TRANSPARENT EXTENSION OF SINGLE-USER APPLICATIONS TO MULTI-USER REAL-TIME COLLABORATIVE SYSTEMS

An Aspect Oriented Approach to Framework Integration

Ansgar R. S. Gerlicher
London College of Communication, Hochschule der Medien, Stuttgart, Dingelstedtstr. 5, Hannover, Germany

Keywords: CSCW, Groupware, Collaborative Editing, Collaboration Transparency, Software-Architecture, Aspect-Oriented Programming, XML, Distributed-Systems.

Abstract: This paper discusses the transformation of a single-user SVG editing application into a multi-user real-time collaborative editing system. The application's extension with collaboration functionality was realized by using a novel aspect-oriented programming approach to framework integration. This approach is platform independent, supports heterogeneous applications and does not require an application specific API or access to the application's source code. The collaboration functionality in this case is provided by the Collaborative Editing Framework for XML (CEFX) which uses the Document Object Model as a standard interface to the application's data model.

1 INTRODUCTION

Cooperative work is a day-to-day activity in many areas. Software development teams cooperatively develop applications or write documentation. Engineers cooperatively design a circuit diagram or work together on a 3D model of a new machine. The documents that are cooperatively edited range from simple text documents over 2D graphics to complex 3D models.

However, software support for real-time group editing in commercial applications today is uncommon. Thus support for collaboration is often limited to turn-taking, split-combine and copy-merge. Existing real-time group editors - derived from research projects - are usually very specialized and despite latest achievements in the research field of Computer Supported Cooperative Work, many such systems suffer from a lack of user acceptance in the professional area. One reason for this seems to be a low motivation of users to learn new user interfaces and application functions if they can not see their personal benefit in the collaboration features (Grudin, J., 1994). Another reason may be that existing real-time group editors generally can not compete with established single-user editors regarding the functionality and usability (Xia, Sun et al. 2004).

A more promising approach therefore is to extend accepted single-user editing applications with collaborative real-time editing functionality. The difficulty with this approach is to extend an application transparently (that is without modifying the application's source code) with as little effort as possible and at the same time providing the best support for real-time collaboration.

Systems for application sharing such as VNC, NetMeeting or Netviewer allow sharing the view of any single-user application among a group of users. These systems are application independent and application transparent. The effort to share an application is very little, but they only support strict WYSIWIS. Independent user interaction on the shared document or application is not supported.

A number of research projects investigated in how to transparently extend single-user applications with collaborative editing functionality (Xia, Sun et al., 2004, Li, Li, Yu and Yang, 2003, Begole, 1999). Their approaches support real-time collaboration and relaxed WYSIWIS. However, these approaches either depend on an application's programming interface (API) or require translating operating system events into meaningful editing operations. This is labour intensive and the integration code has to be

R. S. Gerlicher A. (2007).
TRANSPARENT EXTENSION OF SINGLE-USER APPLICATIONS TO MULTI-USER REAL-TIME COLLABORATIVE SYSTEMS - An Aspect Oriented Approach to Framework Integration.
In *Proceedings of the Ninth International Conference on Enterprise Information Systems*, pages 327-334

rewritten for each new application that is to be extended.

In this paper, we propose a novel approach to extending a single-user application with collaborative real-time editing functionality. Our approach supports relaxed WYSIWIS and heterogeneous applications. An application is thereby extended without modification of the source code and without being dependent on an application API or on operating system event translation. Our approach only assumes a standard interface to the applications data model and a runtime environment that supports introspection. We use the functionality of the Collaborative Editing Framework for XML (CEFX) in order to extend an application (Gerlicher, 2006). CEFX, among other things, takes care of synchronising the shared document. This paper discusses how we applied the concepts of aspect-oriented programming (AOP) in order to integrate CEFX into an existing single-user editing application.

The paper is structured as follows. In the next section, the different approaches of recent research projects are discussed and compared with our approach. Section 3 discusses our approach, the CEFX software architecture, the implementation and how awareness support can be achieved. Section 4 discusses the requirements for this approach and then conclusions are drawn in the last section.

