this might decrease the performance for small edits significantly and hence is undesirable.

After implementing this solution the deletion of 5000 elements was performed in less than 15 seconds, in other words bearable for a user.

7.6 RecordDataObject serialization

Saros does not serialize manually but uses XStream for object serialization to XML (see 4.5.2). At first, I followed this approach but this confronted me with more difficulties than expected.

The first issue occurred associated with the SXE tag names of records. The new-record may correspond to an ElementRecord or an AttributeRecord. It required some time to find out that this would made it impossible to simply use annotations to let XStream do the conversion to XML. The alternative would be to provide a custom XStream-converter that uses a parser, namely the

HierarchicalStreamReader, more or less the same like a SAX parser. The only feedback I got was that we don't do manual serialization, to keep it simple and forget about the protocol. Points that I accepted.

However, the other problem on the other hand even required more investigation. As demonstrated in section 7.3, remote objects have to be serialized to an application specific implementation using a factory. Thus, my XStream converter would require the factory as parameter; something that cannot be done due to the static nature of annotations. Further searching and a discussion on the XStream user mailing list revealed that I actually cannot use annotations in those cases but I could manually use the XStream infrastructure to do those conversions. In other words, I could re-implement the payload feature of the activities in an exceptional way for my purpose.

Unfortunately, at this stage nobody could give any feedback about the problems I was forcing. Only at the very end of my work I got aware that I misunderstood a part of the activities infrastructure about how it applies the conversion from serializable variables to the business variable. This is actually done manually: Maybe there had been room to provide a simpler solution with XStream than above using RuntimeExceptions or doing detailed conversions at a latter state in the SXE logic only. But still this would have required a whiteboard specific exception in the activities logic.

However, at this stage I had spent about a week to learn the XStream API without finding a way for a feasible solution and I skipped it. Instead I implemented a working conversion in few hours using a SAX parser (outgoing) and the XMPP pull parser of the Smack packet extension (incoming). This step actually will result as very useful afterward for start synchronization.

Then, the converted string could be plugged in the activity from chapter 4 and was sent over the network in a base64-encoding. Thus, the actual appearance had almost nothing to do with the protocol described in 6.2.

7.7 XML tags and constants

Frequently, type variables of records or messages in SXE correspond to some XML tag name. Therefore I provided Java enumerations for the type variables as they do not allow illegal values, can have behavior and allow looping over available values. The constants were extended by a constructor to transmit the corresponding XML tag. This way the implementation offers a simple way for tag localization, also tag externalization could be done in a minute. Every enum class offers an utility method for deserialization that enables to achieve the constant from a tag name.

Records in SXE do only have attributes. Therefore they actually can be stored in a map only, implemented in the RecordDataObject. It maps a RecordEntry enum constant to a string. Thus, changes in the record structure do not require a change in the abstract data object class or the serialization in the network, only in the methods to convert it. Furthermore serialization/deserialization with a parser can be done in a loop, making it simple and transparent.

7.8 Start synchronization

The start synchronization described in section 6.2.7 is actually very simple. Not so easy is the question where to add in Saros. The common way was just to hard code the chain of messages inside the XMPPTransmitter and the invitation process classes. At this stage, another team member was working on a refactoring of the invitation process and it seemed to be very inadvisable to mess it up more meanwhile.

Therefore, a first prototyping way to achieve a working whiteboard in the session was to use the string activities to carry out the steps of the SXE protocol. Unfortunately, it resulted that the refactoring would take much more effort than expected and it was not completely finished until the end of this work. Instead the team came to the agreement that this part of Saros at first will need rework and proper structuring. So it did not make sense to wait.

Luckily, searching the code I found a proper position to initiate the synchronization process: the user join notification on the host side. This may be refactored in the future to the new session negotiation and invitation structure.

For this part, finally it was necessary to access the underlying Smack API directly as is done by all other components of Saros. This justifies also the implementation of the network adapter in the next section as well as the SXE specific packet extension.

7.9 Network adapter

The network layer interface in the implementation only need three methods: sendAndAwait() for the chain of events during start synchronization, send() to send bunches of records in a session and later installRecordReceiver() to connect the controller as record receiver to the network layer.

The Saros and Smack specific implementation of sendAndAwait() encapsulates what is duplicated for every message in the invitation process: Install a collector, send the message, and wait for the answer.

Unfortunately, in the existing invitation process the chain is not always kept so clear (sometimes due to changes over the years) and evaluating whether this is applicable for Saros, too, revealed that a lot of almost copied methods differ in small details making it difficult. This will also justify that the refactoring in progress is still pending.

Please also remember the principle idea of prototyping: changes in a latter state of an application may become quite time intensive.

As I needed this infrastructure for the start and as SXE does not depend on any facility given by Saros activities, it could be directly plugged on the network layer. This way, the delay mentioned in 6.2.8 does not trouble the whiteboard implementation.

