Institute for Computer Graphics and Vision
Graz University of Technology
Graz

Towards a Collaborative Information Visualization System in a Multi-Display Environment

Diploma Thesis

Werner Puff
werner.puff@student.tugraz.at

September 2009

Supervision:
Univ. Prof. DI Dr. techn. Dieter Schmalstieg
DI Marc Streit, DI Alexander Lex, Manuela Waldner, MSc

Abstract

Visual data analysis often operates on a vast amount of data. The size of data sets as well as the knowledge related to it raises, among others, two problems. On the one hand display areas and resolutions of standard workplaces are often insufficient to visualize the data in a proper proper manner. On the other hand the knowledge and collaboration of experts from multiple domains is needed for an efficient data analysis process but the collaborative tasks are not supported by the utilized systems.

This work proposes an approach to counter these problems, based on the information visualization software Caleydo. The approach introduces extensions to the existing software in order to connect multiple Caleydo applications for the collaborative analysis of one data set. In addition, Caleydo is integrated into the Multi-Display Environment Deskotheque. This integration provides larger and more flexible display areas and also enables co-located collaboration for small groups. The extensions presented here include a communication layer to provide synchronization and data exchange between the applications. Visual Links are used to make users aware of changes, an important aspect in a setup consisting of multiple displays and multiple users.

Keywords: Information visualization, computer supported collaborative work, multi-display environment, visual links

Zusammenfassung

Auf Informationsvisualisierung basierende Analysen operieren oftmals auf sehr großen Datensätze auf. Sowohl die pure Menge der Daten als auch die damit verbundenen Wissensgebiete führen unter anderem zu zwei Problemstellungen. Einerseits sind Anzeigefläche und Auflösung von Standardarbeitsplätzen unzureichend, um eine durchwegs zufriedenstellende Arbeitsweise zu ermöglichen. Andererseits wird das Wissen von Experten aus verschiedenen Fachbereichen benötigt, um die Daten effizient analysieren zu können.

Diese Arbeit zeigt einen Lösungsweg aufbauend auf Caleydo, einer Informationsvisualisierungs Software, auf. Es werden Erweiterungen vorgestellt, um die Mehrbenutzertauglichkeit der Software zu erreichen. Des weiteren wird Caleydo in das Multi-Display Environment Deskotheque integriert. Durch diese Integration werden nicht nur größere und flexiblere Anzeigeflächen zur Verfügung gestellt, sondern auch die direkte gemeinsame Zusammenarbeit von Kleingruppen ermöglicht. Die aufgezeigten Erweiterungen umfassen eine Kommunikationsschicht, um Synchronisation und Datenaustausch zwischen den Instanzen einer verteilt laufende Applikation zu gewährleisten, als auch visuelle Hilfen, damit beim Einsatz von mehreren Anzeigesystemen und bei Mehrbenutzerbetrieb der einzelne Benutzer Änderungen immer noch zufriedenstellend wahrnehmen kann.

Deutsche Fassung:
Beschluss der Curricula-Kommission für Bachelor-, Master- und Diplomstudien vom 10.11.2008
Genehmigung des Senates am 1.12.2008

EIDESSTATTLICHE ERKLÄRUNG

Ich erkläre an Eides statt, dass ich die vorliegende Arbeit selbstständig verfasst, andere als die angegebenen Quellen/Hilfsmittel nicht benutzt, und die den benutzten Quellen wörtlich und inhaltlich entnommene Stellen als solche kenntlich gemacht habe.

Graz, am …………………………… ………………………………………………..
(Unterschrift)

Englische Fassung:

STATUTORY DECLARATION

I declare that I have authored this thesis independently, that I have not used other than the declared sources / resources, and that I have explicitly marked all material which has been quoted either literally or by content from the used sources.

…………………………… ………………………………………………..
date (signature)

Danksagung

An erster Stelle danke ich meinen Eltern, Friederike und Walter Puff für das Ermöglichen meiner Ausbildung. Ebenfalls meinen Eltern als auch meinen Geschwistern Beate, Dietmar und Gerhard danke ich dafür, dass sie mich immer wieder auf unterschiedlichste Art und Weise motiviert haben, mein Ausbildungsziel nicht aus den Augen zu verlieren.

Weiters möchte ich mich ganz herzlich bei meinen Betreuern bedanken. Dieter Schmalsteig möchte ich für die Möglichkeit danken, am Institut zu arbeiten. Marc Streit, Alexander Lex und Manuela Waldner danke ich für die durchgehend konstruktive Zusammenarbeit in den letzten Monaten, und dass sie immer Zeit für meine Fragen gefunden haben und mir mit Rat und Tat zur Seite standen.

Danke auch an alle meine Arbeitskollegen und Freunde von new10, mit denen ich einen großen Teil meiner bisherigen Zeit in Graz verbringen durfte. Ganz besonders danke ich Erwin Greimeister, der mir in vielen wichtigen Momenten immer eine Stütze war, und Andreas Schneider für seine sprachlichen Tipps und Hilfestellungen.

Ganz besonders danke ich meiner lieben Freundin Petra, vor allem dafür, dass sie mir auch in den stressigsten Zeiten meiner Ausbildung zur Seite stand.

Contents

Chapter 1

Introduction

In recent years, *Caleydo*[1], an Information Visualization tool for biomedical pathways and gene expression data, has been developed in a cooperation of the Institute for Computer Graphics and Vision of Graz University of Technology and the Institute for Pathology at the Medical University of Graz [Streit2007, Streit2008, Lex2008]. The tool is still under development and and is continuously extended with new features requested by medical researchers and test users [Streit2009, Schlegl2009, Partl2009]. Caleydo supports a variety of different visualization methods which are presented as *multiple coordinated views*. Figure 1.1 shows Caleydo running in a dual monitor setup with multiple views on each display.

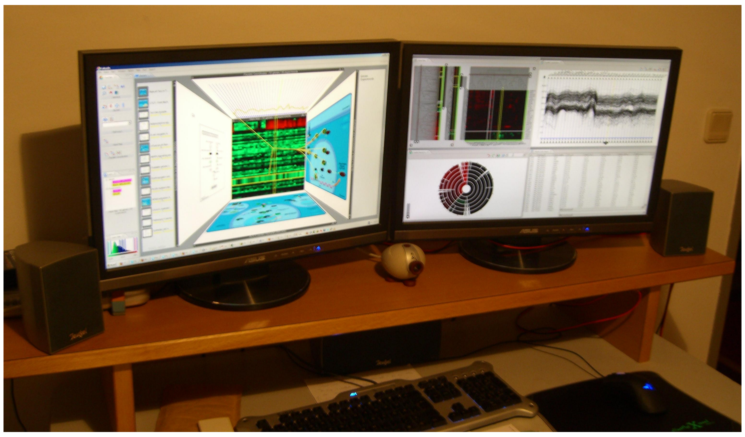


Figure 1.1: *Caleydo displaying multiple views in dual monitor setup.*

This simultaneous use often calls for more display area than a standard single or dual monitor setup can typically provide. Support for Multi-Display Environments would therefore be more than welcome.

Multi-Display Environments also counter another issue related to the analysis of biomedical pathways and gene expression data. The manifold and vast amount of data and the associated knowledge can hardly be handled by a single person. It is necessary for researchers from various domains to share their knowledge and work together during the analysis process. Projection walls and tabletop displays are specifically designed to be used by more than one person simultaneously. The Multi-Display Environment *Deskotheque*[2], another development of the Institute for Computer Graphics and Vision of Graz University of Technology, is able to embed multiple displays of such kind [Pirchheim2009].

[1]`http://www.caleydo.org/`
[2]`http://studierstube.icg.tu-graz.ac.at/deskotheque/`

Deskotheque provides a continuous interactive space across multiple heterogenous discontinuous displays for multiple users operating this environment simultanously. In addition to projection walls and tabletop displays Deskotheque also embeds standard monitor displays intended for users' private workplaces. A typical Deskotheque setup consists of several large displays spatially arranged around private workplaces, providing an environment suitable for the collaboration of small groups consisting of two to five people. Figure 1.2 shows an example setup for two users.

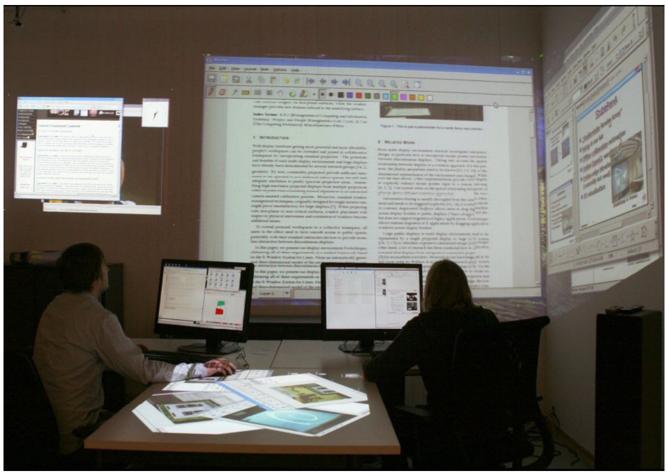


Figure 1.2: *Example setup of Deskotheque. The setup contains two standard TFT monitors, three projection wall displays and one table top display, forming a contiunous interaction space for two users [Pirchheim2009].*

Problem Statement and Contribution

This work's motivation is to fill the gap between Information Visualization systems and Multi-Display Environments designed for collaborative work. The focus lies on the existing Caleydo framework and its successful integration into Deskotheque. The combination of an Information Visualization framework and a collaborative Multi-Display Environment provides new possibilities to aid users in their tasks. Some of these features are examined in detail including the possible impact on the development of single user applications. The part of this work that is related to the implementation of the Caleydo framework consists of the following tasks:

- Data structures for concurrent access issues
- Application state serialization
- Application state persistence
- Establishing a common communication layer
- Network communication module
- Sending and receiving of selected entities over network
- Deskotheque interface

Deskotheque provides a network communication layer that is used in the implementation of the Caleydo-Deskotheque interface. Deskotheque extensions needed for this work are implemented by another research group. Implementation details of these extensions are not explained in detail here.

In addition to the integration of Caleydo into Deskotheque, a spatially distributed collaborative use case for Caleydo is investigated. Collaboration in a small group of two to five researchers, each using Caleydo as analysis tool, is achieved by the provided network communication and synchronization of the respective Caleydo applications.

After this introduction, work in related research fields is discussed in Chapter 2. The presentation of the major topics *Information Visualization*, *Computer Supported Collaborative Work* and *Multi-Display Environments* is followed by examinations related to software engineering. Advantages and disadvantages of application communication related methods and a human interaction design pattern are discussed there. An analysis of the state of the Caleydo framework at the start of this work and derived changes and additions to its software architecture are covered by Chapter 3. The focus lies on the needed application communication layer and the calculation and rendering of visual cues, which should provide an improved change awareness compared to the existing system. Application state persistence and changes to improve the general application stability are covered as well. Details about the resulting implementations are given in Chapter 4. Results and experiences are outlined in Chapter 5. Beside a presentation of the new application state persistence feature the influence of the implemented extensions to the overall application performance and stability is outlined. No results for Caleydo running in a Deskotheque Multi-Display Environment are reported yet, but experiences and measurements for a co-located setup of 2 workstations are presented. Finally the results are discussed in Chapter 6 and solutions for limitations like for example concurrent editing issues with multiple users as well as a new perspective for user interaction design using cross application Visual Links are outlined there.

Chapter 2

Related Work

This thesis uses and extends knowledge particularly from three domains. *Information Visualization*, *Computer Supported Collaborative Work* and *Multi-Display Environments*.

General Concepts of Information Visualization and their application within the pathway and gene expression visualization framework *Caleydo* are shown in Section 2.1. Due to awareness requirements of collaborative system a special emphasize is put on Visual Links in Section 2.1.3. Section 2.2 covers findings related to Computer Supported Collaborative Work in general. Multi-Display Environments and especially the implementation *Deskotheque* are presented in Section 2.3.

Distributed Applications and *Serialization* issues are directly involved when using collaborative setups and Multi-Display Environment build upon distributed systems. Section 2.4.1 and Section 2.4.2 outline previous work on problems related to distributed applications and serialization.

2.1 Information Visualization

Information Visualization (InfoVis) is the computer science field that relates to the usage of visualization techniques for the presentation of abstract data. InfoVis can clearly be differentiated from *Scientific Visualization* which focuses on the natural visualization of spatial data.

Most of today's Information Visualization systems follow the visualization pipeline architecture as outlined in [Card1999]. The visualization pipeline is a stepwise process. Each step defines operations to refine the data which is used as input for the next step. The output of the pipeline is the image resulting from the visualization process. The steps of the pipeline are illustrated in Figure 2.1.

Data Analysis: Operations like smoothing or interpolation are applied on the raw data to prepare it for the visualization process. This operations are performed automatically and only little user input is utilized. This step can be performed only once right after loading the data set as a preprocessing step.

Filtering: This operation applies mainly user defined filter operations on the data set. The data set is reduced to a subset according to the current visualization needs of the user.

Mapping: The data is transformed to geometric primitives. The chosen visualization technique is applied during this step.

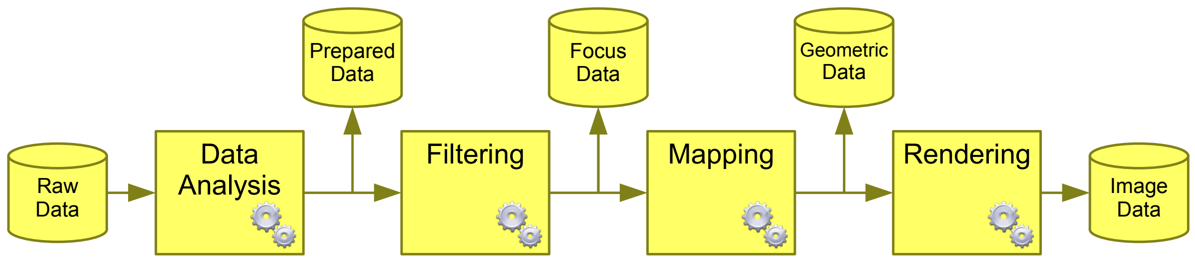


Figure 2.1: *Visualization pipeline. The stepwise process from raw data to a rendered image. Adapted from [Card1999]*.

Rendering: The geometrical information is rendered as an image ready for displaying it on an output device.

The pipeline is the basis for the appliance of the *Visual Information Seeking Mantra* of [Shneiderman1996]: *Overview first, zoom and filter, then details on demand.* The role of this mantra is discussed in [Craft2005] by Craft and Cairns. On the one hand, the mantra is criticized for not being formally evaluated, on the other hand [Craft2005] emphasizes the importance and influence of it for today's InfoVis researchers and InfoVis systems. The filtering operation utilizes the input of the user, that results from supported possibilities of interaction according to the mantra. The mapping and rendering steps are needed to display the results to user.

Overview First: If a human investigates something, it is a natural way to get an overview of the investigated entity in a first glance. An InfoVis application should give an overview of the visualized dataset right after application-start. For instance, Google Earth[1] shows the terrestrial globe at start. On the other hand it can make sense to start with some kind of initial input, as for example in pathway visualization as discussed in [Streit2007], [Streit2008].

Zoom and Filter: Following the natural way of a human to explore new things, the next step after getting an overview is to get more detailed information on a selected subset of the data. Filtering means to show only data that meets attributes specified by the user. Zooming is a modification of the viewpoint. Zooming-in means a reduction in the distance between the viewpoint and the examined object while zooming-out means an increase.

Details on Demand: Due to the often large datasets InfoVis is based on, it is not practical if even possible to show always all details on each information element. When looking on a GIS world map on a computer monitor it would not be possible to show all cities of the world including their number of inhabitants at once. Therefore such details are only displayed on demand of the user, for example by hovering the mouse over a selected entity.

The processes defined by the mantra is not static and can be applied individually according to the requirements of the analysis process.

[1]`http://earth.google.com/`

2.1.1 Multiple Views

With larger display areas using two or more views on one or more given datasets simultaneously is an established way in many different kind of visualization software like GIS, CAD, or InfoVis systems.

The same type of view rendered multiple times with different datasets is a visual comparison method of datasets to find similarities or differences. A special form of multiple views on different datasets is for instance the scatterplot matrix [Cleveland1993]. The scatterplot matrix spatially arranges a number of scatterplots in a matrix form. Each scatterplot is based on the selection of two attributes of the source data set. Each line of the matrix shows the combination of one attribute with all other attributes of the data set. Scatterplot matrices provide a complete visualization of the dataset in combination with the intuitive perception of correlations when using scatterplot views. Figure 2.2 shows a scatterplot matrix of Iris Data.

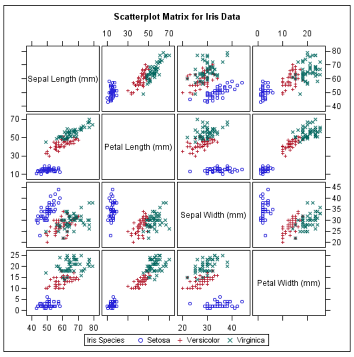


Figure 2.2: *Scatterplot Matrix for Iris Data [Cleveland1993].*

Another utilization of the multiple view approach uses different type of views on the same dataset. This method focuses on multiple aspects of the dataset simultanously by combining the visualization and interaction strengths of the particular views. For example *SimVis* [Doleisch2007] uses multiple views for interactive and feature-based analysis of large and high-dimensional data (see Figure 2.3).

GIS and real time strategy games visualize world maps or a part of it. The typical approach that can be found in most of that kind of software is to show a large detailed view for focused exploration of the data and a smaller view with a low zoom level for context information and overall navigation. Examples of GIS systems using two views can be seen in Figure 2.4.

[Plumlee2006] approved the usability of a multiple view approach by evaluating multiple views in comparison with zooming mechanics. They showed that approaches using one overview window for navigation and another window for detailed focus view has a better user acceptance than the ability to dynamically zoom in and out.

[Baldonado2000] published guidelines for using multiple views in InfoVis systems. They formalized eight rules which tell a InfoVis system designer when and how multiple views should be used to achieve usability goals. The rules are listed in Table 2.1.