2 RELATED WORK AND GOALS

Xia, Sun et al. propose the Transparent Adaptation (TA) approach for the extension of single-user applications with collaboration functionality (Xia, Sun et al., 2004). In CoWord, they have extended the Microsoft Word application transparently by making use of the Microsoft application and execution environment APIs.

The TA approach requires each application to be adapted before being shared. The adaptation of an application requires the developer to have a detailed knowledge of the application and execution environment specific API. Additionally an interpretation of the user actions in relation to the current application contexts is required. Operational Transformation (OT) (Chen, Sun et al., 1998) is used for the concurrency control of a shared Word document. This requires to map each operation executed on the application's data model into an operation that can be processed by the OT concurrency control mechanism. This is the responsibility of the Collaboration Adapter in CoWord. However, mapping user actions to OT operations can become complex and requires that the application's data model supports positional addressing of objects, which may not be feasible for complex 3D modelling applications. These limitations of CoWord are not inherent to the OT approach but come from the design choices made concerning the integration of concurrency control into an application.

A similar approach is proposed by Li et al. called Intelligent Collaboration Transparency (ICT) (Li, Li, Yu and Yang, 2003). The focus of their work is on sharing heterogeneous applications of the same application family. They propose a system that allows extending single-user applications such as GVim and MS Word. For each application a so called ICT agent is implemented that captures events from the operating system and the application, translates them into semantic operations and then transmits those to the other collaborating sites, where the events are replayed in the form of a sequence of editing events. Their event capture and replay mechanism makes intensive use of application and operating system specific APIs and thus suffers from the same problems as the TA approach in terms of implementation complexity. Additionally the complexity is increased by supporting heterogeneous applications. This requires a formalisation of application semantics in order to be able to translate the user actions of one application to the relating user actions of another application.

The high implementation effort was one of the reasons for the second generation of the ICT project, ICT2. In contrast to their previous work, ICT2 does not attempt to intercept and understand the operating system level events. Instead it uses an adapted version of the diffing algorithm (Myers, 1986) to derive the editing sequences between document states (Lu and Li, 2004). However, this new approach is not suited for fine-grained real-time group editing such as TA and ICT, because of its limitations in terms of performance. The support for heterogeneous applications is limited to those applications that have the same writing style. Sharing a document between for example Latex and Word is not supported. The diff algorithm that is applied supports determining the differences of text documents only. In order to support structured and formatted documents, more sophisticated diffing algorithms would be required (Li and Lu, 2006).

Begole (Begole, 1999) proposes a different approach. The Flexible JAMM (Java Applets Made Multi-User) project extends a single-user application by replacing selected components of it with multi-user versions. The basic idea of this approach is not to use an application or operating system specific

API for the integration of collaboration function, but the Java Swing API. This has the advantage that the components which are replaced are well known and it is not required to implement a translation layer as in the TA approach or convert user actions into application semantic commands or API calls as in ICT. The disadvantage is that the set of applications that are suitable for an extension is restricted to Java Swing based applications.

To summarise, all approaches use a certain API in order to extend a single-user application. TA and ICT both use an API on operating system and on application level. JAMM uses an API on the level of the runtime's graphical user interface (GUI) library. The first two approaches face the problem of implementation complexity for each new application that is to be extended. The JAMM approach has the problem of being dependent on the fulfilment of certain runtime requirements.

The goal of our approach is to reduce the complexity of integrating a collaboration framework into a single-user application and provide a solution that is more general, supporting many different types of applications. We argue that this can be achieved by using aspect-oriented programming and concentrating on the application's data model instead of an application, operating system or GUI library API.

The application data model describes how data is represented and used. For example in a text editing application, the data model represents the text that is edited. The structure of the data model can thereby be different to its visual representation. One aspect of an application is the manipulation of the data model. The code for updating or querying the data model can be distributed within the entire application. In the terminology of AOP such aspects are called crosscutting concerns. In our approach, we identify these crosscutting concerns or system-level-concerns within an application and define advices that then create events for the underlying collaboration framework in order to synchronise the data model between the different sites. This has the advantage that once developed advices can be reused for all applications that use the same methods for the manipulation of their data model. This is for example the case for all applications using the Document Object Model (DOM) as a standard interface for the manipulation of XML content. Although new advices have to be developed for applications that use other methods for the data model manipulation, we assume that the implementation effort is low compared to other methods.