Furthermore I would advise to send the messages always by bytestreams. Given SOCKS5, this will be a fast peer-to-peer TCP connection or a connection using the server as proxy.

IBB, of course, offers the bytestream by encapsulating the data in XMPP messages. But the overhead of plugging a packet extension into an IBB stanza instead of a standard packet is minimal.

Finally, bytestreams offer reliability and maintain the message order (although not needed for SXE).

7.10 Usage of a Multi-User Chat (MUC)

As already described in section 6.2.8, the usage of a MUC can have a couple of advantages. Of course, peer-to-peer (P2P) using bytestreams might be preferable but can also result as very tricky. Especially, if a client has to correct a situation because of the remove mechanic (for example see 6.2.3), the same problem may have emerged on other peers, too, leading to a flooding of correcting records.

At this stage, a team member started a refactoring of the existing chat feature in Saros that was know for frequently causing trouble. I proposed, to extend the network layer by a MUC-feature but unfortunately this concept was not taken over because of a significant lack of time. Sadly the result of the refactoring did not really improve the visible effects, instead it seemed that it did not work anymore when using the Openfire[41] XMPP server. Maybe before another server was used to establish the room but I could not investigate further.

However, this experience and other issues mentioned by the refactorer, showed that the whiteboard should not depend on a MUC alone because it may be troublesome. In other words, the implemented P2P solution is pretty cool.

7.11Code review and code walkthroughs

As part of the Quality Assurance, we accomplished a short code walkthough respective IEEE 1028. Therefore I provided four documents presenting the architectural parts of the whiteboard as well as selected parts of the code in

[41] http://www.igniterealtime.org/projects/openfire/

detail. Two team members at a time were divided into reading them.

The walkthrough meeting resulted in acceptance of the provided solution.

There were some minor advises to extract the message class to be an independent one and to refactor the deserialization from the Smack packet extension to an utility class.

7.12 Mapping Saros session roles to the whiteboard

One of the last features I added to the whiteboard was to promote Saros role changes to the editor and extending it by a disabling facility. Therefore the editor listens to role changes to the whiteboard manager.

As already explained, it is not so easy in Eclipse to intercept all changes. Luckily in GEF, some investigation revealed quite feasible methods to make any kind of editing impossible. The basic concept is to extend the editor by a disabling or locking feature.

Drawing and any kind of direct graphical editing or tool usage can be disabled by providing a custom EditDomain class, that does not allow edits to the graphical viewer. The implementation does allow the panning tool though, as well as mouse wheel zooming. Selection should remain disabled as a selected item could be changed by other means, for example by an Eclipse properties view.

In contrast to the whole Eclipse platform, the GEF editor maintains its own action registry. Not being registered here, an action cannot be executed because it is accessed by its ID. As a consequence I provided a custom registry that has a disabling feature, if disabled, every action will be disabled as well.

7.13 Design and refactoring

One thing I really like of the extreme programming approach is that it encourages refactoring over and over again [XPRules]. Certainly, I'm not a domain expert for refactoring but some simple rules helped me to provide small and understandable classes. It was not always necessary to refactor, of course, usually the principles lead to cleaner solutions from the start or to more investigation. These rules can be find in the one or other version in [CodeSmell].

- Separation of concerns

This can be described as the key concept of object oriented design, probably coined by E. W. Dijkstra in 1974 by his paper "On the role of scientific thought" [42]. An application should separate its features with as little overlapping as possible.

This concept highly influenced my decision to keep the view, the network and the data structure as independent as possible. For example, if the view needed notification from any other parts, these were extended by listeners instead of directly accessing each other.

Another aspect I already mentioned in the beginning: a whiteboard should be kept as separated as possible – it only accesses the Saros session and the network layer.

- Don't repeat yourself

Is one of the most well-known principles. If I noticed too much duplication while stepping forward I knew I had to refactor. If happening in the same class like during the application of a SetRecord that can conflict, I extracted the methods to update the values. If in different classes I introduced hierarchies tailoring up the methods. Examples are the AbstractElementRecordCreateCommand or the AbstractRecord.

- Method is too big

If a method became longer than to fit at one page or if it handled different concerns duct-taped together, I extracted a protected method like in the paste command.

[42] http://www.cs.utexas.edu/users/EWD/transcriptions/EWD04xx/EWD447.html

- Class too big

I introduced (abstract) superclasses for the whiteboard manager and SXE controller to avoid having to scroll over the 100 lines for listeners and notifications every time. As well, the notification cache of the ElementRecord is kept in a delegating class.

- Class too complicated

The locking of the editor (see 7.12) is not the most trivial task without understanding GEF too well and it requires some extra explanation. Therefore, instead of plugging it inside the WhiteboardEditor class, it could be realized as a super class extending the GEF graphical editor by the disabling feature.