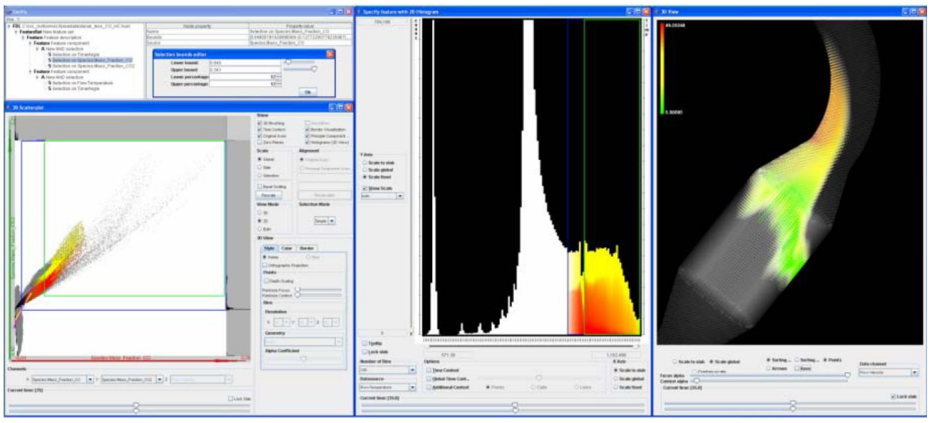


Figure 2.3: *A sample SimVis scenario: simulated flow through a diesl particle filter is visualized [Doleisch2007].*

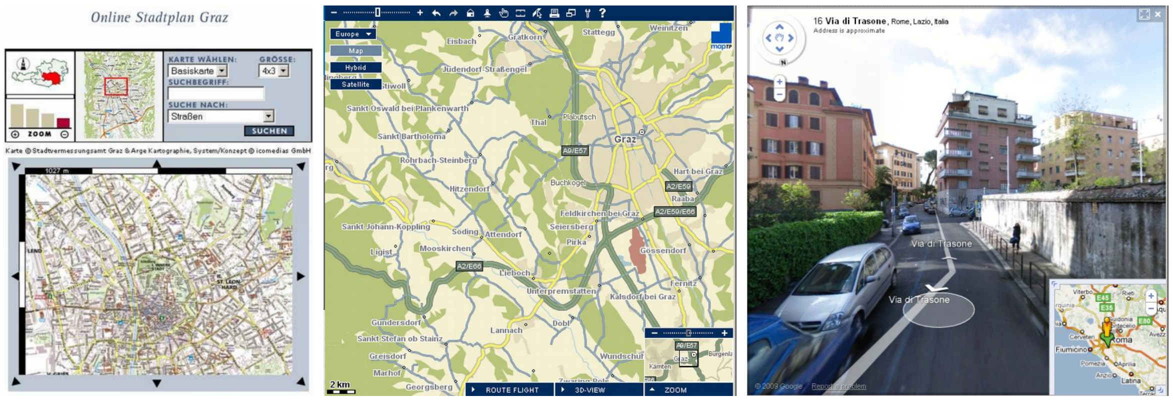


Figure 2.4: *Screenshots from three GIS systems using multiple views, from left to right: Online Stadtplan Graz (`http://www.graz.at/stadtplan`), map24 (`http://www.map24.at`) and googleMaps (`http://www.googlemaps.com`); all three are using a small overview map-view and a large focused view.*

2.1.2 Caleydo InfoVis Framework

Caleydo is a framework focused on the visualization of genetic data. Its major goal is to support life science experts to understand the role of genes in diseases. The framework is constantly extended by using findings of recent research. Details are described in [Streit2007, Streit2008, Lex2008, Streit2009]. The approach to achieve this goal is to find similarities and differences between data of healthy persons and persons with a diagnosed disease. This means that the measured data of thousands of gene-expressions of multiple individuals have to be compared in the context of existent knowledge on pathways, which are models of processes in cells [Streit2007, Streit2008]. Caleydo provides a variety of views to the researchers.

Hierarchical Heat Map: The heat map visualization method was introduced by Eisen et al. [Eisen1998]. Data sets in matrix format with different gene data represented by rows and the different experiments on individuals represented by the columns are a com-

Rule	**Summary**
Diversity	Use multiple views when there is a diversity of attributes, models, user profiles, levels of abstraction, or genres.
Complementarity	Use multiple views when different views bring out correlations and/or disparities.
Decomposition	Partition complex data into multiple views to create manageable chunks and to provide insight into the interaction among different dimensions.
Parsimony	Use multiple views minimally.
Space/Time Resource Optimization	Balance the spatial and temporal costs of presenting multiple views with the spatial and temporal benefits of using the views.
Self-Evidence	Use perceptual cues to make relationships among multiple views more apparent to the user.
Consistency	Make the interfaces for multiple views consistent, and make the states of multiple views consistent.
Attention Management	Use perceptual techniques to focus the user's attention on the right view at the right time.

Table 2.1: *Summary of the rules to use multiple views published by [Baldonado2000].*

mon form of gene expression data files. Heat maps use a matrix format of this kind for visualization. Each cell of the matrix is colored on the basis of the measured fluorescence ratio. The expressiveness of heat maps is strengthened when special orderings are applied, on rows or columns or both. With applied clustering on the genes, as proposed by Eisen et al., genes are grouped by its functions. This results from the observation, that genes with similar expression regulation often have similar functions.

When trying to visualize the thousands of gene-expressions with heat maps, the main limitation of heat maps, the screen space, becomes obvious. Even the largest available display devices would not be capable of displaying that vast amount of rows. The *Hierarchical Clustering Explorer* was introduced in [Seo2002]. This tool uses an overview of the heatmap data for overall navigation in combination with a more detailed view similar to the GIS tools shown earlier in Figure 2.4. The overview is rendered by taking the average values of adjacent cells to calculate the display color. The detailed view can be zoomed in the range of using two to ten pixels for each data cell.

Caleydo supports a Hierarchical Heatmap view with up to three simultaneously displayed detail levels with adjustable zoom [Schlegl2009]. The user may apply one of a variety of cluster algorithms to the gene expression data on rows, columns or both. Hierarchical cluster information is shown in a dendogram directly related to the heatmap view with the highest detail level. Figure 2.5 shows the Hierarchical Heatmap view with and without clustering.

Pathways: The cellular processes can be represented as a huge and complex network. A pathway is a small subsection of this network that describes for example chemical reaction cascades in cells by modeling the cellular functions in a graph. A metabolic network formed by interconnected metabolic pathways is defined in [Bourqui2006]. Biomedical databases

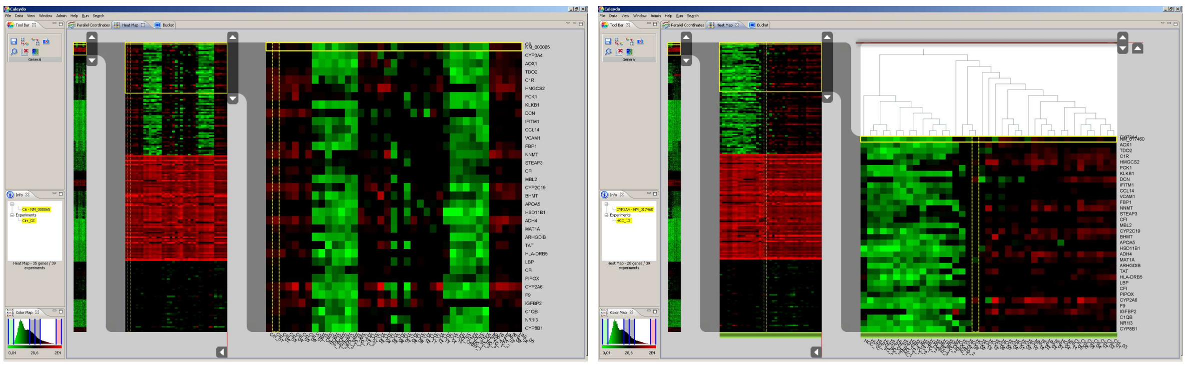


Figure 2.5: *Hierarchical Heatmap view of Caleydo without clustering on the left side and with applied clustering and dendogram on the right side.*

try to organize the steadily increasing amount of data. The existing databases focus on different objectives like correctness or completeness.

Caleydo supports automatic pathway-download from KEGG[2] and BioCarta[3] (Figure 2.6). Information contained within the downloaded pathways are set into relation with each other and the analyzed gene expression data and other meta information displayed in the various views of Caledyo [Streit2007, Streit2008].

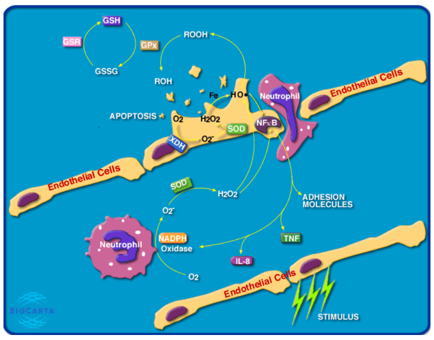


Figure 2.6: *BioCarta sample pathway for Free Radical Induced Apoptosis* (*taken from* `http://www.biocarta.com/pathfiles/h_freePathway.asp`).

Parallel Coordinates: The parallel coordinates visualization method is a way to present multidimensional data in a 2D view originally introduced in [Inselberg1985]. This kind of view builds upon a number of parallel axes, one for each dimension in the multidimensional source data set. One point in the data set is a polyline with vertices on each of the axes according to the coordinate value of the point for the related dimension. [Siirtola2006] presents a survey of interaction techniques for parallel coordinates. Furthermore [Siirtola2006] underlines the usability beyond beeing an experts-only representation with a user study that compared database related tasks performed with the standard query language compared with a parallel coordinates based browser.

As described in [Lex2008], Caleydo uses parallel coordinates by using data of one gene

[2]`http://www.genome.jp/kegg`
[3]`http://www.biocarta.com`

expression as point in the coordinate system of the individual experiments or vice versa. Caleydo's parallel coordinate view supports filtering based on 1D-selection and by the

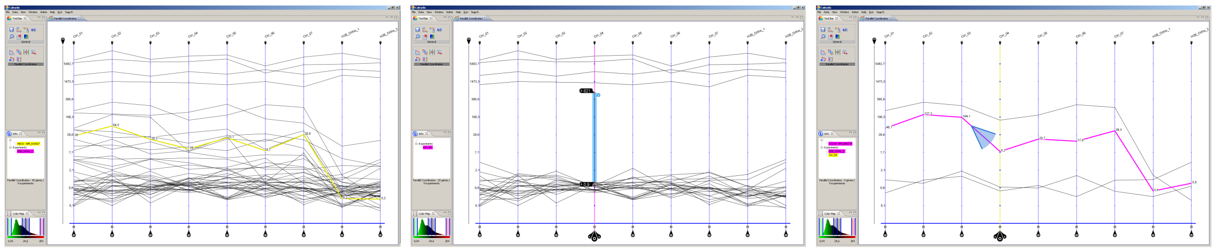


Figure 2.7: *Caleydo's parallel coordinates (left) and the same view with applied 1D selection filter (middle) and with an angular brush filter (right).*

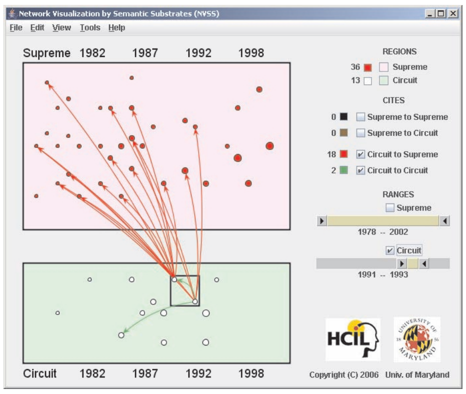
definition of angular brushes [Hauser2002, Doleisch2002]. 1D-selection filters polylines according to a specified range on one of the axis. Angular brushes allow the definition of a relative change of the values between adjacent axes. Figure 2.7 shows the parallel coordinate views with applied filters and brushes. Furthermore the view supports reordering, duplicating or removing of axes.

Radial Hierarchy: [Andrews1998] and [Chuah1998] introduced the first radial space filling hierarchy visualization. This visualization method draws circular rings for each hierarchy level with an angular area of space for each element on that hierarchy according to a measureable value of the element. A radial hierarchy view is available in Caleydo and is used for exploration of hierarchies resulting from the appliance of clustering algorithms on the gene expression data.

2.1.3 Visual Links

Highlighting an element as response to its selection is a well known computer-human-interaction behavior. The same element might be visible in different views at the same time. It could be beneficial for the user if the highlighting would not only concern about the view the user currently interacts with but would also be visible in the other views.

Visual Links extend this idea by not only highlighting the elements in the different views but also drawing edges or curves between related elements. Early works about Visual Links include [Fekete2003], treemaps with overlayed graph links, or [Neuman2005], relations within hierarchical data with edges between related data elements. [Shneiderman2006, Aris2007] used Visual Links to interconnect semantic subtrates of large networks (see Figure 2.8). Semantic subtrates are non-overlapping sub-divisions of the whole data set. They followed their observations that in network visualizations of 10 to 50 nodes and 20 to 100 links the user can still count the number of links and nodes and follow each link from source to destination in opposite to visualizations showing a larger data set at once. In the proposed visualization system the users are able to filter connections in and between semantic subtrates by a number of immediately applicable filter definitions.



Network Visualization by Semantic Substrates (NVSS)
File Edit View Tools Help
Supreme 1982 1987 1992 1998
Circuit 1982 1987 1992 1998
REGIONS
36 Supreme
13 Circuit
CITES
0 Supreme to Supreme
0 Supreme to Circuit
18 Circuit to Supreme
2 Circuit to Circuit
RANGES
Supreme
1978 -- 2002
Circuit
1991 -- 1993
HCIL
Copyright (C) 2006 Univ. of Maryland

Figure 2.8: *Two semantic subtrates views of a network. Each, the green and red area, is a semantic subtrate of one large network. Visual Links connect elements with the views itself and beyond view boundaries [Shneiderman2006].*

Collins and Carpendale [Collins2007] extended the visualization tool *prefuse* [Heer2005]. They render the provided views of the visualization toolkit composed as planes within a 3D scene. They draw edges between related items in adjacent views. The user is allowed to arrange the view-planes in the scene. Instead of drawing multiple lines from elements in one view that are all related to one element in another view, the connection lines are bundled. This avoids some visual clutter.

NodeTrix, introduced in [Henry2007] is a tool for the visualization of social networks. The tool utilizes the fact, that social networks are globally sparse but locally dense. Therefore it uses matrices to visualize local communities and Visual Links to connect the matrices. The user has the possibility to switch between three options for visual link representation. Aggregated links show simply general connections between the matrices as a whole, not showing the interactions of individuals. Underlying links connect the individuals of each of the matrices. Detailed interaction information is shown at the cost of many more links and possible crossings. The third option is a combination of the aggregated and the underlying links options. Attributes of the network may be mapped to visual variables like color or line width, providing the user additional visual information about the network. Figure 2.9 illustrates different visualization methods of *NodeTrix*.

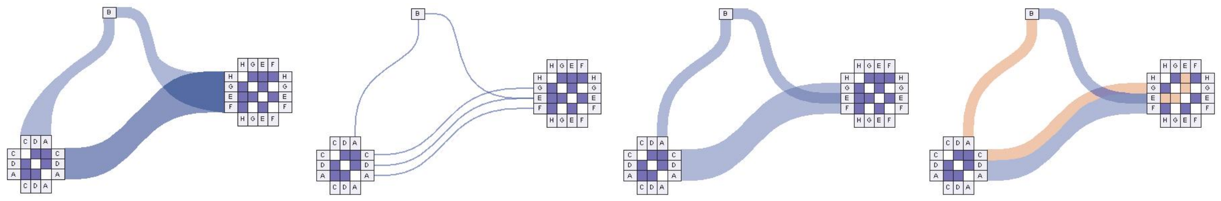


Figure 2.9: *Different types of Visual Links in the social network visualization tool NodeTrix [Henry2007]. From left to right: Aggregated links, underlying links, underlying links with full thickness, underlying links with attributes.*

2.1.4 Composite Views

Bucket and Jukebox: Caleydo's Bucket and Jukebox views are composite views consisting of a variable number of other views. The jukebox view (see Figure 2.10) got its name for working similar to an audio-jukebox where records are selected from a list and loaded onto the turntable. By selecting an entry from a list of pathways, the pathway graph is loaded to the intermediate level, a 2.5D stacked layer view. A third stage shows a large view of a pathway for close investigation.

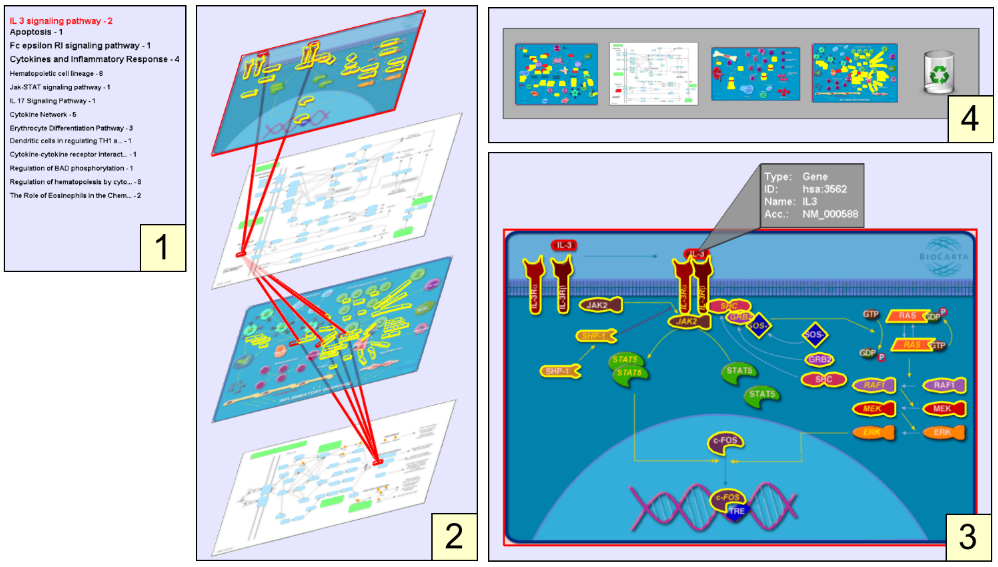

IL 3 signaling pathway - 2
Apoptosis - 1
Fc epsilon RI signaling pathway - 1
Cytokines and Inflammatory Response - 4
Hematopoietic cell lineage - 8
Jak-STAT signaling pathway - 1
IL 17 Signaling Pathway - 1
Cytokine Network - 5
Erythrocyte Differentiation Pathway - 3
Dendritic cells in regulating TH1 a... - 1
Cytokine-cytokine receptor interact... - 1
Regulation of BAD phosphorylation - 1
Regulation of hematopoiesis by cyto... - 8
The Role of Eosinophils in the Chem... - 2
1
2
3
4
Type: Gene
ID: hsa:3562
Name: IL3
Acc.: NM_000588
IL-3
JAK2
STAT5
SOS
GRB2
SHC
RAS
RAF1
MEK
ERK
c-FOS
TRE
GTP
GDP
BIOCARTA

Figure 2.10: *Jukebox view [Streit2008] combining a textual list menu for browsing related pathways by name (1) with an interconnected pathway stack (2), an area designated for a detailed examination of a graph (3), and a memo pad (4) [Lex2008].*

The Bucket is also a 2.5D view, but contrary to the Jukebox the intermediate stack layer view is replaced by positioning the contained views around the large view, comparable to the walls of a Bucket (see Figure 2.11). This placement leads to a closer arrangement on the screen and results also in a closer relationship between the views.

[Streit2008, Streit2009a] used a similar approach as [Shneiderman2006, Aris2007, Collins2007] to interconnect the individual views in Caleydo's Jukebox and Bucket view. On the one hand Visual Links are used to connect elements of the same kind in different views. On the other hand they are used to link pathway graphs (see Figure 2.10), subsets of the cellular network as explained in Section 2.1.2.

The described composite views try to solve the problem of limited display space by arranging views in a 3D scene. This is only possible for the price of a lower detail level and perspective distortion, which handicaps the analysis process [Robertson2000].