3 THE AOP APPROACH

For extending a single-user application with collaborative real-time editing functionality, the Collaborative Editing Framework for XML (CEFX) was used. CEFX is a collaborative real-time system specialised on XML based applications. It satisfies the collaborative requirements of communication, group awareness, session management and concurrency control (Pichiliani and Hirata, 2006).

We selected the GLIPS Graffiti editor (GLIPS Graffiti Editor) as single-user application for the transparent integration of CEFX. The GLIPS Graffiti editor is a cross-platform SVG graphics editor developed by ITRIS (ITRIS). It enables to create regular SVG files. As GLIPS is a Java application, it was necessary to use an aspect-oriented extension to the Java programming language. We used AspectJ for the development of the required aspects.

Aspect-oriented programming (AOP) is a programming paradigm for the separation and encapsulation of concerns, especially crosscutting concerns, within a software. The most popular AOP language is AspectJ (The AspectJ Project), developed by Gregor Kiczales et al. at Xerox PARC (PARC).

In order to encapsulate crosscutting concerns at one place, so called aspects are defined which are then integrated into the software not earlier then at compile time. Using AOP implementations that support byte-code weaving (AspectJ), allows to extend applications without access to their source code. An aspect can alter the behaviour of the base code (the non-aspect part of a program) by applying advices (additional behaviour) at various join points (points in a program) specified in a quantification or query called a pointcut (that detects whether a given join point matches). An aspect can also make binary-compatible structural changes to other classes, like adding members or parents. (Aspect-oriented programming).

The following section describes the relevant parts of the CEFX software architecture. Section 3.2. discusses the implementation of the advices and in section 3.3 the integration of awareness mechanisms is discussed.

3.1 Software Architecture

CEFX is based on a hybrid software architecture, which is a mixture of a centralised and a replicated architecture. The central server side is responsible for the session handling and management of the shared documents. When a client connects to the server in order to work on a certain document, the server checks if a session for that requested docu-

ment is already open and connects the client to it. The client then retrieves a copy of the shared document from the server. Each operation that is executed at a client site is executed locally first and then propagated to each other client in a session and to the server site. This guarantees good response times.

CEFX is composed of a set of plug-in components that are responsible for the different aspects of a collaboration system. The flexible plug-in architecture of CEFX supports the extension of the framework itself by providing new plug-in components. The figure below depicts a simplified model of the internal structure of the client part of CEFX.

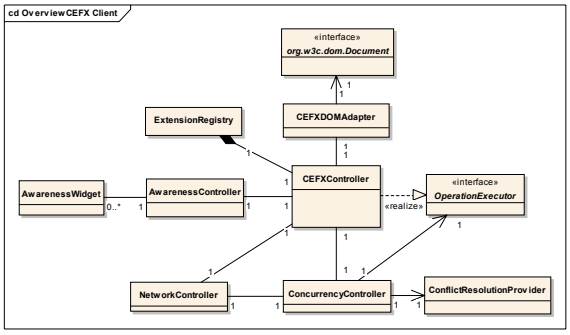

Figure 1: CEFX client components.

The CEFX Controller is the central controller component of the framework. It owns a Extension Registry (ER) which contains information on plug-in components that need to be instantiated. The CEFX Controller processes all framework events and delegates them to the corresponding components.

The Network Controller (NC) is responsible for sending and retrieving information over the network The Concurrency Controller (CC) at each client and the server site takes care of the synchronisation of the shared documents and handles conflicts. The concurrency control algorithm used was specifically developed for the synchronisation of XML documents. It follows the same principles of causality and convergence as the algorithms described in Davis, Sun et al., 2002, Ignat and Norrie, 2002 and Molli et al. 2002. In contrast to these algorithms universal unique ids (UUIDs) are used for addressing nodes and operational transformation is not applied for preserving user intentions. However, a detailed explanation of the used consistency maintenance algorithm in CEFX goes beyond the scope of this paper.

In the case of a conflict the Conflict Resolution Provider (CRP) provides hints to the concurrency controller on how the conflicting operations should be treated.

Furthermore each client site has an Awareness Controller (AC) that is responsible for the propagation of awareness events, such as mouse selection events. It delegates awareness information to the corresponding awareness widgets which are responsible for the visualisation of these events.