The actual concept behind can be described by consecutively adding features by the hierarchy and is found all over the place in GEF. The similar idea caused me to provide an extra edit policy for the freehand drawing feature instead of maintaining it together with the application specific commands.

Although I claim, that there is still room for a lot of improvements in the implementation, following this guideline helped a lot to maintain and extend the whiteboard iteratively. An example is given by a final improvement of the underlying protocol and data structure.

8 Improvement of the SXE protocol and final Version

Issues with the recreate mechanism and the concept forcing an answer remove-record because of moving a record out of a deleted hierarchy kept me quite busy in the last weeks before finishing the work. It did not only require very specialized test cases but it also made testing difficult because same was not actually same anymore. As a solution I provided a testing infrastructure, mocking the network, described in section 8.2. Also the P2P functionality suddenly became questionable.

In general, the concept of recreation somehow was clear but it always resulted as difficult to explain the issues with this concept. My personal impression of the code it caused even corresponded to code smell – it cannot be good to change the ID of an object. It reduces traceability.

Finally, I stepped back a moment to revisit everything. In the following, a perfectly feasible solution was found, reconsidering the mentioned idea in 6.5.3 and following the approach of the WOOT algorithm: Replace the remove and recreate feature by a set-visible.

8.1 Visible mechanic and history of deleted records

As a consequence, the node-record was extended by a boolean variable "*visible"* as a further mutable field and the set-record could set this accordingly. In other words, the whole remove and recreate mechanic could be mapped to the set-record application with very little effort.

As mentioned, the additional memory overhead can be justified by the command stack that is anyway likely to store the references as soon as a peer ever modified it. Furthermore, the invisible records exactly correspond to the history of deleted records that I had already desired before. The only difference is, that remove and undo will not result in a new record on peer sides. In other words, it even may save memory in sessions with a lot of undo and redo usage.

The getChildElements() and getAttributes() methods of the ElementRecords were made protected for the internal logic and publicly replaced by getVisibleChildElements() and getVisibleAttributes(). Further steps added the new visible-field to the RecordDataObjects and the methods for conversion.

The recreate mechanic could be removed, therefore the NodeRecord offered convenient methods to get a remove or recreate SetRecord, setting the visibility respectively.

In the editor, I only had to replace the existing lines to for recreation and removing to a call to newly created methods. The abstract super class of all models now returned the visible children instead of the real children.

Commands before used a isCommitted() method to distinguish whether a record is actually part of the document or is not added yet or anymore. A command had to be disabled, for example, if a new parent due to the execution (or undo respectively) is not part of the data structure. This was replaced by the new method isPartOfVisibleDocument() that checks whether the node is committed and that it is visible as well as all of its ancestors.

As may be noticed, due to a quite tidy architecture this change could be performed in less than an hour and the result is remarkable. A set of test sessions with another client took place without any unknown issue. The cases described in 6.4.3 could be simulated by using undo and redo and did not cause an issue or divergence.

Certainly, this is not a proof of correctness but you may notice that the problem is reduced significantly in complexity. The set-record mechanic is well proved and this change enables undo and redo in a better way than before. An effect of a redo-creation will result in the same reference in the command stack for all peers. This also opens the door for more filed features like selective-undo. To be honest, this already would be possible but the user interface is missing.

8.2 Quality assurance

Apart from the usage of open standards (SXE) and mature APIs (GEF) the walkthrough in section 7.11 remained the only completely accomplished mechanisms for Quality assurance. This can be justified at first place by the prototype characteristic of this work. It was impossible to provide test cases from the start or to apply even test-driven development because neither any structure nor any component could be fixed. On the other hand, manual testing was performed frequently. This can be justified in the context of Batik, where I usually forced visual problems that no test-case in the world could have detected.

Please note, the last days of the work I spent additional time to fix the bug in the Saros concurrency control (see 9.8). Furthermore, I finally achieved to improve the protocol for collaborative XML editing.

On the other hand, as mentioned before, I provided a testing infrastructure for the SXE part. For testing, the SXENetworkMock class can be used to create an arbitrary amount of MockedSXETransmitter instances to be passed to different SXE controllers.

These classes resulted as an appropriate solution to provide test cases for the recreate mechanic that made sense. Unfortunately, these cases are obsolete now. However, until the presentation of this work in January 2010, I will provide an extensive test coverage for this modified SXE protocol implementation.

Finally, in the context of agile software development, refactoring can also be seen as part of Quality Assurance[43].

9 Miscellaneous

As already mentioned before, bug fixing and refactoring is also part of the work in the Saros team. In this chapter I present a selection of done work.

9.1 Network refactoring, IBB and SOCKS5

After the batik prototype, a Saros specific implementation of the Smack API had been finished by H. Staib [Sta10]. He had extracted the internal SOCKS5[44] bytestream feature as well as the IBB[45] fallback from the internal API to make it accessible to Saros.