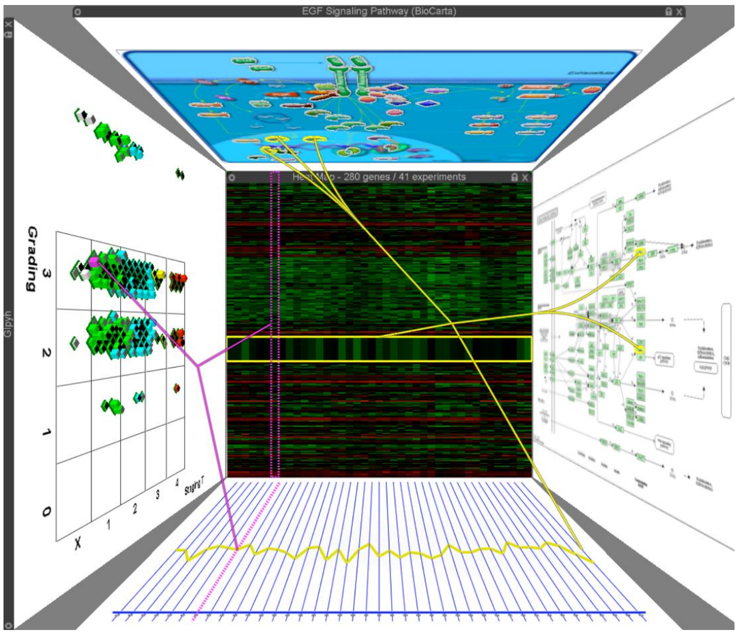

EGF Signaling Pathway (BioCarta)
HeatMap - 280 genes / 41 experiments
Grading
Glyph
X
StgVgT

Figure 2.11: *Bucket view composed of 5 different views and Visual Links.*

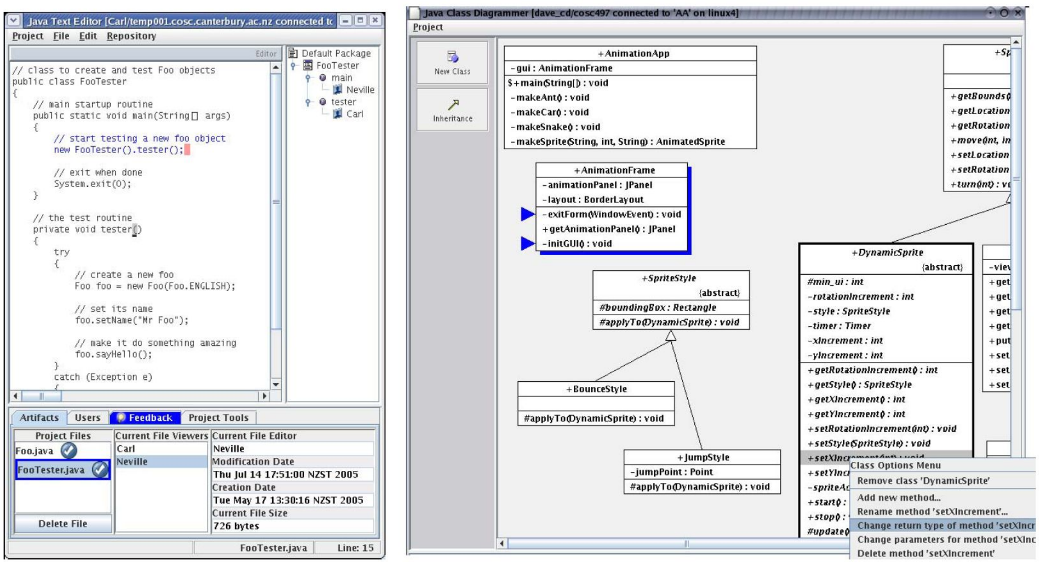
2.2 Computer Supported Collaborative Work

In the life sciences the growth of information in the recent years has been exponential [Streit2007]. This results in a vast amount of available information that can hardly be handled by single persons. It is required to combine the knowledge of experts with specializations in different research fields to fully understand the data. Multiple experts have to work tightly together to master the information and achieve new findings. This kind of team work on a given problem is called collaboration.

As described in Section 2.1.2, computer support is already a common need to visualize the available information. We have computer based tools, like Information Visualization software, for helping problem solving on the one hand. On the other hand, we have the need for collaboration of multiple users and well established computer networks. The research field *Computer Supported Collaborative Work (CSCW)* describes and analyzes ways, how people work together to achieve a common goal with the support of associated hard- and software infrastructure. CSCW investigates the possibility for general collaboration support and the support of collaboration within tools for specialized tasks.

Examples for general collaboration support are instant messengers or conferencing systems. [Nardi2000] discusses an ethnographic study of instant messaging in the workplace. They investigated the support and possibilities of instant messengers in today's working environment and working behavior. They outline the importance of informal communication processes outside of the information exchange, which they call *Outeraction*. Moreover the influence on the overall information exchange process is discussed. [Isaacs2002] found that the primary use was for complex work discussions by logging and investigating thousands of workplace instant messaging conversations. Moreover they pointed out that instant messaging in a group of people working together are used for a range of collaborative activities.

As a matter of fact that computer based tools and software engineering go hand in hand, tools for distributed software engineering are a very common topic when looking into task specialized collaboration tools. Versioning systems like subversion[4] are state of the art in today's software industry. Beside well established technologies and tools, software engineering in combination with CSCW is still an important research topic. [Boulila2004] proposed a solution for distributed software modeling. They built a prototype of a distributed UML modeling software with multi user support based on a floor control mechanism, an integrated chat client, and a knowledge management module. [Cook2005] introduced CAISE, a prototype of an integrated development environment (IDE) with collaboration support that exceeds the common interface to the source control system (see Figure 2.12). CAISE keeps the programmers action synchronized in real time. Beside other features CAISE also supports tele-cursors, remote modification highlighting and visual awareness aspects.



```
// class to create and test Foo objects
public class FooTester
{
    // main startup routine
    public static void main(String[] args)
    {
        // start testing a new foo object
        new FooTester().tester();

        // exit when done
        System.exit(0);
    }

    // the test routine
    private void tester()
    {
        try
        {
            // create a new foo
            Foo foo = new Foo(Foo.ENGLISH);

            // set its name
            foo.setName("Mr Foo");

            // make it do something amazing
            foo.sayHello();
        }
        catch (Exception e)
        {
```

Project Files	Current File Viewers	Current File Editor
Foo.java	Carl	Neville
FooTester.java	Neville	Modification Date
		Thu Jul 14 17:51:00 NZST 2005
		Creation Date
		Tue May 17 13:30:16 NZST 2005
		Current File Size
		726 bytes

FooTester.java Line: 15

Figure 2.12: *CAISE development tools with CSCW awareness support, collaborative code editor on the left side and collaborative UML class diagrammer on the right side [Cook2005].*

CSCW is also an important topic offside software engineering. [Kaplan1992] pointed out the state of the art CSCW support of operating systems in 1992 like networking features. Furthermore they discussed requirements and possibilities of operating systems to support CSCW and suggested four major features for operating systems:

- Objects, not files,
- awareness & object sharing,
- flexible, dynamic groups, and
- multimedia & casual interaction support.

[4]http://subversion.tigris.org

Some of those features have already made their way into today's IT-landscape. For example when managing music libraries used on multiple computers and portable music players, the users have to deal with songs, albums and playlists and do not have to care about files within the file-system of the different devices.

Looking at temporal and spatial aspects of CSCW, it can be divided into different fields. Examining the geographic arrangement of the team members, collaboration can be classified in distributed and co-located. When looking into the timeline of work processes, a differentiation into synchronous and asynchronous collaboration becomes visible. This classifications and its combinations are visualized in Figure 2.13 introduced by [Johansen1988].

	synchronous same time	**asynchronous** different time
co-located same place	**Face to face interactions** decision rooms, single display groupware, shared table, wall displays, roomware, ...	**Continous task** team rooms, large public display, shift work groupware, project management, ...
distributed different place	**Remote interactions** video conferencing, instance messaging, chats/MUDs/virtual worlds, shared screens, ...	**Communication + coordination** email, bulletin boards, blogs, asynchronous conferencing, group calendars, workflow, version control, wikis, ...

Figure 2.13: *The CSCW Matrix shows differentiation of collaborative work along the dimensions, time and place [Johansen1988].*

As collaboration means an ongoing communication and getting along with each other is a must, social aspects come side by side with collaboration. [Ocker2009] describes the advantage of students that are educated in advance before participating in CSCW.

[Linebarger2005] shows that during synchronous collaboration with appropriate application support the team members are more likely to form common mental models of the analyzed problems. The direct influence of shared mental models to the effectiveness of a team has been empirically analyzed by [Mathieu2000]. Outcomes of work of teams are of better quality if not only their knowledge overlaps but also the knowledge organization has synergies.

2.2.1 Co-located Synchronous Collaboration

Collocated synchronous collaboration means that the team members share a room or a comparable workspace and they can see and speak to each other. This kind of collaboration can be seen as its natural form as it typically happens when two or more people come together at one place at the same time to work on the same problem. A simple example for synchronous collaboration is two people discussing a problem face-to-face. More complex co-located synchronous collaboration could embrace more team members and the usage of additional tools like pen and paper, flip-charts or projection walls.

A popular approach for co-located collaborative environments are tabletop displays. [Hornecker2008] investigated the difference of using touch input versus mice input during co-located collaboration. They found out, that with touch input the interactions are more fluid and interferences have been solved quicker. The main reason for this is the higher awareness factor of touch input over mice input. Furthermore this study shows the importance of change awareness in collaboration software not only for the co-located synchronous case.

To support mixed-focus situations, several researchers (for example [Rekimoto1999] and [Scott2003]) recommended the usage of public display spaces in combination with private displays for individual work. Deskotheque adopted this concept by the definition of private and public usage for each display area within the Multi-Display Environment [Pirchheim2009].

2.2.2 Distributed Synchronous Collaboration

Distributed synchronous collaboration means that each team member works at the same at spatially distributed workplaces. All communication has to be done via the tools provided by the CSCW environment.

The global activities of today's companies shows already the importance of distributed collaboration. It is hardly possible for a worldwide active company to arrange face-to-face meetings for all collaborative tasks, as travel costs and time would be enormous. In fact today's global activites would not be possible without the help of communication tools like phones, which are already one form of collaboration tools. More complex examples would be instant messengers and video conferencing systems.

[Nardi2000, Churchill1999] show the importance of instant messaging and other light weight communication tools in distributed CSCW in comparison with face-to-face communication. Some advantages of instant messaging are:

- Less interruptive initiation of conversation than phone calls,
- persistence of textual conversation,
- showing negotiating availability, and
- encourages informal conversation.

[Greenberg2001] investigate the usage of the groupware system *Notification Collage*. They found out, that the users showed an obvious curiosity related to all changes within the groupware software. Especially when used within a multi monitor setup, it was very common to use one monitor exclusively for the groupware software (see Figure 2.14).

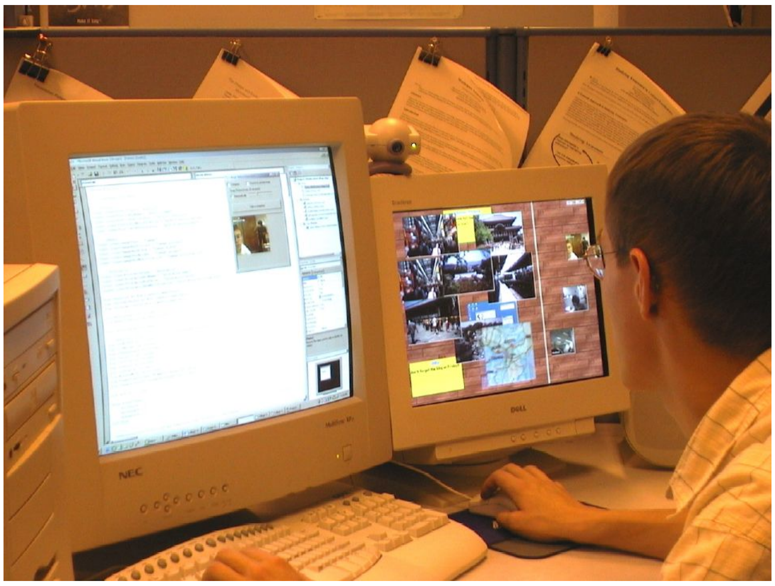


Figure 2.14: *The Notification Collage on a second monitor in a personal setting [Greenberg2001].*

2.2.3 Co-located and Distributed Asynchronous Collaboration

Asynchronous collaboration means that the team members work at different times. One reason for that is, that team members are from different time zones. Visualization tools like Many Eyes [Fernanda2008] or sense.us [Heer2007] focus on asynchronous collaboration. As there is no real time communication between the team members in asynchronous collaboration, change awareness must be handled differently and annotating the available information is a very important aspect.

However this thesis does not concern asynchronous collaboration issues.

2.2.4 CSCW System Architectures

In many cases, CSCW needs to connect multiple computer systems and applications running on those computer systems over a network. Two used architectural designs can be differentiated, client-server or peer-to-peer architecture (see Figure 2.15).

In client-server based models, one system takes the role of the server, that is the only responsible instance for overall management and synchronization tasks. A typical representative of client-server software can be seen in the world wide web, with the web-server in the role of the server and the web-browsers as clients. [Beca1999] introduces Tango Beans,

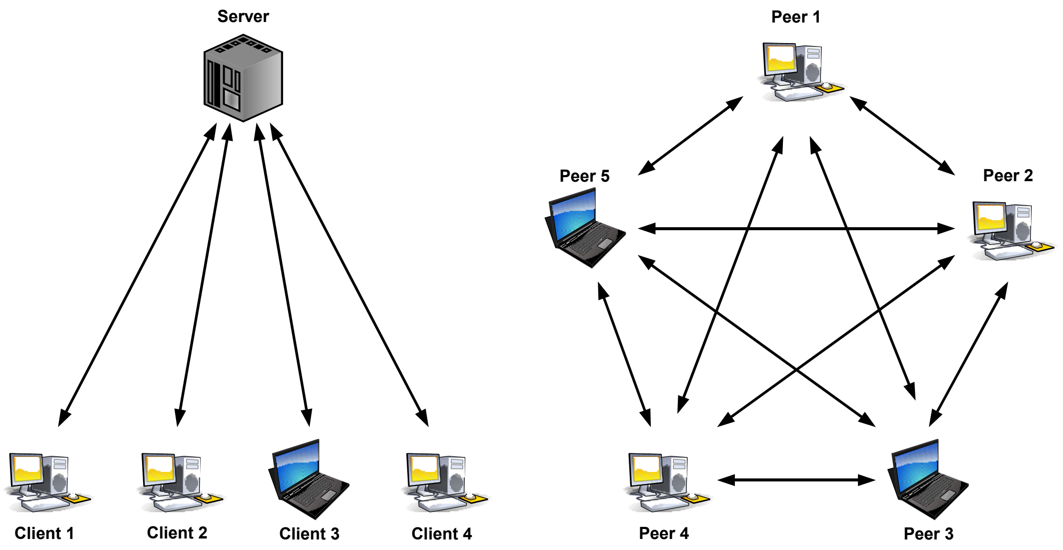


Figure 2.15: *System architecture of image client-server on the left side and peer-to-peer architecture on the right side.*

software components designed for the use within collaboration services in client-server based models.

With the availability of internet connection for the mass market, the possibility of sharing each connected computer resources with each other came along. In opposite to the classical client-server models, peer-to-peer computing sees each computer within a network as a equal usable resource for running an application. In [Osais2006] a peer-to-peer framework for synchronous collaboration support is introduced and possible performance benefits are outlined.

Deskotheque uses a mixed architecture. A central master application coordinates the particular computer nodes, while the window manager related communication uses a peer-to-peer approach.

2.2.5 CSCW and InfoVis

Combining InfoVis and CSCW could either result into using general, separated CSCW tools side by side with the InfoVis tools or by dedicated collaboration support built into the InfoVis software. Using a version control systems to store the data or using messengers to communicate are examples for general CSCW tools. Synchronizing views of different team members or annotating the analyzed data with sticky notes visible for all team members are typical examples for built in CSCW support.

[Mark2002] compared collaboration teams and individuals on finding answers by using information visualization systems. The user study compared individuals' and collaboration groups' findings of question and discovery tasks, either using InfoZoom[5] or Spotfire[6] as information visualization software. Using Spotfire users must plan in advance what variables to use and how to visualize them. Users of InfoZoom, in contrast, can rely much more often on visual cues when accessing both data and system functionality. InfoZoom can therefore be seen as more transparent compared to Spotfire, where transparency means to invoke an easy-to-understand system image in users [Sharp2002]. The results showed an advantage of collaboration groups over individuals when using a transparent information visualization system. In question tasks the collaboration-group had a higher probability of finding the correct answer. In the data discovery task the group showed a higher accuracy, because groups are more likely to locate and correct errors in their findings than individuals.

[Isenberg2008] analyzed collaborative tasks on paper based visualizations and made implications for the needs of a collaboration framework:

Support flexible temporal sequence of work processes: Individuals have unique information analysis practices. Allow group members to be engaged in different types of processes at the same time.

Support changing work strategies: Team-member switch dynamically from closely coupled and more independent work. Support individual and shared views of and interactions on the data.

Support flexible workspace organization: Individuals have different preferences to the workspace layout. Allow users to impose a spatial organization of the information artifacts.

[Isenberg2007] designed a system for collaborative tree comparison on radial views and cladograms. Their system supports free workspace organization. So team members can switch fluidly from independent to more collaborative work. Furthermore the system supports interactive sticky notes and free annotations within the view (see Figure 2.16). Sticky notes and annotations support communication between the collaborators.

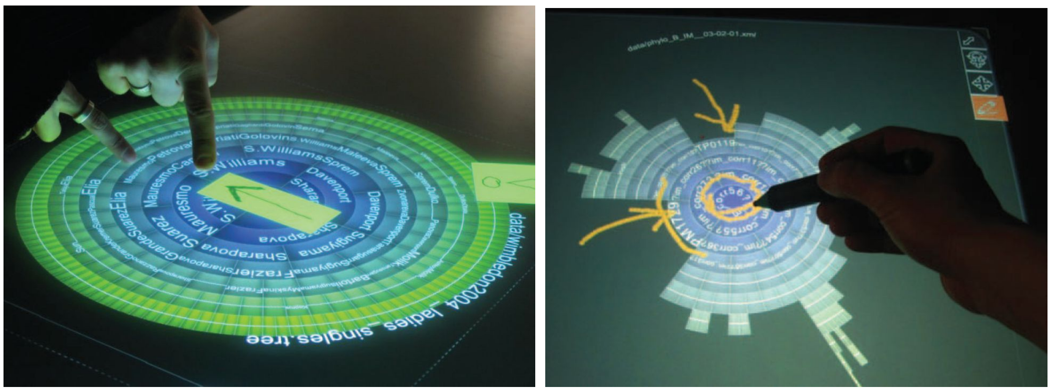


Figure 2.16: *Sticky notes and free annotation supported by the collaborative information visualization system of [Isenberg2007].*

[5]http://www.infozoom.com
[6]http://www.spotfire.com

[Park2000] deployed a collaborative environment for visualizing scientific data using a CAVE [Cruz-Neira1992, Cruz-Neira1993] (see Figure 2.17). The user studies showed that most researchers had a common behavior. They used private views to test out small individual hypotheses then used global views to present their findings to the collaboration partners. This behavior also resulted in the user-request to directly share their visualization parameters which requires support for recording and restoring visualization parameters.