The CEFX DOM Adapter (DA) is our entry point to the framework. At the beginning of each collaborative session, when a document is opened by the user, the DA is provided with a reference to the Document Object Model (DOM) of the application. This is done by one of the advices that are called if a certain join point within the application is executed. For each local modification of the DOM, the DA creates an operation which is then executed and propagated to the other sites. Incoming operations are - after passing different synchronisation steps - eventually executed directly on the DOM of the application, as if they were executed locally by a user action.

3.2 Implementation

When extending the GLIPS Graffiti editor, the crosscutting concerns that we were interested in is the modification of the applications data model. The first step was to identify the relevant join points that are executed, when the data model is modified. User operations such as drawing a line, changing the colour of an object or deleting a object modify the applications data model.

The GLIPS Graffiti editor is used to create and edit Scalable Vector Graphics (SVG). As SVG is an XML document format, GLIPS uses an XML document internally as data model. For the modification of the XML document, GLIPS makes use of the DOM API. For rendering the SVG graphics to the screen, the Apache Batik library is used (Batik SVG Toolkit). Batik provides its own implementation of the DOM which complies to the W3C Document Object Model specification (W3C Document Object Model).

This simplified the identification of the relevant join points. All calls to functions defined by the W3C DOM API were possible candidates for a relevant join point.

The next step was to write an Aspect class that encapsulates the crosscutting concerns at one place. The Aspect class is similar to a Java class and can contain normal Java code and additionally AspectJ components. A simple example:

```
public aspect DOMAccessAspect {
…
}
```

The defined Aspect is then weaved into the applications bytecode. The applications source code is not modified. The Aspect class defines point cuts that are executed by AspectJ when the matching join points are reached within the application.

We identified calls to the following W3C DOM API methods as most relevant to the modification of the GLIPS data model:

```
Node appendChild(…)
Node insertBefore(…)
Node removeChild(…)
void setAttributeNS(…)
void setAttribute(…)
```

The methods `appendChild()` and `insertBefore()` are called, whenever a node is added to the document. This is the case for example if the user draws a line in the SVG document. The method `removeChild()` is called, whenever a node is removed from the document. That is the case if the user deletes an object from the document. The methods `setAttributeNS()` and `setAttribute()` are called if for example the user changes the colour of an object. In SVG the colour information of an object is contained in the value of the `style` attribute.

The figure below shows a scenario of what happens, if for example the colour of an object is changed. Whenever the method `setAttribute()` on an element node of the XML document is called, the call is intercepted by AspectJ. Instead of directly executing the code of the DOM implementation provided by Batik, the defined pointcut of our aspect class selects the relevant join point and the specific advice is applied. The advice then delegates the call to the DA. The DA creates an update operation which is handled by the CEFX Controller and asynchronously propagated to all sites of the current editing session by the NC. The operation is instantly executed on the local element node and the attribute value is updated.

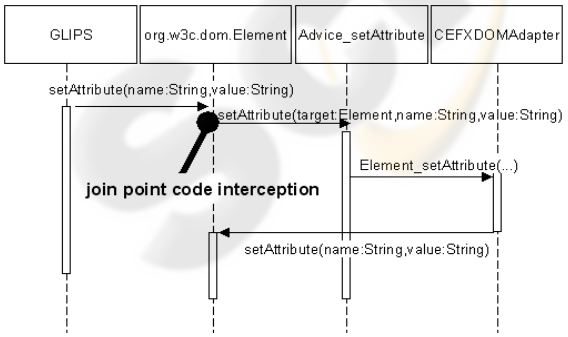

Figure 2: Scenario of code interception.