SOCKS5 is an XMPP Extension that enables clients to establish a direct TCP connection. If this fails due to NAT problems, the XMPP server may act as proxy if it is configured correctly.

IBB, In-Band bytestreams, is another extension that transparently offers to stream data. Behind the data is encapsulated in XMPP stanzas relayed by the XMPP servers. Without a special server configuration denying IBB stanzas, an IBB bytestream should always be possible. Saros depends on bytestreams

[43] more precisely as quality defect removal: http://www.scribd.com/doc/20071647/Agile-Software-Development-Quality-Assurance
[44] http://xmpp.org/extensions/xep-0065.html
[45] http://xmpp.org/extensions/xep-0047.html

due to the invitation process when the whole project may have to be transmitted to peers.

Before, Saros used the Jingle File Transfer[46] implementation of the Smack API that did not always result in a working bytestream.

As a consequence, these newly available features had to be integrated in the Saros network layer, replacing the old usage of Jingle File Transfer. For more details and references I refer to the work of H. Staib.

9.1.1 Fixing bugs and introducing hierarchies

In the beginning I could help to find a bug in the newly create IBB manager. The usage of two synchronized methods in the IBB stream manager could lead to a deadlock under certain circumstances.

Furthermore I encouraged to provide interfaces for IBB and SOCKS5 because both the respective managers did actually not differ in method names.

9.1.2 Introducing transport concept in the DataTransferManager

The existing DataTransferManager class was modified in the way to maintain a list of transport methods. Every transport singleton offers methods for initialization and establishing a bytestream connection to a peer. One transport after the other was tried to establish a connection until accomplishing a working one. Hence, if SOCKS5 failed, IBB was intended.

The introduction of the interfaces above made it possible to maintain most of the logic in an abstract BytestreamTransport class, at first, subclasses only had to provide the manager.

The transports are initialized after being connected to the XMPP server, listening to incoming requests to establish a bytestream.

9.1.3 Adapting the existing Jingle connections

In a next step, the already existing infrastructure for Jingle connections based

[46] http://xmpp.org/extensions/xep-0166.html and http://xmpp.org/extensions/xep-0234.html

on binary channels using the Google Protocol buffers[47] had to be adapted to IBB and SOCKS5. Therefore, the binary channel was modified to accept a bytestream session – the interface returned by the bytestream manager implementations – instead of a Java socket. The different jingle connection steps and connection classes could be extracted and reused to create a general binary channel connection, corresponding to the old jingle connection expected by the DataTransferManager. This way, other parts of the network layer could almost be maintained as they were. However, Jingle had a much more complicated mechanic and a lot of code could be removed though.

9.1.4 The concrete transport subclasses

The concrete IBB transport remained very simple as it did not require any special treatment.

SOCKS5 on the other hand is specified to be single directed only. However, in the case of a direct connection, this does not matter. But on mediated connections using the server as proxy it counts. Although most XMPP servers provide a bidirectional connection, some of them don't, for example Openfire.

The solution was not so trivial, as a comparison, the SOCKS5 transport class became quite longer than the whiteboard controller.

At first, the general concept is the following. After establishing a connection on peer A, check, whether it is direct, than use it. Else, check whether it is bidirectional by sending a byte over the stream in a round-trip. If it succeeds, use the stream, else wait for the peer to connect.

If peer B arrives the connection request and it is not direct, it will test for bidirectionally at the same state. If it fails, it establishes a second bytestream connection to peer A. Afterward he can put both unidirectional streams in a bidirectional one.

9.1.4.1 Improvement of the SOCK5 transport

It emerged that we did not always have the same kind of SOCKS5

[47] http://code.google.com/p/protobuf/

bytestreams (mediated or direct) depending on the platform (Windows, Linux, Mac) and the peer that starts to establish a connection. Especially a direct connection failed when trying to connect from Linux or MAC. The reason was that the new Smack SOCKS5 implementation was not able do connect to their SOCKS5 proxies.

To achieve the best connection possible, the mechanisms was slightly changed. After receiving a request to establish the bytestream, a peer will immediately start to establish a bytestream in the other direction. In this way, both clients will have two connections established and can choose the direct one if one of them got mediated only. The other connection can be discarded.

9.1.5 Making bytestreams configurable

Another fix had to be introduced due to the port the local SOCKS5 proxy uses. If the desired port was occupied it failed at first and we added to choose any port higher than the desired one in those cases.

The port to use, as well as to try other ports or whether to use SOCKS5 at all (or prefer transfer by chat), finally, had to be made configurable in the Saros advances preference page. The preferences did not apply directly but on a reconnect only (it would not make sense to change the port while using the stream).