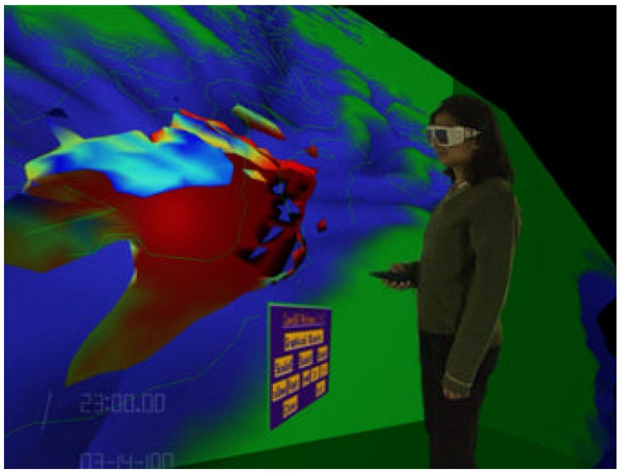


Figure 2.17: *CAVE based environment for scientific data visualization [Park2000].*

2.3 Multi-Display Environments

An often seen setup of a single user computer-workplace is one computer with a single display device. Onward development made larger display devices affordable for standard users. Anyways, the display area of a single display device is rather limited, by physical dimension as well as by the number of used pixels. This often results into the need of temporarily hiding required information while looking at another window which provides also useful information. Enhancing the computer-workplace with more than only one display device is an approach to encounter the lack of display area. As soon as a second display device is used for a computer workplace, it can be called a *Multi-Display Environment*.

As today's state of the art graphics adapters have two independent connections for display devices, dual-monitor setups are already very popular. Typical seen setups are either a standard display connected to a notebook and used side by side with the notebook's built in display or two separated displays connected to a desktop computer.

Beside multi monitor setups for single user workplaces, the research field of MDEs addresses the combination of different kinds of displays like standard monitors, projection walls and tabletop displays. Also, the interoperability of displays connected to different computers and multi-user interaction is analyzed. Furthermore MDEs open broad new possibilities in conjunction with CSCW and InfoVis.

A spatially continuous work space was set up by [Rekimoto1999]. Within this environment the user could smoothly interchange data between their personal portable computers, a tabletop and a wall display by using the so-called hyperdragging feature. Hyperdragging made it possible to move content over the edge of a display onto another display. The location of the portable computer's display and the related user were determined by camera based fiducial marker tracking.

The *iRoom* interactive collaboration workspace with its *iROS* software infrastructure was setup by [Johanson2002]. They used their experimental setup during each day's work for themselves. The setup itself followed a few fundamental design principles:

- Practice what we preach by using *iRoom* during our own work,
- emphasize co-location,
- rely on social conventions,
- aim for wide applicability, and
- keep it simple.

The daily work of the research team and experiments with teams from different working fields brought a positive feedback. The environment supports team meetings with collaborative features. Team members are able to adjust the environment according to the needs and proceeding of the user's tasks. Different devices of individuals like PDAs, workstation and laptops can be integrated within the environment. Figure 2.18 shows the floor plan and a photo of the meeting room of the iRoom.

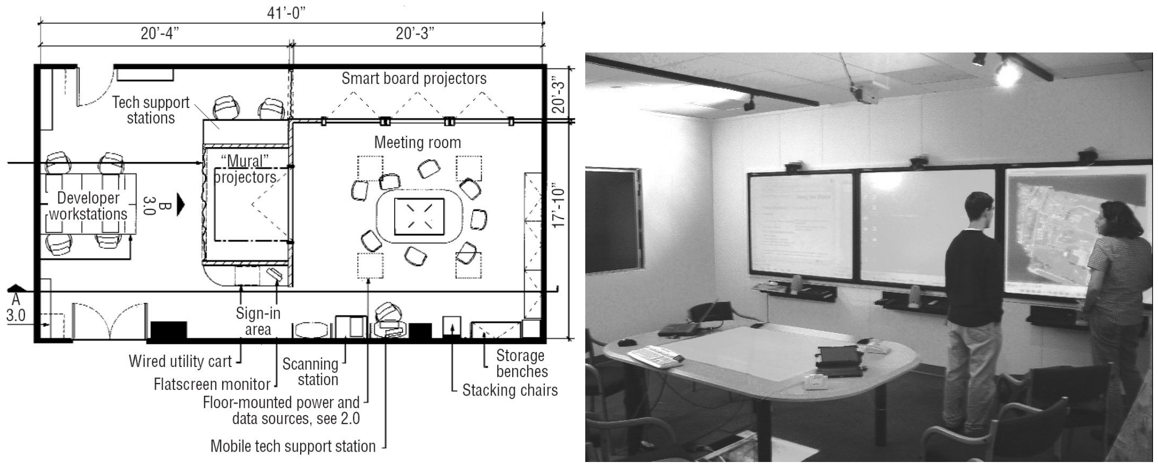


Figure 2.18: *Floor plan and meeting room of the iRoom [Johanson2002].*

Existing software needs to be adopted for integration within the iRoom environment. Therefore the iROS meta-operating system was developed. This software ties together devices, each with its own low-level OS. iROS is diavided into three subsystems. The *Event Heap* subsystems forwards and stores messages, known as events, and provides a central event repository for all applications. The *Data Heap* is a central data storage system where all applications may store to or obtain data from. Finally the *iCrafter* provides a system for service advertisement and invocation.

[Convertino2005] build up prototypes for geo-collaborative tasks for user studies. In their setup the collaborators were seperated in different rooms able to communicate over phone and the collaboration software. They identified three important areas of interaction between MDE and CSCW:

- role differentiation using diverse and complementary views,
- distinction between private and public space or content, and
- cognitive resources that are involved in using multiple views visualizations in individual vs. collaborative work contexts.

[Biehl2008] introduced IMPROMPTU, an interaction framework for collaboration in MDEs. Their focus in the user studies was on collaborative software development. The teams involved have been working together over several years. The goal was to to support their well established collaboration behavior with collaboration tools. Therefore the framework supported showing and sharing windows to other individuals or on public displays, telepointers and concurrent editing. Figure 2.19 shows screenshots of user interaction elements of IMPROMPTU.

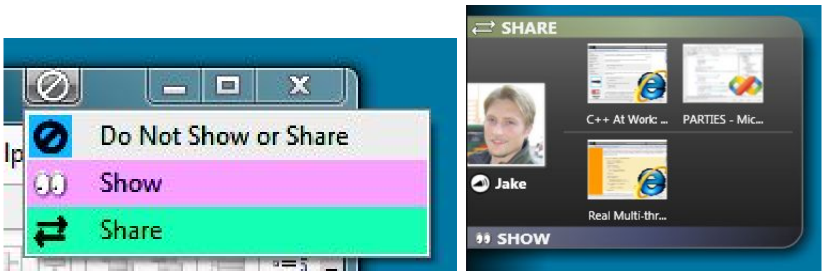


Figure 2.19: *Closeups of IMPROMPTU's window-share-controls and collaboration drawer [Biehl2008].*

The study showed, that the user utilized the framework for many collaboration tasks. The users' feedback was, that they wanted to continue using the collaborative features for teamwork.

Deskotheque

[Pirchheim2009] introduced *Deskotheque*[7], an MDE to combine personal and projected displays into a continuous teamspace. Deskotheque supports geometric display compensation for distorted and overlapping projections, seamless mouse pointer navigation, and window redirection. Furthermore, the display areas are distinguished into private and public with appropriate restrictions for the users.

Dekotheque setups consist of multiple workstations with one or more connected displays and a Deskotheque master application. The Deskotheque master application is the central control instance for the workstations. The master application receives and distributes events, triggered by the workstations.

Deskotheque manages a 3D-model of the working environment. This 3D-model is built with the help of structured light and cameras. The 3D-model contains the geometrical and topo-

[7]http://studierstube.icg.tu-graz.ac.at/deskotheque/

logical information about all the displays. An example setup and its 3D model is shown in Figure 2.20. This information is used for two components of Deskotheque:

Geometric Compensation : A 3D compositing window manager is used to compensate distorted and tiled projection areas. This component is implemented as plugin to the window manager and therefore transparent for applications running in the MDE.

Continuous Workspace : A modified version of the Synergy[8] mouse pointer sharing tool utilizes the 3D model to provide a spatially consistent mouse pointer navigation. Window sharing is supported by *xmove* [Solomita1994], a pseudo-server implementation.

Each of the workstations may be assigned to a user. Each user might have one private displays, which cannot be operated by other users. All other displays are public displays which can be used without restrictions. Users are able to open windows and move windows from or to any display they have access to. For example might *User A* ask his or her co-worker, to have a look at a document and drags this document from *User A*'s private display to a public projection wall. *User B* retrieves the document from the projection wall by dragging it to his or her private display. Exchanging documents in this form might be seen less abstract and closer to used behavior as for example sending an email with the document as attachment. Furthermore the application state of the application that is used for displaying the document does not change during the movement operations across display boundaries. This means that the currently displayed page of a document is not changed when moving it from one to another display.

The implementation of the continuous workspace mainly builds upon modifications and add-ons to the windowing system of the workstations. Therefore no need for modification of the employed applications is necessary. On the other hand, some drawbacks come along with this approach. The most noticeable disadvantage is, that all windows of one application can only exist onto one single display. For example it is not possible that an office application displays its document on one display while the toolbars are displayed on another. Another disadvantage is the close binding to the window manager. This makes the installation of updates of the window manager and the underlying operating system rather difficult and often results in additional modifications of the source code of the used components.

2.4 Software Engineering

2.4.1 Distributed Applications

Requirements to the responsiveness of applications makes it necessary that within distributed groupware applications, each application holds a copy of the current data state. It is not tolerated by users that they would have to wait on network requests when performing, for example, a single scroll operation. Groupware software running on multiple computers used by multiple users and synchronized over network rises the problem of concurrency. Let us consider a groupware application running at two sites. Each of the site

[8] `http://synergy2.sourceforge.net/`

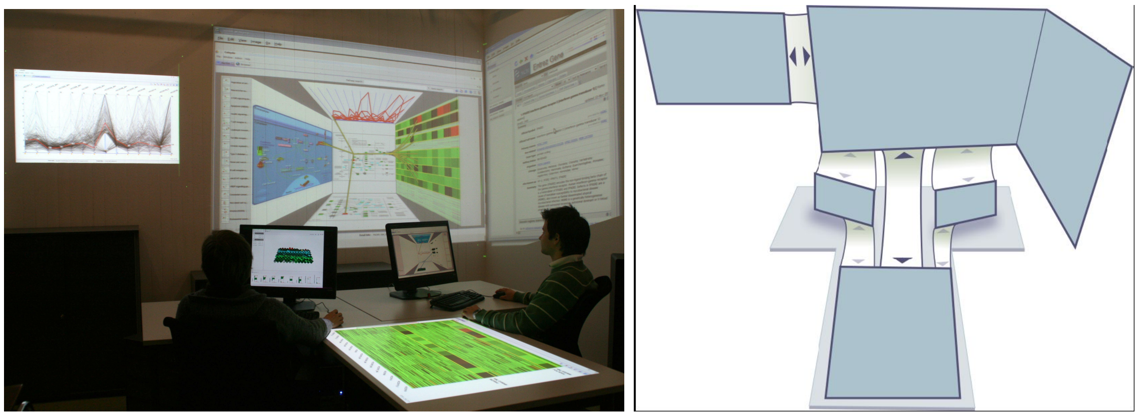


Figure 2.20: *Example setup of Deskotheque [Waldner2009] and 3D model of this setup including mouse pointer paths[Pirchheim2009].*

holds the whole state of the application. All changes at a site are applied immediately and also transmitted over the network to the other site. At each site a change is triggered nearly at the same time. Because of the network latency, each site applies its own change on the data state first and the remote change later. If those changes are not completely independent the two applications might have different states from now on. In the example of a distributed text editor in Figure 2.21.

There are a number of approaches to encounter this problem. Most of them rely on locking or event ordering mechanisms.

Locking means to lock data for exclusive usage by one user before any editing operations may be applied. Locking is a very common method to provide data consistency in distributed database management system (*DBMS*). For example, [Singhal1997] investigates the locking behavior of three DBMSs.

In collaborative systems locking methods have to be combined with the user behavior. Users with active locks may be temporarily inactive, users must wait for other users to release locks [Greif1986, Stefik1987]. In many cases it could be hard to determine which objects have to be locked before a user operation is performed. For example, inserting a character into a word processor does not influence only one word, it may change multiple lines or even multiple pages [Ellis1989].

Events are communication objects used within the common publish/subscribe design pattern [Gamma1995]. The pattern is used to provide a loose coupling of publisher (sender) and subscribers (receivers) of messages. Events are the data objects that encapsulate all data of one single message. There are various possible sources that trigger events, for example a mouse click of a user or the termination of a timer.

Event Ordering approaches have their common root in Lampport's work on virtual clocks [Lamport1978]. During event ordering mechanics, changes on the data are ordered using a global timeline. An instance of the application may execute user operation immediately, but information is kept so changes may be undone. If a new incoming remote operation has a timestamp, that signals that it has to be performed before the recent and already executed operations, those operations are undone. Then the newly arrived operation is

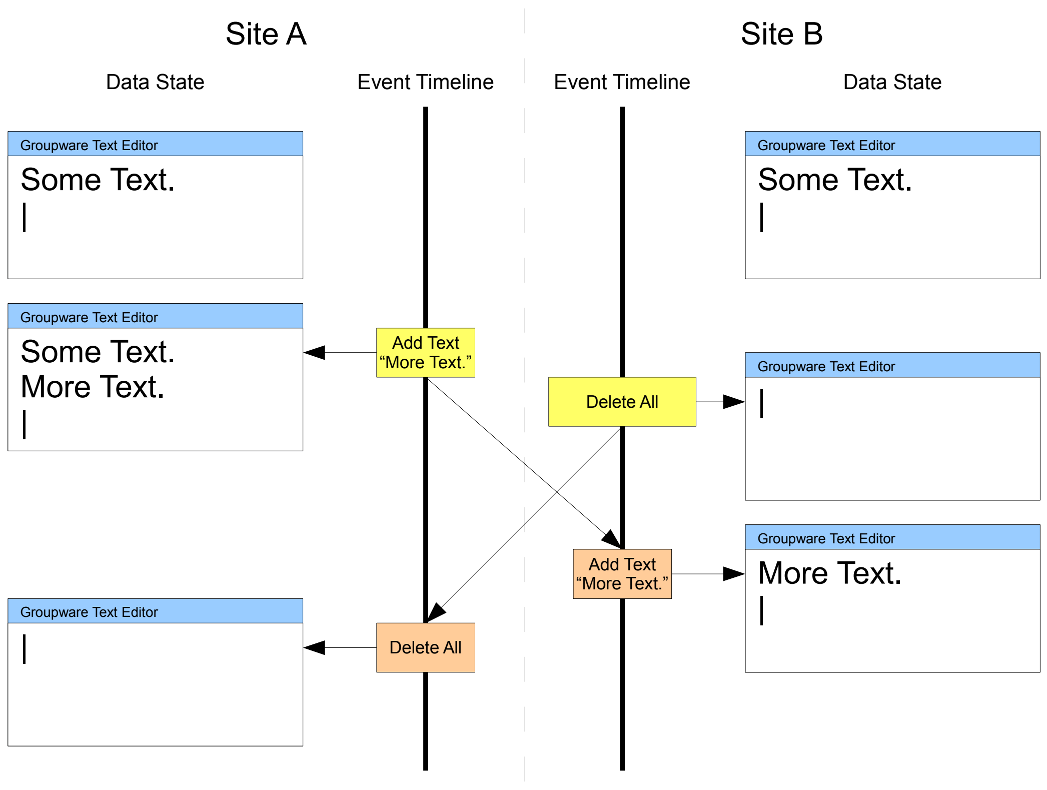


Figure 2.21: *Groupware text editor used at two sites and a concurrent editing problem resulting in different data states, adapted from [Mauve2004].*

performed, and again the previously undone operations are executed. The main disadvantage of this system is, that the result of an operation may already be recognized by a user and afterwards it is undone automatically and replaced with something else which in turn confuses the user.

Varieties of event ordering proposes the idea of local-lag [Mauve2004] or user interaction [Li2000]. Local-lag means that the user operations are never executed immediately. All operations are distributed to all groupware application instances immediately but are delayed to some point within the global valid timeline in the near future. This should ensure, that each operation is executed on all instances at the same time. Figure 2.22 adapts the example from Figure 2.21 by applying a local lag algorithm. In particular the local-lag method has its strength in continuous applications like computer games. In most cases the local-lag time is used to show an animation to the player that simulates the needed actions before a result can be achieved.

User interaction is the idea to let the user interact with the system in case of conflicts instead of solving the conflicts mechanically. If automated event ordering is applied, a user operation might be based on a data state that is completely different than the data state after the event ordering was applied. Therefore the operation might be undesired by the user. The result in Figure 2.21 with applied event ordering is the same as shown for *Site A*, as the *Delete All* operation is the last triggered operation. However, the user at *Site B* might have canceled the operation if he or she would have been informed by the

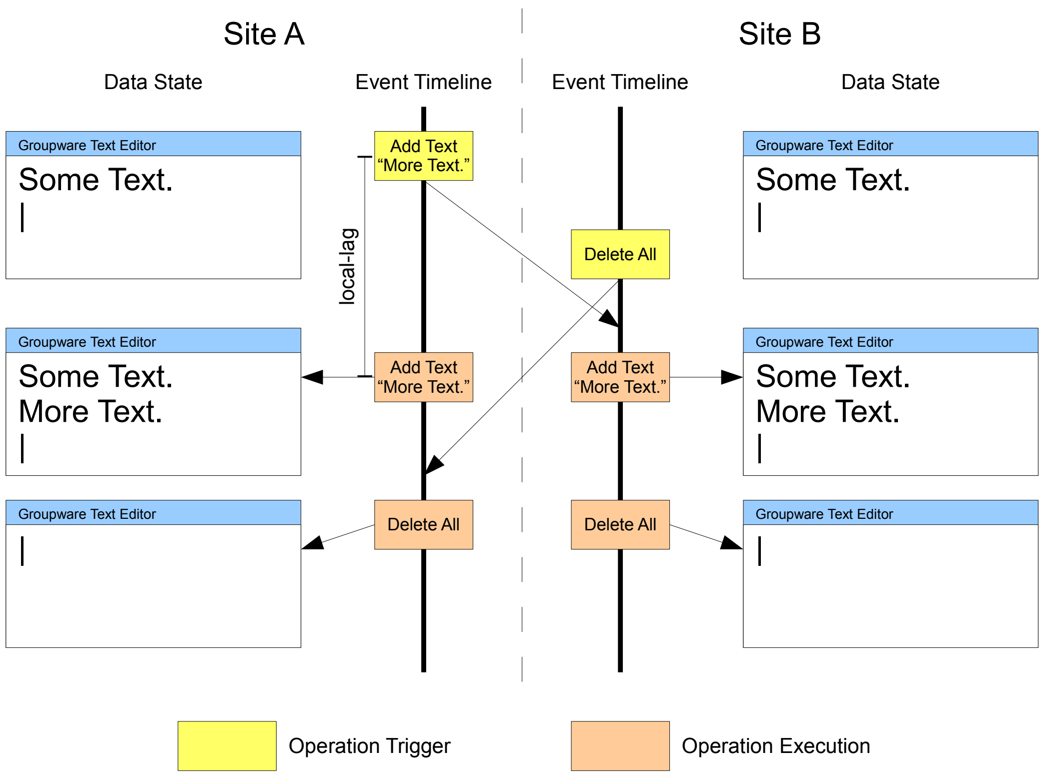


Figure 2.22: *A Groupware text editor with Local-Lag mechanics to counter concurrency problems, adapted from [Mauve2004].*

system that another user has entered some additional text. Therefore [Li2000] proposed to detect and report conflicts to the user and let him or her decide how to resolve the conflict.