Pointcuts pick out interesting join points in the execution of a program. These join points can be for example method invocations and executions. An AspectJ pointcut definition gives a name to a pointcut. The code snippet below shows how the pointcut of the above scenario is defined in AspectJ.

```
…
pointcut setAttributePC(Element p,
String attr, String value):
target(p) && args(attr,value) &&
call (
void setAttribute(String,String))
&& !within(DOMAccessAspect);
…
```

This pointcut picks out all join points matching a call to a method with the same signature and parameters as given in the `call()` statement. For each pointcut an advice is defined, that contains the code that is to be executed if the pointcut is met.
For each of the relevant pointcuts, we defined an advice. The advice, that is executed for the above pointcut is shown in the following code snippet.

```
…
void around(Element p, String attr,
String value):
setAttributePC(p,attr,value)
{
Advices.setAttribute(p,attr,value);
}
…
```

The static method `setAttribute()` of the class `Advices` is called here. The class `Advices` is a helper class containing the advices code. This was done for clarity reasons and in order to separate the Java code from AspectJ code. The code that is executed here is shown in the following code snippet.

```
…
public static void setAttribute
(Element p,String attr,
 String value)
{
 CEFXDOMAdapter doma =
 CEFXUtil.getDOMAdapter();
 if (doma != null) {
  if(doma.isCollaborationReady()) {
   doma.Element_setAttribute(p,attr,
    value);
    return;
  }
 }
 p.setAttribute(attr, value);
}
…
```

First a reference to the `CEFXDOMAdapter` is retrieved by calling the static method `getDOMAdapter()` of the `CEFXUtil` class, a utility class provided by CEFX. If the DA is already initialised and ready for collaboration, the call to `setAttribute()` is delegated to the corresponding DA method. If the DA was not properly initialised, the `setAttribute()` of the target object is called directly, setting the new value to the attribute. This is done for example, if the client is not connected to a collaboration session.

For all other relevant pointcuts the same procedure was applied. The following code shows an example on how the other advices were defined using anonymous pointcuts.

```
…
 Node around(Element p, Element c):
 target(p) && args(c) &&
 call(Node appendChild(Node)) &&
 !within(DOMAccessAspect)
 {
   return Advices.appendChild(p,c);
 }
…
```

Additionally to the advices for the manipulation of the data model, advices for creating or loading of a document and the initialisation of the render context are needed. Creating a new SVG document or loading and parsing of an existing document is handled by the `SAXSVGDocumentFactory` class provided by Batik. Rendering of an SVG document to the screen is handled by the `SVGCanvas` class (the render context) provided by GLIPS. When a document is opened, the DA is provided with a reference to it, in order to integrate changes from the remote sites. After the execution of a remote operation, the render context of the application is notified in order to repaint the document. For this reason, the DA is provided with a reference to the application's render context. The following advice code is executed, when a document is opened by the user.

```
…
SVGDocument
around(SAXSVGDocumentFactory fac,
String uri):
target(fac) && args(uri) &&
call(SVGDocument createSVGDocu-
ment(String)) &&
!within(DOMAccessAspect)
{
//Initalisation of the DA
 CEFXDOMAdapter da =
 new CEFXDOMAdapterImpl();
//Proving DA with factory reference
 da.setDocumentFactory(fac);
//Creating the document and
//providing DA with a reference to it
 SVGDocument doc = (SVGDocument)
 da.createDocument(uri);
…
}
…
```

The DA is also provided with a reference to the document factory. This is required for example, if a session for the opened document already exists. In this case, CEFX loads the document from the server and handles the document parsing and initialisation.

The advice for setting the render context is called when the `SVGCanvas` is initialised within the application. The following code illustrates how this is achieved.

```
…
after(SVGCanvas panel): target(panel)
&& call(* initializeCanvas(*))
{
 CEFXDOMAdapter da =
 CEFXUtil.getDOMAdapter();
 if (da != null) {
  da.setRenderContext(panel);
 }
}
…
```

The `SVGCanvas` class is derived from `javax.swing.JlayeredPane` and provides a method `initializeCanvas()`. The advice is executed after the initialisation method has been called. Thus in this case the method call is not intercepted. The advice is solely used for notifying CEFX of the initialisation and providing it with a reference to the render context.

To summarize, using aspect-oriented programming for the integration of CEFX into the GLIPS Graffiti editor did not require much programming effort. Only seven advices were needed to provide GLIPS with the basic collaboration functionality. Five of the used advices were related to the DOM API and can be reused for other applications using the DOM. One advice was specific to the Batik library and one was specific to the application. The overall performance of the application did not change noticeable.

AspectJ is one of many existing implementations of AOP. In this case for example HyperJ (HyperJ Overview) could alternatively be used. AOP implementations exist for many different languages and platforms such as Java, C#, VB.NET, JavaScript, C/C++, Lua, Python, Ruby, Perl, PHP, Common Lisp and many others.