9.1.6 Problems

Quite confusing were problems due to the XMPP server configurations. Openfire requires to set its property “xmpp.proxy.transfer.required” to false. Also the university XMPP Server, ejabberd, saros-con.imp.fu-berlin.de, was configured wrong in the beginning.

Other issues were caused for example in the SOCKS5 transport as it was quite difficult to keep track of the chain of events, other were connected to the Smack roster cache and the IBB connection.

The whole refactoring and bug fixing process took about a month.

9.2 Closing a session doesn't cancel running invitations

If the host initiated a new session inviting the peers and subsequently immediately closes the session, the outgoing invitation processes were not stopped. This could be fixed by an listener that interrupts these tasks respectively.

By the way I fixed a pop-up interrupting the host's work if a peer cancels the invitation. Therefore the existing usage of the Job API was fixed to return a correct status object.

9.3 Fixing invitation delay in Linux systems

In the SarosPacketCollector, synchronized methods together with the usage of a linked blocking queue did not allow to add a new packet as long as another client was waiting. This problem was worked around by a “force wait” boolean that retried to receive a packet from the collector even after the timeout elapsed.

Making the mentioned methods synchronized did not make sense because of using a thread safe linked blocking queue. By the way some unused methods could be removed from the collector.

The effect was considerable: on Linux platforms, the invitation took disproportional longer than it should. This patch fixed it.

9.4 Removing the bin folder from the class path

An accidental commit had added the bin-folder to the class path. That is why the debugger sometimes ended up in a source-not-found editor. Furthermore this caused weird access-restriction error warnings in the separate whiteboard project.

9.5 Extending the Invitation process by an acknowledgment

In the invitation process, the host used to send the the invitation to the peer and afterward he awaited the file list request (without a real timeout, see SarosPacketCollector, section 9.3). On the peer side instead, the dialog opened. If the Dialog was canceled, the host received the cancel message, else the file list request.

However, some XMPP servers do not implement the service discovery correctly. That's why in Saros it is possible to invite a peer without Saros. In this cases the host's outgoing invitation job gets stuck, awaiting the request for the file list or a cancel.

To avoid this, the patch made the peer now acknowledge the invitation. Therefore I studied the invitation process and the XMPPTransmitter. For every message in the invitation, the transmitter maintains an extension provider singleton and provides two or more methods. One to send the message and one to receive it, and sometimes one more for the collector. The idea is clear: you have to start receiving before you send, that's why the collector may be required. But one may imagine that this makes the transmitter a little bit bigger.

However, the new message for invitation acknowledgment was added in exactly the same way to maintain the same code style. First, I was supposed to fix the bug in a specified week. Secondly, if anyone ever works with these parts again, he will understand all methods including the new invitation acknowledging. A custom solution might lead to confusion. A quick refactoring on the other hand is not possible, as some of these methods have a slightly different implementation, tailored to only one purpose, although in general same.

9.6 Sending activities by bytestreams if P2P

By default, Saros send activities as a standard XMPP message. Only if they exceeded the maximum packet size, the bytestreams were used. This was changed in the way, that they are sent always by bytestream if the connection is P2P. If you imagine the round-trips and the server load presented in 6.2.8, this was already overdue.

9.7 Provide transparency about the sending method

The whiteboard could only access the sending methods of the XMPP transmitter that sent messages as Smack packets. Having a closer look revealed, that the XMPPTransmitter actually could provide abstraction to the sending method.

9.7.1 Saros activities delay

Remember the Saros timer for activities noted in section 6.2.8. Actually, the reason to introduce the above timer in Saros was not the network load but the possibility of being disconnected from the XMPP server due to the amount of sent messages. In fact, XEP-205 mentions potential solution for server implementations against Denial of Service attacks, beyond to limit the number of XML stanzas that a connected client may send to different recipients in a given time period. However, in the recommendations this more likely includes to forbid In-Band bytestream stanzas instead of counting single messages.

Saros depends essential on bytestream: without them, the invitation is not possible. In other words, if it is neither possible to establish a SOCKS5 nor an IBB bytestream, a session cannot be established. This makes it questionable why we should not use a bytestream for all Saros communication.

At the end of my work there was an attempt of Kent Beck (one of the creators of Extreme programming, [Beck00]) to try out Saros. He noticed the mentioned delay and explained that this makes Saros less than perfectly for them. This also caused some reconsideration in the team. Given the above assumption of potential disconnects from the server, shows that the timer only makes sense if sending messages by the XMPP chat. As consequence we agreement to tailor the timer down to the network layer and remove it from the activity sequencer.

Another point is, that the mentioned timer actually does not queue all messages (other parts like request for activity use the XMPPTransmitter directly). If adding the timer to the XMPPTransmitter (plain XMPP messages), the delay would influence involuntarily other parts of Saros. So it would be a good idea to send all messages transparent to the clients, internally using a bytestream if possible.