2.4.2 Serialization

During the runtime of an application the application's data in memory is usually stored in an object representation. *Serialization* is the process to transform data from its object representation into a sequence of bits so it can be written to stream oriented media like files or transmitted over the network. The reverse process, converting data from its serialized representation to is called deserialization. The terms marshalling and unmarshalling are also often used instead of serialization and deserialization.

In fact serialization of data is an important part within distributed systems to send objects across application and computer boundaries. Within the Java[9] programming language two major serialization approaches are directly supported:

- Binary Serialization
- XML Serialization (see [JAXB2.0])

In binary serialization the serialized data is basically a byte-by-byte copy from the object

[9]`http://java.sun.com`

representation plus some structural information. Nevertheless the serialization framework has to care about system specific differences like, for example, big or low endian representations of numbers.

```
@XmlRootElement
@XmlAccessorType(XmlAccessType.FIELD)
public class Address implements Serializable {
    String street;
    int housenumber;
    String city;
}
```

Figure 2.23: *Example of an address class including annotations for XML serialization and the interface for binary serialization.*

```
0000   ac ed 00 05 73 72 00 07  41 64 64 72 65 73 73 75   |....sr..Addressu|
0010   6b cd 7e 19 c6 71 46 02  00 03 49 00 0b 68 6f 75   |k.~..qF...I..hou|
0020   73 65 6e 75 6d 62 65 72  4c 00 04 63 69 74 79 74   |senumberL..cityt|
0030   00 12 4c 6a 61 76 61 2f  6c 61 6e 67 2f 53 74 72   |..Ljava/lang/Str|
0040   69 6e 67 3b 4c 00 06 73  74 72 65 65 74 71 00 7e   |ing;L..streetq.~|
0050   00 01 78 70 00 00 00 10  74 00 04 47 72 61 7a 74   |..xp....t..Grazt|
0060   00 0b 49 64 6c 68 6f 66  67 61 73 73 65            |..Idlhofgasse|
```

Figure 2.24: *Binary serialized representation of an address-object.*

XML serialization converts the data into an XML-representation. For each field of the object that has to be serialized, an XML-element or XML-attribute must exist in the corresponding XML document.

```
<?xml version="1.0" standalone="yes"?>
<address>
    <street>Idlhofgasse</street>
    <housenumber>16</housenumber>
    <city>Graz</city>
</address>
```

Figure 2.25: *XML serialized representation of an address-object in hexadecimal and ASCII format.*

The major advantage of XML serialization is that XML documents and so the XML serialized form of the objects are human readable. Therefore development and debugging might be more difficult with binary serialization compared to XML serialization. On the other hand XML generation and parsing is a rather expensive process and so binary serialization is much faster than XML serialization. [Hericko2003] compared the performance of Java's and Microsoft .NET's XML and binary serialization. Their measurements showed that binary serialization in Java is five to nine times faster than XML serialization.

With binary serialization in Java the programmer has either to rely completely on the serialization implementation of Java, or the programmer can provide a self written serialization method that is fully responsible for the serialization. Figure 2.23 shows a simple address-class example, the serialized representations are illustrated in Figure 2.24 for binary serialization and Figure 2.25 for XML serialization. Using [JAXB2.0] XML serialization the programmer has more distinct ways to influence the serialization process on different levels. Figure 2.23 shows the most simple way to annotate a Java class for being serializable into an XML document. Moreover the programmer can specify annotations on the whole class or single fields. For example, to specify if a field should end up as an XML-attribute or XML-element in the resulting XML-document the XMLAttribute or XMLElement annotations are used. Furthermore the programmer can even specify a serialization method for each of the fields. Beside human readability this flexibility of Java's XML serialization framework is another advantage compared to binary serialization.

2.4.3 Remote Method Invocation and Remote Procedure Calls

The execution of tasks on remote computer is anything else than a recent problem. [Emmerich2008] provides an overview of commonly used communication middleware technologies and their role within today's IT landscape. Well known specifications are the *Common Object Request Broker Architecture (CORBA)*[10] - a successful industry standard - and SOAP[11] - a common protocol for web services.

Java Remote Method Invocation[12] [JavaRMI] is the native RMI implementation of the Java programming language. *The Internet Communications Engine (Ice)*[13] is an open source RMI framework available for multiple programming languages. Support for communication of applications written in different programming languages is one important feature of Ice. For the development with Ice, the programmer has to write descriptors in an Ice specific language. Programming language specific code generators generates the interface classes from these descriptors.

2.4.4 User Interaction

The model view controller (MVC) design pattern was introduced in [Krasner1988]. This design pattern for object oriented programming languages describes a possible separation into different classes with clearly defined fields of responsibility.

Model: The model is the application's central data structure. It is responsible for the domain specific software simulation.

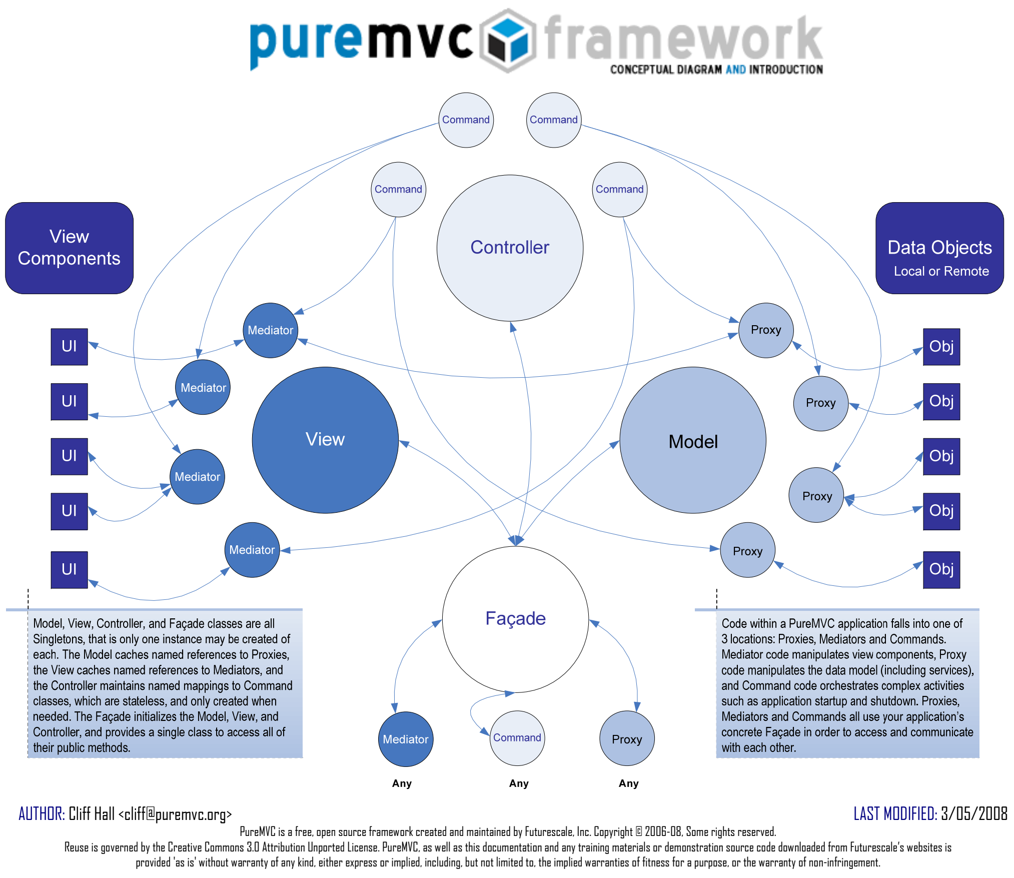
View: The view is the part that deals with graphical issues. It is the representation layer that is responsible to render the data obtained from the model.

[10] `http://www.omg.org/technology/documents/corba_spec_catalog.htm`
[11] `http://www.w3.org/TR/SOAP/`
[12] `http://java.sun.com/javase/technologies/core/basic/rmi/index.jsp`
[13] `http://www.zeroc.com/`

Controller: Controllers are the interface between the model and the view and its input devices. They are responsible to deal with user interactions and handle resulting operations on the model.

Each of model, view and controller should be in a separate class. Typical systems will have multiple models, views, and controller classes.

The MVC design pattern is a broadly used architecture. [Heer2006] underlines the importance of this design patterns for today's object oriented software and information visualization system development. Spring[14] and Struts[15] are very common frameworks for J2EE web applications, that are fully based on this design pattern for any user interaction. The *pureMVC*[16] framework provides MVC implementations for several programming languages. The framework's conceptual architecture is illustrated in Figure 2.26.



Model, View, Controller, and Façade classes are all Singletons, that is only one instance may be created of each. The Model caches named references to Proxies, the View caches named references to Mediators, and the Controller maintains named mappings to Command classes, which are stateless, and only created when needed. The Façade initializes the Model, View, and Controller, and provides a single class to access all of their public methods.

Code within a PureMVC application falls into one of 3 locations: Proxies, Mediators and Commands. Mediator code manipulates view components, Proxy code manipulates the data model (including services), and Command code orchestrates complex activities such as application startup and shutdown. Proxies, Mediators and Commands all use your application's concrete Façade in order to access and communicate with each other.

AUTHOR: Cliff Hall <cliff@puremvc.org> LAST MODIFIED: 3/05/2008

PureMVC is a free, open source framework created and maintained by Futurescale, Inc. Copyright © 2006-08, Some rights reserved.
Reuse is governed by the Creative Commons 3.0 Attribution Unported License. PureMVC, as well as this documentation and any training materials or demonstration source code downloaded from Futurescale's websites is provided 'as is' without warranty of any kind, either express or implied, including, but not limited to, the implied warranties of fitness for a purpose, or the warranty of non-infringement.

Figure 2.26: *Conceptual diagram of the pureMVC framework* (`http://puremvc.org/pages/docs/current/PureMVC_Conceptual_and_Intro.pdf`).

[14]http://www.springsource.org/
[15]http://struts.apache.org/
[16]http://puremvc.org/

Chapter 3

System Architecture

Levering Caleydo from a single user application to a distributed multi user application running in an MDE comes hand in hand with a considerable amount of software re-engineering. Therefore Section 3.1 attends on necessary refactoring within the existing framework. The analysis focuses on extensions needed for a distributed application but also outlines general issues of the current architecture. Section 3.2 builds upon the insights of Section 3.1 and covers needed changes to achieve the goal of this work.

3.1 Single User Caleydo

Caleydo is a Java application based on the *Eclipse RCP*[1] framework. The RCP framework bases on the open source *Standard Widget Toolkit (SWT)*[2] which is used for the standard 2D graphical user interface. The main Information Visualization related tasks are implemented in views rendered with the implementation of the *Java Binding for the OpenGL API (JOGL)*[3]. A screenshot containing both, JOGL and SWT rendered sections is shown in Figure 3.1. See Section 4.1 for a more detailed description on used technologies and their role within Caleydo.

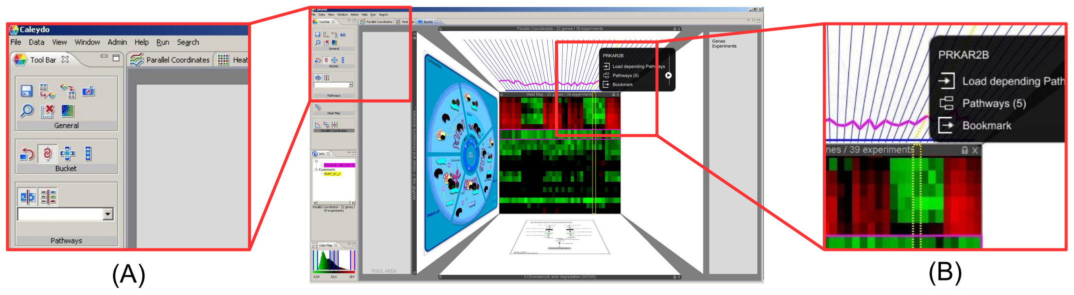


Figure 3.1: *2D GUI-elements and 3D GUI- and InfoVis-elements and InfoVis of Caleydo. (A) SWT provides the overall user interface like windows and widgets, (B) JOGL/OpenGL is used for sophisticated Information Visualization tasks.*

The combination of SWT and JOGL results in two main rendering and execution threads within the application: One for the SWT event-handling and rendering loop and one for

[1]http://www.eclipse.org/home/categories/rcp.php
[2]http://www.eclipse.org/swt/
[3]http://www.jcp.org/en/jsr/detail?id=231

the OpenGL display and event-listening loop. Both threads are executed simultanouesly within a running Caleydo application.

3.1.1 Modules

The Caleydo framework consists of several distinct modules.

Application Startup: This module is responsible for retrieving and utilizing the initial start parameters from the user, like the data set to load and the application mode to run. Furthermore this module loads pathway data from supported internet pathway databases and stores it in a local disk cache. After all the module initializes the SWT and JOGL frameworks and starts the other Caleydo modules.

View Manager: The View Manager is capable of adding and removing views from and to the application and manages resources needed by the views. Each view is a separated sub-module. Views are divided into 2D-SWT views and 3D-OpenGL views. Additional views can be added to the Caleydo framework by fulfilling the provided view API. The most important supported view types of Caleydo are described in detail in Section 2.1.2.

Pathway Manager : The Pathway Manager loads the locally cached pathway data and manages the references among them. This includes the mapping of different IDs of distinct databases (see also the description of metabolic pathways in Section 2.1.2).

Event System : This module is responsible for event distribution of events received by registered event triggers to registered event receivers.

Command System : The Command System is capable of executing actions in form of serializable commands. Furthermore commands have the option to be undo-able.

Data Management : This module holds the loaded gene expression data set and is responsible for changes performed on the data and its distribution to the other modules.

Selection Manager : This is the central instance to store user selections and publish them to visualization modules.

Connected-Element Manager : This module stores view related coordinates of selected elements for Visual Link rendering.

3.1.2 Communication

As Caleydo is divided into modules, communication between the modules is a must. Selections performed within one view should be propagated to all views holding entities directly related to the selection. Filters, that are applied by using the interface of one particular view, have to be propagated to all views. Three different ways of inter-module-communication are used by the current implementation: *commands*, *events* and *method calls*.

3.1.2.1 Commands

Caleydo's commands are implementations of the Command Pattern [Gamma1995]. According to this design pattern, a command object represents and holds the information for a method call. Bundling the information of a method call in one command object facilitates the implementation of a multi-level-undo feature. This is achieved by storing a history of all executed commands and an additional undo-method for each of the commands. For each undo step one command is taken from the command history and its undo method is executed.

Commands are heavily used during the startup process of Caleydo. According to the chosen application mode a file holding all related commands is loaded and the commands are performed. The undo-feature in Caleydo is not implemented. Because of the complexity of the executed processes of some commands they do not implement the needed undo-method for this feature. Furhermore some of the commands implement execution-methods with a rather complex degree of execution-logic which exceeds the idea of the original command pattern.

3.1.2.2 Events

Events are the communication objects used within in the publish/subscribe design pattern (see Section 2.4.1). The propagation of events is performed by so called event systems. Each, SWT and JOGL/OpenGL provide an event system of their own. These event systems focus on the events needed and triggered by the frameworks. Because of the need of asynchronous inter-module communication, Caleydo cannot rely on only one of the event systems provided by these frameworks. Caleydo features its own event system implementation for asynchronous communication between the modules. This event system supports multiple event divisions. Each event sender and receiver has to register explicitly for one or more of the event divisions. According to this registration, events from one sender are only distributed to the receivers which are registered at the same event division and of course for the particular subscribed event types.

Events are handled by the event system immediately in the thread of the event trigger. As mentioned at the beginning of this section, two main execution threads are running within a Caleydo application, one for the SWT framework and one for JOGL. Each of these threads manages data that does not provide any synchronizing mechanism. Triggering an event from one execution thread that has a module with data related to the other execution thread raises the problem of access restriction failures. The SWT framework forbids external thread access by threads other than SWT's main execution thread, attempts result in ThreadAccessExceptions. Accessing data related to the JOGL thread is more sophisticated. There is no detection mechanism to avoid data access from other threads. The resulting possible concurrent modification of data might result into a hard reproduceable ConcurrentModificationException or simply undesired behavior of the application.

3.1.2.3 Method Calls

Method calls are the standard way of communication between objects in object oriented programming languages like Java. Thus it is not a big surprise that in Caleydo as a single user application method calls are used frequently for communication between different modules.

3.1.2.4 Compound Communication

Beside the use of the three described communication methods, Caleydo uses compositions of subsets of them.

Event-commands are commands that are encapsulated within an event-object and distributed to other modules via the event system. The receiver unpacks the command object from the event and triggers its execution. As command objects contain execution method for performing the related task, the usage of commands within events puts execution logic into events, even events were originally designed as message entities only. Thus the sender and receiver of such an object are tighter coupled compared to the usage of standard events.

The data update mechanism uses both, method calls and events. One module, for example a view, publishes updated data using method calls to the data management module. Afterwards the module sends a data update event to inform other modules that the existing dataset has undergone a manipulation. The receivers retrieve the updated data again by method calls from the data management module (Figure 3.2).

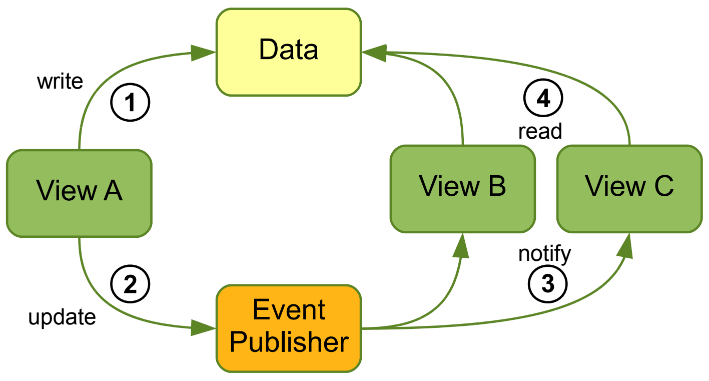


Figure 3.2: *Caleydo data update mechanism. After updating the data with a direct method call other subsystems are informed of the update by sending an event [Streit2007].*

The weakness of this architecture lies in the possible coincidental access to the data set. One module might be in the middle of the update process while another module processes a data update event and starts reading the data set in its currently undefined state. One might argue, that this cannot happen in an application designed running on a single workstation with one single user. However, using this architecture in a distributed multi user system landscape, running into undesired behavior or system failures is inevitable. Another problem that comes hand in hand with a distributed system is that simple method calls are not possible beyond the boundaries of the application without using extensions like remote method invocation (RMI).