3.3 Awareness Support

The discussed advices are used to integrate CEFX in a way that satisfies the requirements of communication, session management and concurrency control. In order to satisfy the requirement of group awareness, additional effort is necessary.

CEFX provides simple awareness widgets that allow to notify each user in a collaborative session for example on other user's mouse movements. This can help a user to get an understanding of what other users are working at. The awareness widgets are little windows, controlled by the CEFX client and independent of the extended application. The application is not aware of those widgets.

For the integration of the awareness support provided by CEFX into the GLIPS editor, additional advices can be used. Java applications make use of certain interfaces from the `java.awt.event` package in order to retrieve information on mouse clicks and mouse movements. In the following example we show how we notified CEFX of a user's mouse clicks. A new Aspect class containing advices for mouse events was developed.

```
public aspect SelectionAspect {
…
 after(MouseEvent event):
 args(event) && execution(
 void mousePressed(MouseEvent))&&
 !within(SelectionAspect)
 {
   Advices.mousePressed(event);
 }
…
```

The above code snippet shows the advice that is executed when the user presses the mouse button. The static method `mousePressed()` of the Advices helper class delegates the mouse event to the `MouseEventPropagator` component of CEFX.

```
…
public static void
mousePressed(MouseEvent event)
{
 CEFXDOMAdapter da =
 CEFXUtil.getDOMAdapter();
 if (da != null) {
  MouseEventPropagator li =
  (MouseEventPropagator)
  da.getEventPropagator(event);
  if(li!=null)
  {
   li.mousePressed(event);
  }
 …
```

The `MouseEventPropagator` component is registered with the AC component of CEFX and is responsible for propagating the mouse events to the other sites, where they are displayed in the corresponding awareness widgets. The same kind of advices can be implemented for all other kinds of user events such as typing the keyboard or mouse movements. Using AOP here allows a transparent integration of awareness mechanisms into the application.

4 REQUIREMENTS

AOP implementations are available for a large number of programming languages and platforms. One requirement though is that the target application is written in a language that is supported by AOP. Additionally some AOP implementations require recompilation of the application's source code in order to weave the generated aspect code into it. Other AOP implementations do not require source code. AspectJ for example supports byte-code weaving and advanced load-time weaving. This allows using AOP without access to the application's source code, which makes it suitable for the extension of commercial applications.

The integration of CEFX into the GLIPS application had the advantage that GLIPS uses the DOM for accessing it's data model. This simplified the identification of relevant join points. For applications that do not use a standard interface for modifying their data model the identification of the join points may be more difficult, but still feasible.

It is worth noticing that the support for heterogeneous applications in this approach is limited to applications using the same type of XML document. The support for real heterogeneous applications (using different XML document types) is a subject for future research.

5 CONCLUSIONS

This paper proposes a novel approach to the integration of a collaborative editing framework in order to transparently extend a single-user application with group editing functionality.

We assume that using standardised data model interfaces and aspect-oriented concepts can dramatically reduce implementation efforts in comparison to other approaches using window event translation and application specific programming interfaces. This paper shows how little the effort is to transpar-

ently extend a single-user SVG editing application using this approach. The application was extended, by making use of the standard Document Object Model and the Collaborative Editing Framework for XML. The developed aspect-oriented programming advises are reusable and the next step will be to extend a number of other single-user applications with group editing functionality.

More and more applications today use XML as a file format, for example OpenOffice, Mircosoft Word 2007 and a number of editors for other XML based file formats. If those applications make use of the DOM API internally for the modification of their data model, this will ease their extension with real-time collaboration features.

However, one aim of this research project is to provide collaboration support to existing and future applications used for the design of vehicle electrical systems in the automotive industries. Today, the SVG format is a de facto standard for the representation of circuit diagrams in this area. Other XML based document formats such as ELOG (Electrological Model) are currently under development. The development of a vehicle electrical system is a complex process requiring intensive collaboration between a number of different companies such as the OEM, the suppliers and manufacturers of the cable loom and different subcontractors, but the current applications used in this area do not provide support for real-time collaboration. Providing a system that supports real-time collaborative engineering would allow all parties to work on a single source. This could lead to a better quality and higher productivity.