9.7.2 Two packet listeners for activities

While sending activities was encapsulated in the XMPPTransmiter, the way to receive a bytestreams in Saros resulted as implemented quite complicated.

First, the DataTransferManager transmitted an IncomingTransferObject to the XMPPReceiver – this object can be used later on to receive the stream. Afterward, the receiver transformed it to a Smack PacketExtension, plugged in a Smack packet this was transmitted to the listeners, for example the ActivityHandler. This in turn had an extra listener for transfer objects.

The listener extracted the passed extension, using it to finally receive the bytestream data and transforming it using XStream. Of course, the handler also had another listener that received standard chat messages only.

9.7.3 Implementation

To enable a central entry point to the network layer, Smack PacketExtension are the best way to go, compatible with all existing parts. File transfer types (or rather bytestream message types) used an enum, an unnecessary static restriction, and this was changed to use strings only. This way, the name of a packet extension could be mapped to the name of the file transfer type. Furthermore the transfer description was extended by the respective namespace value.

On the way, this refactoring fixed some issues that could lead to errors due to false usage.

It was possible to register multiple listeners for IncomingTransferObject-packets but if using a second one to download the bytestream, an exception was thrown. As a solution the XMPPReceiver will only let it be processed by the first listener and not take any more action.

If there is no listener for an IncomingTransferObject registered, it checks if there is a provider for the name and namespace registered in the Smack API.

If so, the data can be received by the transfer object and parsed to the PacketExtension using the provider. Before, the transformation was done by XStream using the data as string. The extension – plugged in a packet – can finally be passed to the listeners.

As a consequence, the ActivityHandler now only needs one listener to Smack packets and does not need to take any special action respective the transfer method. Furthermore, the XMPPTransmitter now offers a method to send an extension with transparency. A default transfer description is used corresponding to the previous activity transfer description. Hence, any other XMPP message can now be sent and received, internally by a bytestream if applicable.

Another internal fix achieved during this refactoring concerned a Saros specific implementation of XStreams XPPReader. The implementation used a hack to make XStream think it just started parsing (although it already may have parsed a couple of extensions before). Unfortunately, this hack caused that it did not work anymore if it REALLY just started parsing.

At the end of my work, the review was still pending.

9.8 Fixing the broken concurrency control

The problem was introduced by a new dispatcher in the SarosSession for the SVN operations. As a consequence a bunch of received activities was not executed consecutively anymore in the SWT thread and local actions could be executed in between - milliseconds after the transformation but before the execution of the transformation.

In other words, the transformation of Jupiter from the earlier state did not anymore match to the current state.

10 Conclusion

The goal of this work was to developed a whiteboard feature for Saros, an Eclipse plug-in for Distributed Party Programming. The process was lead by the prototyping approach to evaluate requirements, external components and design decisions.

A prototype for the SVG rendering library Batik revealed, that external components based on Java Swing may not feasible to be integrated in Eclipse for graphical interaction at the current stage. Swing is the primary Java GUI widget toolkit while Eclipse uses its own GUI toolkit called SWT. Although SWT offers a possibility to wrap primary Java GUI elements, this results in a double threaded behavior leading to significant performance problems and unresolvable graphical errors, presumably inside the underlying rendering libraries OpenGL or DirectX respectively.

A second prototype was based on the Graphical Editing Framework, GEF, a mature Eclipse Project with 10 years of active development. It could be verified to be perfectly dedicated for above purposes. Besides, it was confirmed by team members that a simple freehand drawing tool cannot be enough for a distributed whiteboard, different shapes, selection and modification as well as nesting of elements are useful features.

In favor of the implementation of the real-time collaborative drawing support, a closer inspection of the current state of research for collaborative XML editing became necessary. It was set out that a problem free and approved solution is not yet publicly available.

The first implementation of the experimental XMPP Extension Shared XML Editing (SXE) can be consulted to visualize the problematic. This work discovered cases where it did not ensure convergence. Combining aspects from related works, namely causal readiness and visibility, helped to develop an improved version, that works peer to peer and solves the issues respective convergence. Subsequently, the protocol even offers an adequate mechanic now to carry undo and redo operations into effect.

Unresolved remain issues that are caused by the need to ensure a well-formed XML document. This work therefore can help to initiate further research.

Besides, this work helped to improve Saros, mainly in refactoring parts of the network layer and by resolving bugs due to threading issues. Also, a prototype is provided that shows a proper way to fulfill the basic idea of separation of concerns by encapsulating new features in a separate project. Finally, it is tried to point out that a project with frequently altering members may depend more than any other on a comprehensible code base and thereby on refactoring, modularization and abstraction.

At the end, the view, network and data architectural parts of the implementation build up to a collaborative whiteboard feature, that enables freehand drawing and the creation of rectangles and ellipses. Furthermore objects can be manipulated in position and size and almost nested without limits, commands can be undone and redone as long as this has visible effects and stays in line with a well-formed XML document.