3.1.3 Views

The various existing views utilize and combine knowledge and experience from multiple InfoVis research topics. Views like pathways, heatmap, parallel coordinates or radial hierarchy provide the user with the possibility to analyze the data from different perspectives. Each of the specialized views focuses on particular tasks. The compound views, Bucket and Jukebox, set information from multiple views in direct relationship and encourages the user to use the most appropriate view for each of his analysis tasks.

Most of the currently existing views encapsulate display related program logic and data manipulation logic in one single class, except a few tasks that are performed with the help of command objects. This results not only in very huge classes with thousands of lines of code which are rather hard to maintain. The Model-View-Controller (MVC) design pattern [Krasner1988, Gamma1995, Heer2006] demands that objects from different classes take over the operations related to the application domain, the display and the user interaction. Thus it can be stated that Caleydo's classes do not completely follow the MVC design pattern. With regard of a distributed application architecture this means that it is not possible to apply changes only at the level of controllers and exchange of the model as underlying data set. The needed changes have to be applied directly within the view classes and might have side effects on the entire behavior of the view.

3.1.4 Visual Links

Caleydo supports Visual Links in the compound views like the Bucket view [Lex2008, Streit2009]. In these views, Visual Links are a major aid for the user in finding relationships between the entities in the different views. Whenever the user selects an entity in one view, by either hovering the mouse pointer over or clicking it, the view publishes the selection information in combination with view related coordinates of the selected entity to the connected-element manager. Other views are informed of the selection via the event system and also provide their selection coordinates to the connected-element manager. Afterwards the Bucket view retrieves the coordinates from the connected-element manager to render connection lines between all the selected entities of the contained views. Figure 3.3 illustrates the rendering process of Visual Links.

This implementation only works, as long as the visual connection lines are drawn within one single drawing area, in particular the OpenGL canvas of the view. At the start of this work, there is no support to draw connection lines between several independent views. In addition, for currently being a single workstation and single user application Caleydo does not provide any interface for an external application to draw Visual Links across multiple applications.

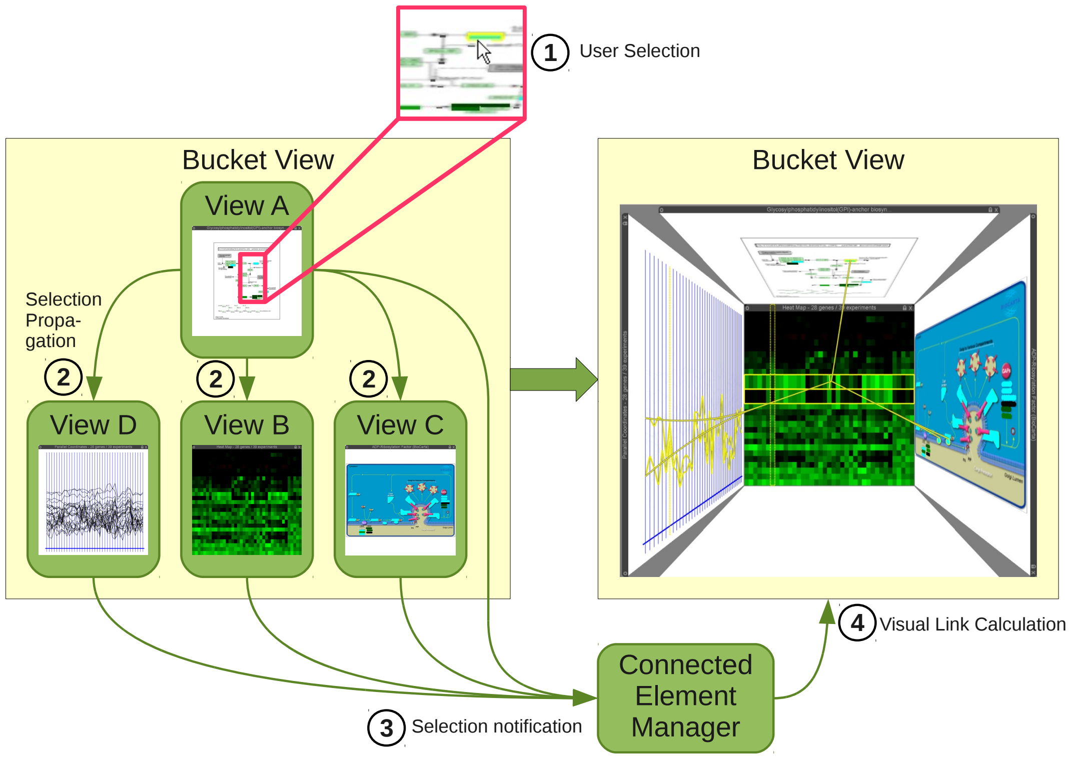


Figure 3.3: *Selection propagation and rendering of Visual Links in Caleydo. A user selection (1) is propagated to all standard views (2) via the event system. Each view delivers the selection related coordinates to the connected-element manager (3). The Bucket view retrieves the visual link information (4) and renders connection lines and contained views.*

3.1.5 Serialization

The data set is read from a serialized representation at startup, in particular from a file from disk. During this process unneeded information is dropped according to the data analysis step of the visualization pipeline (see Section 2.1). Currently there is no possibility to serialize the data set again after it has been read. Furthermore none of the modules supports any kind of serialization to its data. Therefore it is not possible to store a snapshot of the current application state, which could be used for example to continue work later. Command objects are the only entities that can be serialized and deserialized on demand. Therefore, a specialized XML-format exists, the same for all kinds of command objects. Parameters of the command are stored with attribute names attrib1, . . . , attrib4. This means that the number of attributes is limited by the syntax of the XML-format. Currently the limit is four. Furthermore, the files are not easily human readable because the original naming of the attributes is lost.

Figure 3.4 shows an example of a serialized command object. An interpretation of the attribute attrib2="600 600", as it can be seen in this example, could be the size of the view in pixels but also the position of the view on the screen. A valid statement about the meaning of this attribute cannot be made without examination of the source code.

```
<Cmd
    mementoId="0"
    process="RUN_CMD_NOW"
    type="CREATE_VIEW_SWT_GLCANVAS"
    cmdId="69991"
    uniqueId="16401"
    gl_canvas="99075"
    parent="10331"
    label="Canvas"
    attrib2="600 600"
    detail="1077991"
/>
```

Figure 3.4: *XML serialized command object.*

3.1.6 Multi Selection Support

The existing selection manager is not restricted to handle only one selection. Actually it handles any number of selections. The single user use case is supported by mouse over and mouse click selections. Figure 2.11 shows a Bucket view with two selections - the selection resulting from the recent mouse click colored in purple and the current mouse over selection in yellow. The same differentiation mechanism can be used to distinguish between selections of multiple users within a distributed Caleydo application landscape.

3.2 Multi User Caleydo

The major goal of this work is to support a distributed version of the Information Visualization system Caleydo. Support for

- distributed synchronous collaboration
- co-located synchronous collaboration within the Multi-Display Environment Deskotheque

should be achieved. Asynchronous collaboration is not directly supported. However asynchronous collaboration is possible with external tools for communication and synchronization of persisted data.

For a distributed synchronous application each user owns a workstation and runs a Caleydo application instance. Displaying the different windows of a single application on displays controlled by different workstation is not possible with Deskotheque due to implementation limitations (see Section 2.3). Each workstation has to run its own application instance for co-located synchronous collaboration. Both use cases require multiple application instances that have to be synchronized over network to support collaboration.

This section covers refactoring and streamlining needs of existing modules and the design of two new modules according to the requirements of multiple synchronized application instances and multi-display support. One of the new modules is the network stack (see

Section 3.2.4). This module takes care of all network related communication tasks between multiple Caleydo applications. The second new module is the Deskotheque interface, which is described in Section 3.2.7. Central part of this module is the groupware manager. The groupware manager is the interface between the main application and the Multi-Display Environment. Currently the groupware manager focuses on the interface supported by Deskotheque. A second implementation of the groupware manager provides distributed synchronous collaboration support. As this implementation merely is a simplified version of the Deskotheque related groupware manager, it is also covered by Section 3.2.7. The integration to other collaboration and Multi-Display Environments is possible by additional specialized implementations of the groupware manager.

Generally the system architecture enforces the goal to identify possible independent subsystems within the application and replace existing tight coupling with loose coupling. The major requirement to this new kind of coupling is that all involved communication has to be accomplished via serializable objects. This is necessary to allow modules running on different workstations with communication using a network connection. Following this guideline consequently and thinking of a Multi-Display Environment with consisting of several workstations, a possible setup could be having each of Caleydo's views running on an individual display.

3.2.1 Event System

In a distributed Caleydo application setup, it must be possible to send events over a network. On the one hand this requires that all events that are needed for inter-module communication must be serializeable. Section 3.2.3 deals with this requirement. On the other hand this implies that there must not be any direct dependencies between events and the environment they are processed in. Compound Communication (see Section 3.1.2.4) between modules has to be strictly avoided.

Therefore all data that has to be transported between modules is part of the payload of events. Execution logic based on event processing is part of the receiving entity.

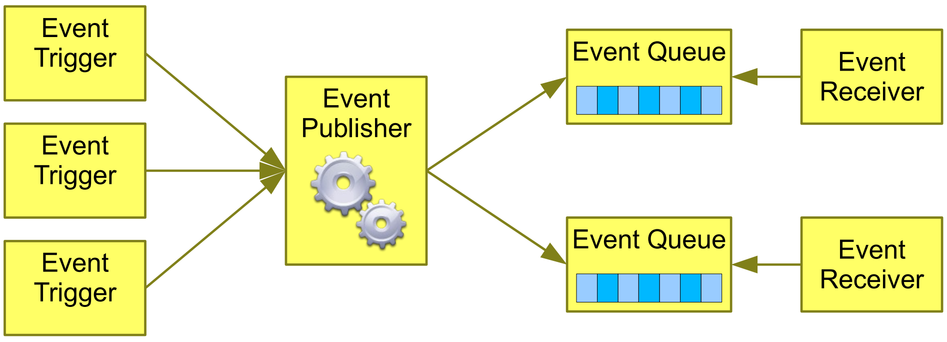


Figure 3.5: *Redesigned Event processing in Caleydo. Events are put into event-queues. Receivers fetch the events from the queues when they are ready for event processing.*

Event queues are introduced for thread safety on data structures that are not designed to provide thread safe access. Each event receiver owns a thread queue, a simple first-in first-out storage data structure. The event system puts the events into this queue. Each receiver gets the events from this queue at a convenient time in its execution. This not only avoids concurrent write access on data structures, but also avoids that data structures are modified in the midst of rendering a view based on this data. Events also provide a sender-attribute. This attribute holds the triggering instance of the event. This is used to avoid possible endless circular event distribution. The overall architecture of the redesigned event system is illustrated in Figure 3.5.

The previously known event divisions (see Section 3.1.2.2) have all been used as a distinctive feature of events. Therefore it is replaced by more specific events and additional parameters within the according event classes. For the explicit definition of particular module groups, where each module of one group should not be directly connected via the event system to any other group, the usage of multiple event publisher is proposed.

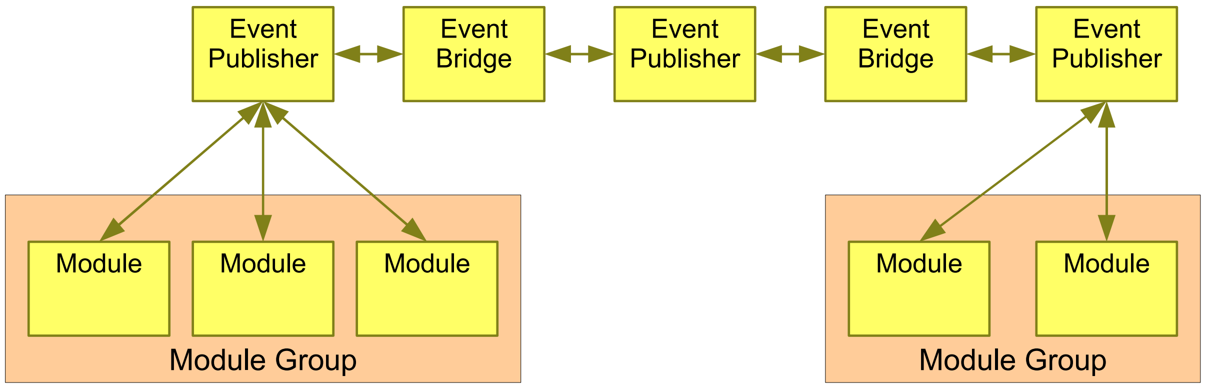


Figure 3.6: *Multiple event publisher instances and module groups.*

Two connected module groups are illustrated in Figure 3.6. The communication within a module group is handled via an event publisher dedicated to the module group. An additional event publisher establishes the communication between the two module groups. An event bridge dedicated to each of the module groups is the main communication interface between its module group and other sub-systems. The event bridge forwards events between the module related and the inter-group-communication event publishers, depending on the subscribed events of each of the event publishers. This design allows to distinguish between events being only of interest within a specific group of modules and events that are of interest for the whole application. For example, a user selection should only be visible within a group of views while the application shutdown should be propagated to all parts of the application.

3.2.2 Synchronization

Because of the introduction of event queues, all receivers need their own execution time frame. In particular this affects manager modules that are not dedicated to one of the main execution threads (see Section 3.1). Therefore a special OpenGL component is introduced. This component is called during OpenGL's display loop and calls the execution methods of all registered manager classes.

OpenGL related modules are registered and executed within the display loop and can make use of this for processing the event queue. SWT related modules utilize the asynchronous execution features of SWT.

3.2.3 Serialization

Serialization is needed to address three issues:

- Module communication across application boundaries
- Application state
- View state

As described in Section 3.2.1, all communication between modules is done by sending and receiving events. Therefore all events that are needed for inter-module communication must be serializeable. Of course this includes the event's data payload as well. As serialization subsystem the JAXB 2.0 (see [JAXB2.0]) implementation included in the Java Runtime Environment 6 is used. There are only few implementation tasks and refactoring issues on existing classes to support JAXB 2.0 serialization. This method of serialization also provides much better human readability compared to binary serialization (Section 2.4.2 provides a comparison).

Furthermore XML formats ensure an easier upward compatibility. With binary serialization, particular code has to be provided to support compatibility across different Java Runtime Environment versions. Another issue when using binary serialization arises with changes related to the serializable classes. For each particular version of an implementation a mapping possibility to the current version has to be provided. Using XML as serialization format version changes can be performed directly on the serialized representation. For example by applying an XSL stylesheet[4] to the XML document.

New Caleydo application clients should be able to connect to a running application cluster at any time. That is why it is important to serialize the application state and send this information to new connected clients for initialization. Beside this direct requirement, an application state serialization feature is a basis for persisting the application state to disk, which is a basic requirement of a save-project mechanism. The initialization based on a serialized application state (as needed by the connecting client application) can be used for a load-project mechanism. The only difference is the data source. For distributed application initialization it is the network, whereas for a load-project feature a file on disk has to be chosen by the user.

Caleydo's startup mechanism uses XML based bootstrap files, which hold a various number of commands. Different bootstrap files exist for different application modes supported

[4]`http://www.w3.org/Style/XSL/`

by Caleydo. More detailed parameters of the application modes are evaluated by these commands during startup.

The application state consists of the application mode, its parameters and a CSV input data file. As used with events, the JAXB 2.0 implementation is used to serialize the aggregated application state.

An important feature for applications running in Multi-Display Environments is that the elements the user interacts with - usually the windows and views - can freely be placed anywhere in the environment. The Deskotheque setup features a workstation cluster. Each workstation is connected and responsible for only a subset of the existing display devices. Therefore, moving a Caleydo view from one display to another requires the support to move views between individual Caleydo applications. See Section 3.2.7 for a more detailed description of the interaction between Caleydo and Deskotheque. This is achieved by the possibility to serialize each of Caleydo's views and sending it over the network as payload of an appropriate event. Moreover, view serialization mechanism can be used in conjunction with application state serialization within the load- and save-project feature. See Section 3.2.5 for a more detailed description of loading and saving of project files.

3.2.4 Network Stack

For keeping the particular instances of a distributed application environment, synchronized network communication is inevitable. Caleydo is based on a client-server network architecture (see Figure 2.15). One Caleydo application (the server) is the central communication node for all other applications (the clients) in the network. Application data is completely redistributed to all running instances and synchronized according to user operations. The server is responsible to distribute all events to the clients.

The network module itself is initialized on demand by the user. A running Caleydo application can be switched to server mode at any time. The data set to work with is loaded during the startup and initialization of the application. This startup behavior requires to choose the client application mode already during application start. The client application mode causes Caleydo to connect to the server and retrieve application mode and the dataset from the server.

Client and server architecture is very similar. Each connection between two Caleydo applications depends on two threads, one to handle incoming and one to handle outgoing network traffic. Of course, a client only opens one connection, while the server holds one connection to each of the connected clients. Furthermore the server starts a dedicated thread to listen for incoming connections.

The network stack utilizes the architecture of the event system by using multiple event publisher instances. A global network related event bridge listens for events, that might be of interest for remote applications and delivers them to an event publisher dedicated to network connections. Connection specific event bridges listen to events on the network dedicated event publisher and pass them to the connection specific network thread. This is how local events are propagated to remote applications. A similar architecture provides reception of remote events. The connection specific network thread related to incoming traffic receives an event from the network. The event is passed via a connection specific event bridge to the network related event publisher. Another event bridge listens to this network event publisher and passes received events to the global event publisher to which other modules are connected to. On the server, incoming events are also propagated to the outgoing network event publisher and thus passed to all connected clients. See Figure 3.7 for the overall design of the network stack.

Event bridges support an event filtering mechanism. This filter is parametrized to not bridge events back to their sender. This avoids endless circular event propagation. The different event bridges may subscribe to a different set of events. While the global network event bridges subscribe to all events that might be of interest for remote applications, the connection specific event bridges can have client specific event subscription sets. In a Multi-Display Environment a client connected to a touch display device can be used to display toolbars only. Accordingly, this client only needs to listen to view activation and selection events to display related buttons and options for context sensitive manipulation.

3.2.5 Persistence

This section outlines the requirements and mechanics to save and load an application state of Caledyo. This feature can be easily achieved as a side effect of the serialization support needed for network communication. To store and rebuild a snapshot of a running Caleydo application, the aggregated application state has to be serialized and written to a file. This aggregation consists of four major parts:

- application mode parameters
- data set
- data transformation
- view states

The application mode and its parameters describe the general usage of the Caleydo application. Section 3.2.3 describes the aggregation and serialization possibility of the application mode parameters in detail.

The data set, as for example the gene expression information of a number of experiments, is stored in tabular form, usually as a comma separated values (CSV) file [Streit2007]. These files provide a lot more information than currently needed by Caleydo, hence the information has to be filtered in a preprocessing step. Possible future features of Caleydo might utilize currently unneeded information. If the filtered data would be stored within project files, the additional information would be lost for the future. This is one reason why the data set as provided by the original file is stored within project files. Another reason is that by storing the same file, the same and already existing reading processes can be reused to load the data set again.