REFERENCES

Chen, D., Sun, C., Jia, X., Zhang, Y., Yang, Y., 1998.: Achieving convergence, causality-preservation, and intention-preservation in real-time cooperative editing systems. In *ACM Transactions on Computer-Human Interaction, Vol.5, No.1, pp.63-108.*

Ignat, C., Norrie, M. C., 2002.: Tree-based model algorithm for maintaining consistency in real-time collaborative editing systems. In *ACM Proceedings of the Fourth International Workshop on Collaborative Editing Systems, New Orleans, Louisiana*.

Molli, P., Skaf-Molli, H., Oster, G., Jourdain, S., 2002.: Sams: Synchronous, asynchronous, multisynchronous environments. In *Proccedings of the Seventh International Conference on CSCW in Design, Rio de Janeiro, Brazil.*

Davis, A., Sun, C., Lu, J., 2002.: Generalizing Operational Transformation to the Standard General Markup Language. In *Proceedings of ACM 2002 Conference on Computer Supported Cooperative Work, New Orleans, Louisiana, USA.*

Xia, S., Sun, D., Sun, C., Chen, D., Shen, H., 2004.: Leveraging single-user applications for multi-user collaboration: the CoWord approach. In *Proceedings of ACM 2004 Conference on Computer Supported Cooperative Work, Chicago, IL USA.*

Lu, J., Li, R., Li, D., 2004.: A state difference based approach to sharing semi-heterogeneous single-user editors. In *Proceedings of CSCW'04 workshop on collaborative systems (IWCES-6) and application sharing systems. Chicago.*

Li, D., Li, R., Yu, Y., Yang, Y., 2003.: Using Familiar Single-User Editors for Collaborative Editing. In *Proceedings of the 36th Annual Hawaii International Conference on System Sciences (HICSS'03).*

Begole, J.M.A., 1999.: Flexible Collaboration Transparency: Supporting Worker Independence in Replicated Application-Sharing Systems. *Ph.D. Dissertation. Virginia Polytechnic Institute and State University, Blacksburg.*

Gerlicher, A.R.S., 2006: A Framework for Real-time Collaborative Engineering in the Automotive Industries. In *Proceedings of Third International Conference on Cooperative Design, Visualization and Engineering, CDVE 2006, Mallorca, Spain.*

Myers, E. W., 1986.: An O(ND) difference algorithm and its variations. *Algorithmica I, pages 251-266.*

Li, D., Lu, J., 2006.: A Lightweight Approach to Transparent Sharing of Familiar Single-User Editors. In *Proceedings of ACM CSCW'06, Banff, Alberta, Canada.*

Pichiliani, M. and Hirata, C. M., 2006.: A Guide to Map Application Components to Support Multi-User Real-time Collaboration. *ITA (short paper), CollaborateCom 2006, Atlanta, Georgia, USA.*

Grudin, J., 1994.: Groupware and social dynamics: eight challenges for developers. *Communications of the ACM, Volume 37 ,Issue 1, pages 92 – 105.*

HyperJ Overview (Tarr, P). Retrieved January 14, 2007, from http://www.alphaworks.ibm.com/tech/hyperj.

GLIPS Graffiti Editor (n.d.). Retrieved January 14, 2007, from http://glipssvgeditor.sourceforge.net/.

ITRIS (n.d.). Retrieved January 14, 2007, from http://www.itris.fr.

PARC, Palo Alto Research Center, Inc. Retrieved January 14, 2007, from http://www.parc.xerox.com/.

The AspectJ Project (n.d.). Retrieved January 14, 2007, from http://www.aspectj.org.

Aspect-oriented programming - Wikipedia (n.d.). Retrieved January 14, 2007, from http://en.wikipedia.org/wiki/Aspect-oriented_programming.

Batik SVG Toolkit (n.d.). Retrieved January 14, 2007, from http://xmlgraphics.apache.org/batik/.

W3C Document Object Model (n.d.). Retrieved January 14, 2007, from http://www.w3.org/DOM.

Electrological Model, ELOG, VDA. Retrieved January 14, 2007, from http://www.ecad-if.de/elog.html.