10.1 Outlook

The Eclipse Graphical Editing Framework is famous for its extensibility, furthermore the underlying model can map all kind of graphical data that will be communicated to peers in an abstract way and vice versa.

Some features I will add until my presentation, especially saving and maybe reloading.

The next feature may be a possibility to change the foreground and background color. Furthermore remote edits should be displayed in the status bar and surely a feature like a telepointer will soon be requested.

I affirm, that this work with the name “Iterative, prototype-driven development of a whiteboard feature” was composed by myself and only by myself. All used auxiliary material like books, papers, web references and technical documents is listed in the references, and quotes from other works are explicitly indicated.

Berlin, 28th December 2010

Michael Jurke

References

[XPRef03]: Stephens, M.; Rosenberg, D., Extreme Programming Refactored: The Case Against XP, 2003

[XPExam01]: des Rivieres, J.; Gamma, E.; Mätzel, K.-U.; Moore, I.; Weinand, A.; Wiegand, J., Extreme Programming Examined, 2001

[DPPFacetop04]: Stotts, David; McC. Smith, Jason; Gyllstrom, Karl, Support for Distributed Pair Programming in the Transparent Video Facetop , 2004

[SubEt]: Online reference: http://www.codingmonkeys.de/subethaedit/, last access: November 2010

[WAVE_VS]: Online reference: http://www.wave-vs.net/, last access: November 2010

[ECF]: Online reference: http://www.eclipse.org/ecf/, last access: November 2010

[RTSE]: Online reference: http://wiki.eclipse.org/RT_Shared_Editing, last access: November 2010

[Nichols et al. 1995]: Nichols, D.; Curtis, P.; Dixon, M.; Lamping, J., High-Latency, low-bandwidth windowing in the Jupiter collaboration system, 1995

[ECF30]: Online reference: http://www.eclipse.org/ecf/NewAndNoteworthy_3.0.0.html, last access: November 2010

[ECON09]: Online reference: http://www.eclipsecon.org/2009/sessions?id=429, last access: November 2010

[Dje06]: Djemili, Riad, Entwicklung einer Eclipse-Erweiterung zur Realisierung und Protokollierung verteilter Paarprogrammierung, 2006

[Bro75]: Brooks, F., The Mythical Man-Month, 1975

[Pom96]: Pomberger, G.; Blaschek, G., Software Engineering – Prototyping und objektorientierte Software-Entwicklung,

[Bisch92]: Bischofberger W., Pomberger G., Prototyping-oriented SoftwareDevelopment, 1992

[Cri91]: Crinnion, J., Evolutionary Systems Development, a practical guide to the use of prototyping within a structured systems methodology, 1991

[Over91]: Overmyer, S. P., Revolutionary vs. Evolutionary Rapid Prototyping:

Balancing Software Productivity and HCI Design Concerns. In: Proceedings of the Fourth International Conference on Human-Computer Interaction., 1991

[SWN*03]: Stotts, D.; Williams, L.; Nagappan, N.; Baheti, P.; Jen, D.; Jackson, A., Virtual Teaming: Experiments and Experiences with Distributed Pair Programming, from: Extreme Programming and Agile Methods, 2003

[Kro08]: Kröger, D., Evaluation von Distributed Pair Programming mit XPairtise , 2008

[LaMy95]: Landay, J.A., Myers, B.A., Interactive Sketching for the Early Stages of User Interface Design, November 2010

[Bü01]: Büse, D., Konzeption und prototypische Umsetzung eines SharedWhiteboard für eine kooperationsunterstützende Lernumgebung,

[GMP]: Online reference: http://www.eclipse.org/modeling/gmp/, last access: February 2010

[Inkwb09]: Online reference: http://old.nabble.com/Bye-bye-for-now-Whiteboard-td25247611.html, last access: 2009

[MOBILIS]: Online reference: http://mobilisplatform.sourceforge.net/, http://www.inf.tu-dresden.de/index.php?node_id=578&refer_id=573&ID=112&ln=de, last access: November 2010

[EclFAQ]: Online reference: http://wiki.eclipse.org/FAQ_Where_did_Eclipse_come_from%3F, last access: November 2010

[JFACE]: Online reference: http://help.eclipse.org/helios/index.jsp?topic=/org.eclipse.platform.doc.isv/guide/jface.htm, last access: November 2010

[Saros10]: Salinger, S.; Oezbek, C.; Beecher, K.; Schenk, J., Saros: An Eclipse Plug-in for Distributed Party Programming, 2010

[Ger07]: Gerlicher, A.R.S., Developing Collaborative XML Editing Systems, 2007

[XEP-0284]: ‚Govenius, J., Saint-Andre, P., Pusateri, T., XEP-0284: Shared XML Editing, 2010, http://xmpp.org/extensions/xep-0284.html, access:December 2010