During the runtime of Caleydo the user can apply various transformation operations onto the data set. For example the application of brushes or manual reordering and duplication of experiments. [Lex2008] describes that the results of such operations do not manipulate

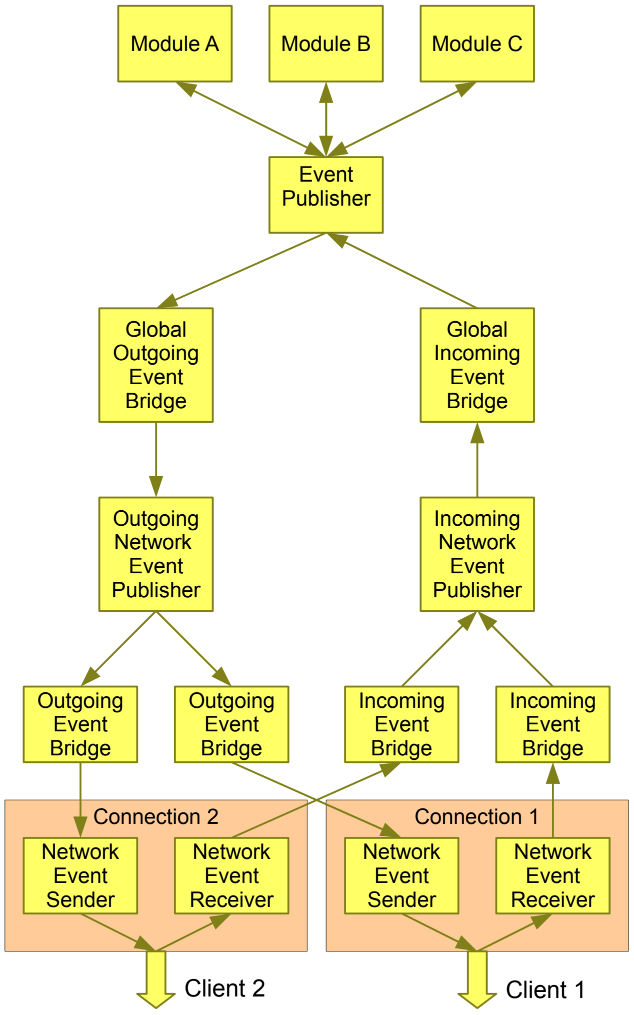


Figure 3.7: *Network stack using multiple event publishers. Two remote clients are connected in this example. Network event senders and receivers each run as separated threads.*

the original data set. So called *virtual arrays* map the original data sets to the form visible and utilized by the application (see Figure 3.8).

[Schlegl2009] introduced clustering algorithms for a reordering of the data set. Clustering is a calculated reordering of the data rows, columns, or both, based on the gene expression data itself. Cluster algorithms try to reorder the data in a way, that similar kind of data rows, respectively columns, are grouped close to each other. Moreover some of the supported algorithms provide additional hierarchical information on the data set. Both, the virtual array information and the hierarchical cluster information are stored within project files.

Each of the provided views has a number of attributes defining its state. Examples are the arrangement of the particular sub-views within the Bucket, used brushes within the parallel coordinates and the currently displayed level within the hierarchical cluster information in a radial view. Window position, size and arrangement within the Eclipse

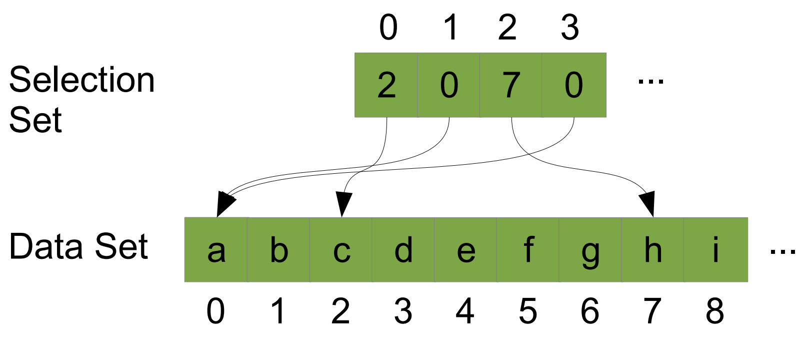


Figure 3.8: *A virtual array maps data from the original data set to a selection set [Lex2008].*

RCP environment are also important parameters that characterize views. The Eclipse RCP framework supports software developers with an API for view serialization and reconstruction of the views on application restart. Unfortunately, the provided mechanism focuses on one single application state only and is therefore not feasible for loading and saving project files. Anyway, this might not be a major drawback as we assume that a single user prefers to work within the same view arrangement independently from the investigated data set. Hence, general view parameters like position, size and arrangement are reused across multiple Caleydo sessions with the help of Eclipse RCP features, while the more specific and usually data set related view information is stored with the data sets.

All four kinds of application state information are stored within separated files of a zip archive.

3.2.6 Visual Links

As described in Section 2.2, change and activity awareness are major aspects of collaboration systems to establish a shared mental model of the collaborators. An MDE offers a much larger display area than single displays and therefore multiple different high resolution views on the data set can be displayed simultaneously. On the other hand, a user might not be aware of changes and relationships of entities within the different views. Out hypothesis is that Visual Links support the user in finding relationships that might be unrecognized or only recognized at a later time without their aid.

Rendering Visual Links within a single view of one application can be done directly in the view or in a separated render step. Visual Links in a distributed environment rise an additional technical challenge. Lines have to be rendered across application and workstation boundaries. Deskotheque already supports line rendering across multiple displays. This feature is currently used to show mouse pointer paths and display relationships. The same feature can be used to draw connection lines between multiple Caleydo applications, each one running on one of the displays provided by Deskotheque. Therefore Caleydo has to deliver screen coordinates to Deskotheque.

Caleydo's connected-element representation manager is the central instance to collect user selections. A single view receives selection events that may either be the result of processing an appropriate user interaction or by retrieving a selection update event from another module. Each time a view receives such an event, the view looks up possible entities displayed by the view and calculates their view coordinates. These coordinates are passed to the connected-element representation manager, which stores them along with the selection information. Compound views apply additional 3D transformations on the views, to place them on their appropriate place within the compound views. Therefore the compound views also apply these transformations to the stored selection coordinates of the particular views. After processing all needed 3D transformations the views are projected onto the 2D canvas. The same projection is applied to the selection coordinates. The 2D canvas coordinates are transformed into screen coordinates. At this step of the process, a selection is stored with a number of 2D screen coordinates.

In a distributed environment these actions are not limited to one application. Each selection is parameterized with a unique identifier and therefore each selection can be clearly identified independent of the application instance. Each client within a distributed application environment sends these screen coordinates to the server application. The server application delivers the aggregated selection coordinates to Deskotheque, which is responsible for drawing the connection lines. Figure 3.9 illustrates the overall process in a chart.

3.2.7 Deskotheque Interface

The MDE Deskotheque consists of a variable number of workstations (Deskotheque modules), each with one or more connected displays. All workstations are registered at the central Deskotheque master application, that works as a supervisor within a Deskotheque setup. A so called Deskotheque server application is running on each of the workstations as local managing instance. Deskotheque uses the Ice middleware (see Section 2.4.3) for communication between applications. Therefore, Deskotheque provides methods for communication with applications like Caleydo also via Ice.

The main part of Caleydo's Deskotheque interface is the groupware manager, which provides a communication interface between Caleydo and the Multi-Display Environment. The groupware manager is responsible for setting up and shutting down the network stack.

During the initial registration process each Caleydo instance establishes a connection to the local Deskotheque module. A connection to the master application is available through each of the Deskotheque modules and the registration of the Caleydo application as groupware application is performed. During registration each application is assigned a unique identifier, which has to be provided with subsequent interface method invocations. This process is passed by each of the application instances of Caleydo because it is required by the interface provided by Deskotheque. Figure 3.10 shows the system architecture with a setup of three Caleydo applications. The Caleydo applications operate in client-server mode, while Deskotheque has a mixed setup of peer-to-peer Deskotheque modules and an additional master application as control instance.

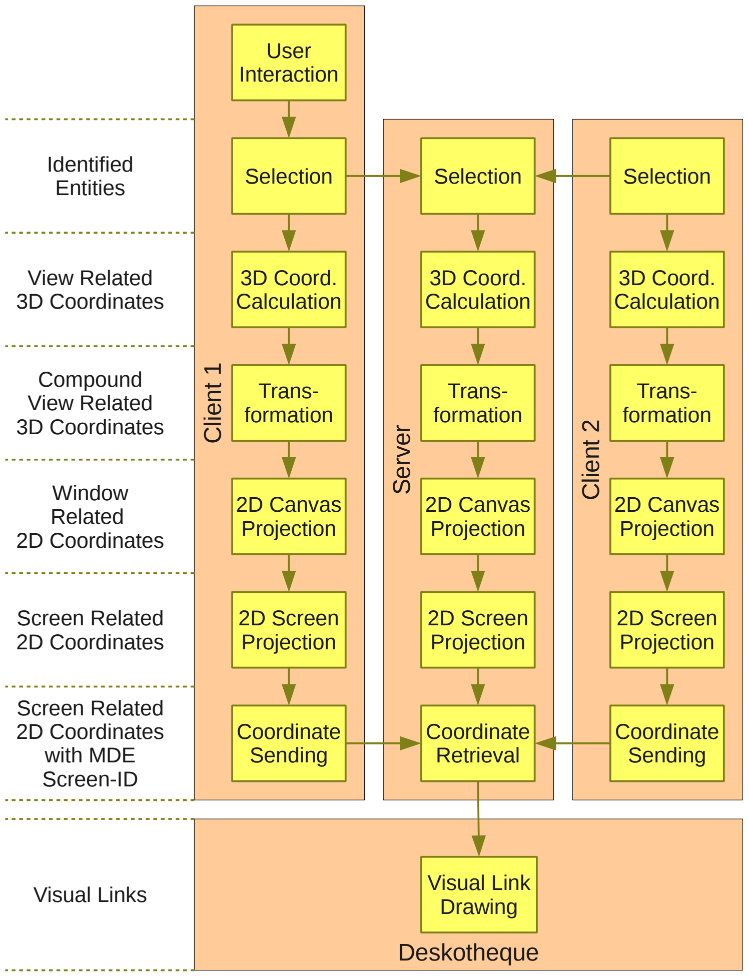


Figure 3.9: *Visual Link processing within three connected Caleydo applications. The process starts with the User Interaction at the top which causes a selection. The selection is propagated to all application instances which individually calculate the resulting screen coordinates of the selected entities within visible views. The screen coordinates are collected at the server and sent to Deskotheque for Visual Link visualization. The left part of the diagram shows the available information at the appropriate state of the process.*

The groupware manager provides methods for the registration process and differentiates between client and server mode. The Deskotheque registration process is the same for clients and servers. However, the network stack has to be setup in a different way and a client has to retrieve the application state from the server (see Section 3.2.4).

Furthermore, the groupware manager also provides the methods for the collaboration related tasks for view sending between connected Caleydo applications. Each display in the Deskotheque MDE is classified as either public or private. In usual setups, the private displays are the monitors directly in front of each user's workplace and the public displays are various projected and tabletop displays in the near spatial environment of the user workplaces. See Figure 2.20 for an example setup. The classification in public and private

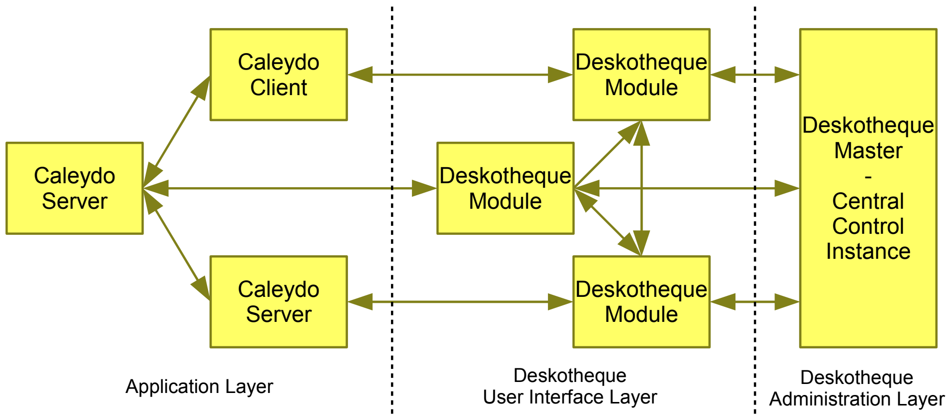


Figure 3.10: *Overall system architecture of three Caleydo applications in a Deskotheque setup.*

displays provides desired private areas for users when working in a collaborative working spaces.

Two use cases related to Caleydo views are derived from the display classifications:

Publish View: A user may publish a view currently displayed on one of his private views. The environment takes care of finding a free public display area. Caleydo closes the source view and opens a view with identical view parameters within the destination application that is related to the free public display.

Retrieve View: A user requests a view displayed on a public display to be sent to his private display. The view is moved from its current place to one of the users private views.

These use cases are considered high level, as the implementation has to evaluate the environment setup and its current state. Beside these use cases users are offered a low level *move view to specific display* option. This option lets the users directly specify the target display for a chosen view. The three different view sending methods provided by Deskotheque are directly available by methods of the groupware manager.

As described in Section 3.2.6 Deskotheque is responsible for visual link rendering across application and display boundaries. Before Deskotheque is advised to draw the connection lines, the screen coordinates and display identifiers from views of all application instances have to be collected by the server application. The groupware manager of each of the applications collects the screen coordinates of the views within the application and sends them to the groupware manager running on the server application. Afterwards the groupware manager on the server passes the coordinates to Deskotheque.

Beside co-located synchronous collaboration with MDE support Caleydo provides support for a distributed synchronous setup. This setup differs from an MDE in the spatial arrangement of the workstations and that using the particular displays as one continuous workspace does not make sense, because the user cannot see each other's displays. Furthermore, the distributed setup has no public displays devices. Therefore the *publish view* and *retrieve view* options are not supported by this groupware manager implementation, but the direct sending of a view to a chosen client is still possible. Rendering of visual links is also not possible in distributed setups and not supported by the related groupware manager implementation.

Chapter 4

Implementation

This chapter explains the particular implementation tasks performed on the Caleydo framework. Section 4.1 describes the libraries and tools that are directly related to the implementations. The subsequent sections describe the changed classes and developed implementations in detail. Beside describing their overall way of operation and interaction within the framework also its usage and role in future extensions are explained. The complete framework constis of about 1000 classes, thus unchanged existing implementations are omitted.

Note: As Caleydo's classnames are unambiguous, all Caleydo related classnames are denoted without their package-path. Described classes from other libraries contain the full package path to clearly outline their framework membership. Most of the time parameter types and names of methods are omitted, as they usually do not provide explicit information to understand the implementation itself. However, detailed documentation about methods and their required parameters and return values can be looked up within the source code or Javadoc[1] documentation.

4.1 Used Technologies

Java Development Kit 6

The Caleydo framework is completely written in the Java programming language[2]. The current Caleydo implementation uses version 6 of Java. As for the concept of Java to execute the programs in a portable Runtime Environment, Caleydo can be executed independently of operating system versions of different kind as long as the workstation comes with an installed Java Runtime Environment. This results in an easier roll-out of the application to test users, as no operating system issues have to be concerned about by the development team.

SWT, Eclipse RCP and RCP Plugins

The current Caleydo implementation makes use of Eclipse' Rich Client Platform (Eclipse RCP)[3]. This framework provides a set of modules for the implementation of classic desktop

[1]Javadoc is a tool for automated HTML documentation generation out of Java source code, `http://java.sun.com/j2se/javadoc/`

[2]`http://java.sun.org`

[3]`http://www.eclipse.org/home/categories/rcp.php`

applications. It is completely based on Eclipse' Standard Widget Toolkit (SWT)[4]. The open source SWT library is a portable toolkit for user-interface facilities of different operating systems. Implementations exist for all common operating systems and they are still maintained. Eclipse RCP provides the programmer with a window management system that supports typical features of today's GUI based applications like window tabs and arrangement. Caleydo's view management is based on Eclipse RCP's window management. Another important feature of RCP is the possibility to separate modules into so called Plugins. Plugins are function sets with a well defined interface to the main application or to other plugins and can be loaded dynamically on demand.

Eclipse IDE

One of the most popular applications built onto the RCP framework is the Eclipse Integrated Development Environment (IDE)[5]. Originally developed as a Java IDE, Eclipse IDE meanwhile supports a broad range of programming languages and software engineering tools, as for example subversion. In most cases the particular tools are realized by the the implementations of a RCP plugin. Caleydo is completely developed with the Eclipse IDE.

JOGL

Java Binding for the OpenGL API (JOGL)[6] is a wrapper between Java and the OpenGL 3D API. It provides direct access to OpenGL functions from inside Java programs and therefore does not suffer from performance issues that might come with Java programs for not being native executables.

Ice

Ice[7] (Internet Communications Engine) is an open source RMI implementation with support for serveral programming languages including C++ and Java. A comparison between Ice and other RMI technologies can be found in (see Section 2.4.3). As Deskotheque is already implemented using Ice for communication between applications, Ice is used as communication framework for the Caleydo-Deskotheque interface.

JAXB

An implementation of the Java Architecture for XML Binding (JAXB) 2.0 specification [JAXB2.0] comes along with Java 6. A more detailed description of the advantages of using JAXB compared to other possible serialization methods can be found in Section 3.2.3.

[4]`http://www.eclipse.org/swt/`
[5]`http://www.eclipse.org`
[6]`http://www.jcp.org/en/jsr/detail?id=231`
[7]`http://www.zeroc.com`

4.2 Event System

The event system follows the publish/subscribe design pattern, which is explained in Section 3.1.2.2. The core classes are:

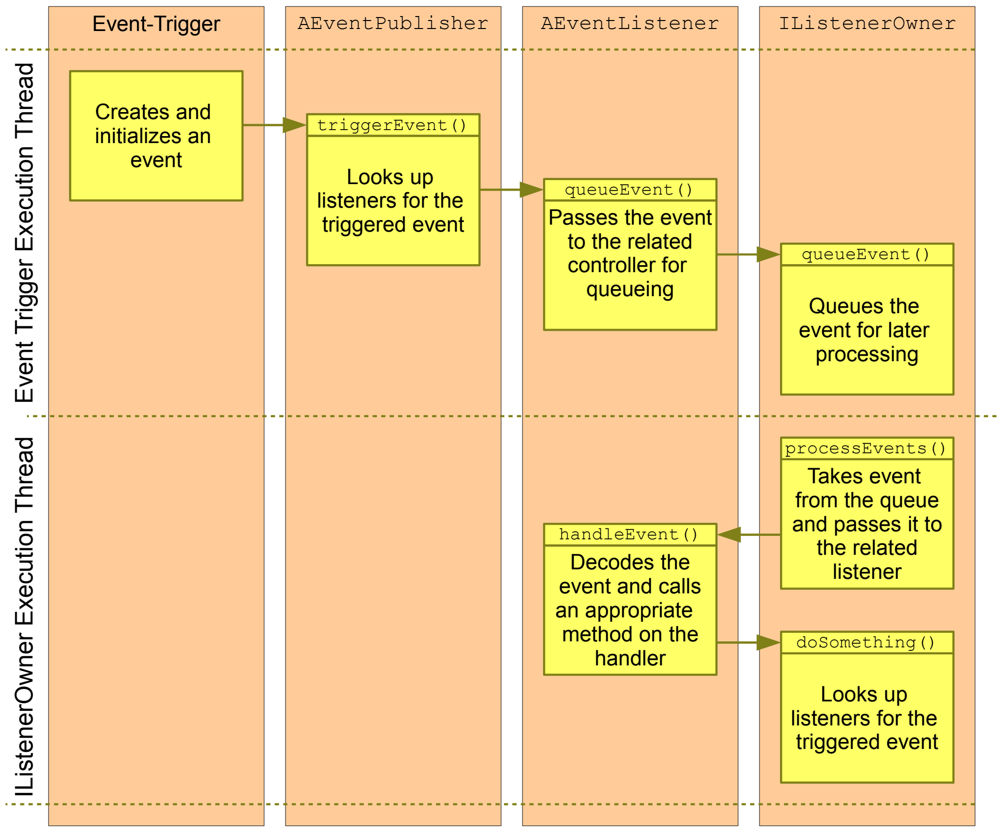
`EventPublisher`: This is the central class which is responsible to process triggered events by propagating them to subsribed listeners. The main `EventPublisher` instance is obtained via the `GeneralManager`, the central singleton hub to obtain access to the particular modules. The `EventPublisher` holds a list of subscribed event listeners, each one directly mapped to the events it is listening to. Listeners have to register themself with the `addListener()` method. To unsubscribe from a particular event or from all events one of the *removeListener()* methods has to be used. Listeners must unsubscribe from the event system, when they are not needed anymore. This happens for example during the disposal of a recently closed view. The propagation of an event is performed by calling the `triggerEvent()` method.