[GWave]: Online reference: http://www.waveprotocol.org/whitepapers/operational-transform, last access: December 2010

[Rie08]: Rieger, O., Weiterentwicklung einer Eclipse Erweiterung zur Realisierung und Protokollierung verteilten Paarprogrammierung im Hinblick auf Kollaboration und Kommunikation, 2008

[Zi09]: Ziller, S., Behandlung von Nebenläufigkeitsaspekten in einem Werkzeug zur Verteilten Paarprogrammierung,

[Ellis89]: Ellis, C.A.; Gibbs, S.J., Concurrency control in groupware systems, 1989

[OTFAQ]: Online reference: http://cooffice.ntu.edu.sg/otfaq/, last access: November 2010

[Ignat02]: Ignat, C.; Norrie, M., Tree-based model algorithm for maintaining consistency in real-time collaborative editing systems, 2002

[Sun et al. 1996]: Sun, C.; Yang, Y.; Zhang, Y.; Chen, D., A consistencey model and supporting schemes for real-time cooperative editing systems, 1996

[XMPPHistory]: Online reference: http://xmpp.org/about-xmpp/history/, last access: November 2010

[XEP-0234]: Online reference: http://xmpp.org/extensions/xep-0234.html, last access: November 2010

[Sta10]: Staib, H., Verbesserung einer XMPP-Bibliothek für den Einsatz in verteilter Paarprogrammierung, 2010

[XEP-0002]: Online reference: http://xmpp.org/extensions/xep-0002.html, last access: 2001

[XEP-0019]: Online reference: http://xmpp.org/extensions/xep-0019.html, last access: 2002

XEP-0113: Imbens H.-J., XEP-0113: Simple Whiteboarding, 2003

[WBX06]: ‚Govenius, J., JEP-xxxx: An SVG Based Whiteboard Format, 2006, http://xmpp.org/extensions/inbox/whiteboard.html, access:December 2010

[WBX07]: ‚Bishop, M.; Lirette, K.; Fletcher, B., XEP-xxxx: Whiteboard, 2007, http://xmpp.org/extensions/inbox/whiteboard2.html, access:December 2010

[SXDE06]: ‚Govenius, J., XEP-xxxx: Shared XML Document Editing, 2006, http://xmpp.org/extensions/inbox/sxde.html, access:November 2010

[SVGOrig]: Online reference: http://www.w3.org/Graphics/SVG/WG/wiki/Secret_Origin_of_SVG, last access: November 2010

[Her09]: Hering, D., Entwicklung eines Dienstes für Real-time Collaborative Editing für die Mobilis-Plattform, 2009

[Ign06]: Ignat, C.-L.,
Maintaining Consistency in Collaboration over Hierarchical Documents, 2006

[Voi09]: Voigt, M.,
Erweiterung und Anpassung des Collaborative Editing Framework for XML (CEFX), 2009

[SVG2010]: Online reference: http://www.codedread.com/svg-support.php, last access: November 2010

[Batik]: Online reference: xml.apache.org/batik/, last access: February 2010

[SVG08]: Online reference: http://www.svgopen.org/2008/papers/72-Scalable_Vector_Graphics_to_the_Extreme/#d4e79, last access: February 2010

[SwingSWT]: Online reference: http://www.eclipse.org/articles/article.php?file=Article-Swing-SWT-Integration/index.html, last access: February 2010

[GEF]: Online reference: http://www.eclipse.org/gef/, last access: December 2010

[GEF04]: „ The Graphical Editing Framework, 2004, http://www.eclipsecon.org/2004/EclipseCon_2004_TechnicalTrackPresentations/47_Hudson.pdf, access:November 2010

[Ignat et al.03]: Ignat, C.-L.; Norrie, M. C., Customizable Collaborative Editor Relying on treeOPT Algorithm, 2003

[Ign02]: Ignat, C.; Norrie, M., Tree-based model algorithm for maintaining consistency in real-time collaborative editing systems, 2002

[Shapiro et al.]: Shapiro, M.; Bhargavan, K., The Actions-Constraints approach to replication: Definitions and proofs, 2004

[XEP-0045]: ‚Saint-Andre, P., Multi-User Chat, 2008, http://xmpp.org/extensions/xep-0045.html, access:November 2010

[Molli et al.06]: Molli,P.; Oster, G.; Urso,P.; Imine, A., Data Consistency for P2P Collaborative Editing, 2006

[Beck00]: Beck, K., Extreme Programming Explained: Embrace Change, 2000

[XPRules]: Online reference: http://www.extremeprogramming.org/rules.html, last access: December 2010

[CodeSmell]: Online reference: http://c2.com/cgi/wiki?CodeSmell, last access: December 2010