`AEvent`: This is the abstract parent class for all kind of events that should be distributed via the event system. It is policy that the publishing class of an event sets the `AEvent`'s `sender` attribute. The `sender` attribute is evaluated by the `EventPublisher` to prohibit, that events are sent back to its creator. The `creationTime` attribute is set on event creation. Events should not be reused. A new event class for each event has to be created to avoid undefined behavior. Event classes should always be JAXB-serializable, so they can be serialized and sent to remote applications via network. To guarantee data integrity, derived event classes may provide a `checkIntegrity` method that validates the event's payload.

`AEventListener`: Classes that register at the `EventPublisher` to listen to events have to be derived from this abstract parent class. Each event listening module has a number of such listeners to listen to particular events. The controllerSection 2.4.4 of the module must be assigned with the `setHandler()` method of `AEventListeners`. Controllers have to implement the *IListenerOwner* interface, which is described in the next paragraph. This attribute is compared with the `AEvent`'s sender attribute to avoid that events are sent back to its source class. Derived classes must provide a `handleEvent()` method that is responsible for decoding the event and calling appropriate event handling methods of the assigned handler.

`IListenerOwner`: This interface has to be implemented by event handling controllers of modules. Its only method, `queueEvent()`, must be implemented to put events into a thread safe queue for later processing. Eclipse SWT related controllers perform this processing with the help of the `org.eclipse.swt.widgets.Display.asyncExec()` method to execute code within the SWT execution thread. The usual implementation for OpenGL related controllers can be found in `AGLEventListener`. All JOGL based views are derived from this abstract parent class. Special event handling classes might have a non queueing implementation of the `queueEvent()` method. For example the `EventFilterBridge` of the network stack directly passes events to a connected `EventPublisher` instance.

The overall event handling process across the involved classes is illustrated in Figure 4.1.



Event-Trigger
AEventPublisher
AEventListener
IListenerOwner
Event Trigger Execution Thread
IListenerOwner Execution Thread
Creates and initializes an event
triggerEvent()
Looks up listeners for the triggered event
queueEvent()
Passes the event to the related controller for queueing
queueEvent()
Queues the event for later processing
processEvents()
Takes event from the queue and passes it to the related listener
handleEvent()
Decodes the event and calls an appropriate method on the handler
doSomething()
Looks up listeners for the triggered event

Figure 4.1: *Event processing across classes. The x-axis shows the particular classes involved during event processing. The y-axis corresponds to processing time and is separated in two sections related to specific threads.*

4.3 Serialization

To serialize a class to its XML representation one of the different `marshal()` methods of a `javax.xml.bind.Marshaller` has to be called. An appropriate instance of this class has to be obtained with `javax.xml.bind.JAXBContext.createMarshaller()`. The `JAXBContext` instance has to be created with its static `newInstance()` method, which gets a list of classes as parameter. This list contains all classes that should be serializable by the created `JAXBContext` instance. Referenced classes of the serializable root-classes are not needed within this list, as those references are resolved by the JAXB framework automatically.

The `SerializationManager` takes care about serializable classes within the framework. The manager support getter-methods that deliver `JAXBContext` instances for three use cases:

`getEventContext()` : Gets the context that is utilized for network communication (see Section 4.4).

`getProjectContext()` : Gets the context that is needed for loading and saving projects.

`getViewContext()` : Gets the context that is needed to serialize views for Eclipse' automated view persistence mechanism. This mechanism is explained later in this section.

The `ProjectSaver` class provides the methods to save the aggregated application state. The `saveRecentProject()`-method saves to a predefined position in the user's home directory. It is called automatically on application shutdown but also in definable intervals. The `ProjectSaver` provides a second method to save to a specified file (`save()`). The file name is obtained via a standard file-dialog.

The counterpart of the `ProjectSaver` is the `ProjectLoader` class. It can be utilized during application startup to rebuild a previously saved application state. This works by using the contained startup parameters for bootstrapping, the gene expression data as data set and the serialized view information to reopen and initialize views. Therefore the `ProjectLoader` class provides the `load()` and `loadRecentProject()` methods.

The Eclipse RCP framework features also a view persistence mechanism. The framework uses a callback method on each RCP-view which is called during shutdown. This callback method may return a various number of serializable key-value pairs. This pairs are stored in the user's home directory along with unique identifiers that RCP requires from its view anyways. On application startup this information is again read from the user's home directory and all views that were been displayed during the last shutdown are reopened and reinitialized with the provided key-value-pairs. Caleydo has an implementation for these alternate application persistence mechanism.

4.3.1 Event Serialization

Events are simple classes that may hold some data payload. Holding execution logic or references to complex and not serializable data structures is prohibited for events. Because of this policy on events it is sufficient to add the appropriate JAXB-annotations[8] to them and their payload holding classes to provide JAXB serialization of events.

4.3.2 View Serialization

Each of Caleydo's views consists of several classes with more or less high complexity, execution logic, display logic and references to managers and centrally stored data structures. This is why separated classes for serialization of views are introduced. This separates the complexity of views from their serializable representation. To make views serializable, three implementation steps have to be performed.

`ASerializedView`: Each serializable view gets his own `ASerializedView`-derived class. This class contains all fields that characterize the state of the view. Its no-argument default constructor must initialize the object with the default view parameters. They are used for opening a new view of this kind. The `getViewGUIID()`-implementation

[8]javax.xml.bind.annotation package

must return the view-identifier as it is needed by the RCP-framework. Finally, appropriate JAXB-annotations have to be added to provide JAXB serialization support. Furthermore the class has to be added to the list of classes in the `ASerializedView`'s `@XMLSeeAlso`-annotation. This is necessary for the JAXB-framework to resolve the class hierarchy.

```
@XmlType
@XmlRootElement
public class SerializedBrowserView {
  public static final String GUI_ID = "org.caleydo.rcp.views.swt.
      HTMLBrowserView";
  private String url;
  public SerializedBrowserView() {
    url = "http://www.caleydo.org";
  }
  public String getUrl() {
    return url;
  }
  public void setUrl(String url) {
    this.url = url;
  }
  @Override
  public String getViewGUIID() {
    return GUI_ID;
  }
}
```

Figure 4.2: *`ASerializedView` implementation of a browser view. The only attribute used here to characterize the browser is the URL of the currently displayed web-page which is stored in the `url`-attribute.*

`getSerializableRepresentation()`: The `AGLEventListener` of the view has to implement this method. It has to create an instance of the previously described `ASerializedView`-implementation of the view and initialize it with its current state.

`initFromSerializableRepresentation()`: The `AGLEventListener` of the view has to implement this method. This method is the counterpart of `getSerializableRepresentation()`. It initializes the view from a provided `ASerializedView` object.

The resulting implementation are used for all kind of view creation within Caleydo. There are various possibilities when and how an application is requested to open a view:

- The default views are opened on startup of the application.
- The user chooses to open a view with the view-menu.
- A project file containing serialized views is loaded.
- A view is reopened with the help of Eclipse RCP view persistence mechanism
- An open-view event containing a serialized view is retrieved from the network

All this use cases for opening views are handled with the implementations described above. First, the view's state is obtained as an `ASerializedView` instance, either with the default initialization from its constructor or from its XML-serialized form from disk or network. Then the view is created and initialized with the obtained view state.

4.4 Network Stack

The network stack is responsible for establishing connections between particular application instances and for communication via existing connections. The core class is the `NetworkManager`. This manager class is the central access point for networking related issues. Its main methods are:

`startNetworkServices(), stopNetworkServices():` The network framework is not started by default on startup of the application. Caleydo is currently mainly used as single user application. Initialization of network related classes would only slow down the startup process and consume additional resources. The start-method is called on demand to initialize the network framework. The tasks performed during the start of the network services are:

- Create the network related event publishers.
- Create event bridges between the central and the network related event systems.
- Create the JAXB related objects for serialization purposes.

The role of the network related event bridges and publishers are described in Section 3.2.4. The stop-method is called to close all possibly open connections and dispose all resources and is usually called on application shutdown.

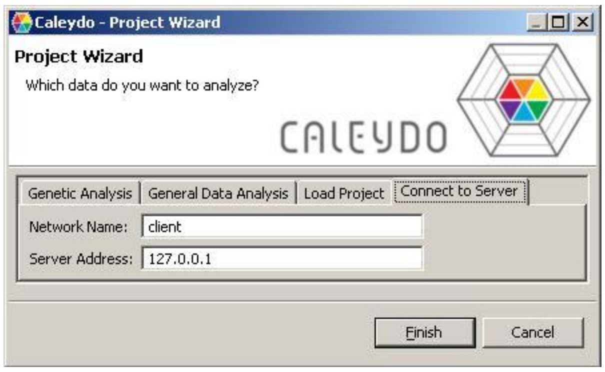
`startServer():` One instance of a distributed Caleydo application has to be used as server. This method has to be called only on the server to start a thread that listens for incoming connections. Any incoming connection are passed to the `createConnection(java.net.Socket)` method, which is described below.

`createConnection(java.net.Socket):` This method performs a handshake with a connecting client on the provided socket. The handshake includes the validation of application versions and the assignment of a client-identifier. Running Caleydo in a Deskotheque setup the client-identifier is retrieved from Deskotheque. Otherwise the client-identifier is created by the Caleydo server application. On a successful handshake, the current application state is sent to the client. This includes the application startup parameters and the used data set. No view information is provided to clients, as clients have to arrange and initialize their views for themself according to the available display area.
The initialization of the connection specific event bridges and the network event sender and receiver are also triggered by this method. The network event sender and receivers are provided with their own threads they are running in. If the sender would not be provided with its own thread, the sending of events over the network would have to be done within the thread of the sender. If two sender would send events at the same time, the resulting two events would be merged into one and such undefined network traffic cannot be evaluated by the receiver. The receiver uses a blocking read on the socket, which demands also an own thread. Serialization

issues are also executed by these threads. Therefore possible available CPU-cores are utilized for serialization tasks without slowing down the main application.

`createConnection(java.net.IndetAddress)`: This method is called on client applications at the beginning of the application startup. The difference to the `createConnection(java.net.Socket)` method described above is, that this method initiates the connection to a running server application and works as counterpart for the handshake process. It delivers the `ApplicationInitData` object received from the server as return parameter, which is needed to perform a similar application startup as on the server.

`getEventBridgeConfiguration()`: This method retrieves the global or a connection specific configuration of an event bridge by delivering all event classes the particular event bridge is listening to. This method can be used in combination with the global or central network event publisher instances to reconfigure the events which are received and transmitted to either all or specific remote applications.


Caleydo - Project Wizard

Project Wizard

Which data do you want to analyze?

Network Name: client

Server Address: 127.0.0.1

Figure 4.3: *Dialog shown on startup to connect the application to a running Caleydo server.*

Each established connection between two Caleydo applications results in one `Connection` object. This object is a collection of all needed resources of one particular client-server connection. This includes the client specific event bridges as well as sender and receiver threads. As each client can only be connected to one server, there can only exist one `Connection` instance on each client. However, one instance exists on the server application for each of the connected clients. This instances are also managed by the `NetworkManager` and a `java.util.List<Connection>` containing all existing `Connection` objects can be obtained via its `getConnections()` method.

4.5 Visual Links

The implementation of Visual Links across multiple applications is split into multiple calculation steps. These steps are described in Section 3.2.6 and shown in detail in Figure 3.9.

In the first step a particular view is informed of a selection. The source of the selection can either be an interaction of the user with the view or a selection event received from another view. In either case the selected entities have to be identified and located. The associated view related 3D coordinates in object space are provided to the `ConnectedElementRepresentationManager`, the central object of each application instance to store Visual Link information. This object stores the source connection point coordinates in conjunction with the unique selection identifier, which is determined during the originally performed user interaction.

The next step is the transformation of the 3D point from the coordinate system of the source view to the coordinate system of the rendering view. This is only relevant if a view is rendered in a composition of multiple views, like it is done in the Bucket or Hierarchical Heatmap view. The transformation is performed by particular implementations of `ISelectionTransformer`. Currently, two implementations of this interface exist. The `RemoteRenderingTransformer` performs this operations for Bucket and Jukebox view. The `StandardTransformer` performs the copy operations for planar views that does not need to transform their connection points. An implementation for the Hierarchical Heatmap view is currently not supported. The results of this step are directly used to render the 3D connection lines within the Bucket and Jukebox view. The following steps involve rather expensive 3D projection operations. Therefore, they are only executed if the Caleydo application is running within a Multi-Display Environment that can take use of the 2D connection line information.

In step three on the way to get 2D screen coordinates the 3D coordinates have to be projected to the OpenGL canvas of the view. This is done with OpenGL's `GLU.gluProject()`. The results are 2D window related coordinates.

All previously described calculation steps feature OpenGL as 3D API. Therefore each transformation or projection is based on the appropriate state of OpenGL's transformation stack and must be executed within the OpenGL display thread during the correct OpenGL state. The final step is to calculate the screen coordinates from their window coordinates. As for being SWT the utilized window and widget toolkit, the projection from window to screen coordinates involves SWT related resources. These cannot be accessed from outside the SWT execution thread. The calculation is performed with `org.eclipse.swt.widgets.Display.asyncExec()`, which executes a given `java.lang.Runnable()` at a convenient time within the SWT execution thread. The projection itself is done using the SWT method `Composite.toDisplay()`. This method converts 2D coordinates related to a `Composite` to its screen coordinates. The drawing area of SWT windows are of type `Composite`. Thus the related composite of the OpenGL canvas has to be determined and the `toDisplay()` method invoked delivers the desired screen coordinates.

After their calculation, the screen coordinates and the associated application identifiers are sent to the server application, encapsulated within an `AddConnectionLineVerticesEvent`. The calculation on the server triggers the sending of the same event. However, instead of sending it over the network it is evaluated within the application itself. The server collects all retrieved `AddConnectionLineVerticesEvents` and passes them to Deskotheque for Visual Link rendering.

Due to network behavior and differences of the calculation speed of different workstations, the arrival time of the events cannot be determined in advance. Thats why the Visual

Link coordinates are checked once during each pass of the display loop. Every time additional coordinates are retrieved or a selection is canceled, an update of the Visual Link information is sent to Deskotheque.

4.6 Deskotheque Interface

This section describes the implemented Caleydo-Deskotheque interface to use Caleydo for co-located synchronous collaboration. As the distributed synchronous implementation is merely a subset of the features of the co-located case, it is described in this section, too.

Beside being a separate module the Deskotheque interface is also implemented as a separated RCP-plugin (see Section 4.1). This makes it possible to release Caleydo versions with or without Deskotheque support or even load Deskotheque support during the runtime of the application. RCP plugins can either be completely independent or they can use a dedicated plugin interface to enhance an application on defined extension points. As the Deskotheque interface provides specialized multi-user features that have to interact with the main application, the latter is needed.

`IGroupwareManager` interface is the extension point used by the Deskotheque interface plugin. This interface has one implementation contained within the main application for the multi user support without MDE support. This is the `StandardGroupwareManager` class. The second implementation is provided by the Deskotheque plugin. This is the `DeskothequeManager` class. This implementation uses the Ice interface provided by Deskotheque (see Figure 4.4). Data structures used during Ice method calls are also defined within the Ice configuration files (see Figure 4.5).

The `IGroupwareManager` interface defines the following methods:

`startServer()` and `startClient()`: These methods are called to use the application as server or client. They perform all required tasks like for example the initialization of the network stack as described in Section 4.4.

`stop()`: This is the counterpart to `startServer()` and `startClient()`. It stops all groupware services and frees the related resources.

`getInitData()`: This method is only used on clients. It returns an `ApplicationInitData` object as it is retrieved from the server, required to startup the client similar to the server.

`getNetworkManager() and setNetworkManager()`: Getter and setter methods for the `NetworkManager`. As described in the Section 4.4, the `NetworkManager` plays an important role for collaboration. Access to it is required for communication configuration issues.

`getHomeGroupwareClient()`: Retrieves the client identifier of the Caleydo application of the user that performed the most recent user interaction. This information is provided by Deskotheque and can only be used with the `DeskothequeManager` implementation.

```
interface MasterApplicationI extends ApplicationI {

  ...

  // registers a groupware application, for example one Caleydo instance
  GroupwareInformation registerGroupwareClient(
    GroupwareClientAppI* client, string id,
    ServerApplicationI* serverApp, int x, int y, int w, int h);

  // draws VisLinks between the given 2D screen coordinates
  void drawConnectionLine(ConnectionLineVertices vertices, int selectionID);
};

interface ResourceManagerI {

  ...

  // Retrieves all available groupware client IDs
  StringList getAvailableGroupwareClients(string clientID);

  // gets an ID of a client running on the current user's private display
  string getHomeGroupwareClient(string clientID);

  // gets a free public display to move a view to
  string getPublicGroupwareClient(string clientID);

  // logs off the given groupware client
  void unregisterGroupwareClient(string clientID);
};
```

Figure 4.4: *Ice descriptor of the groupware interface provided by Deskotheque.The "..." denotes additional methods not involved in the groupware application interface.*

`getPublicGroupwareClient():` Retrieves one client identifier of an Caleydo application that uses a public display. This information is provided by Deskotheque and can only be used with the `DeskothequeManager` implementation.

`getAvailableGroupwareClients():` Retrieves a list of all Caleydo applications connected to the system. This information is either provided by Deskotheque if the `DeskothequeManager` is used or the by network manager if the `StandardGroupwareManager` is used.

`isGroupwareConnectionLinesEnabled():` Returns a boolean variable that tells if the used `IGroupwareManager` implementation supports the drawing of connection lines. This information is evaluated in advance of 3D-view-related to 2D-screen-related coordinate transformation.

`sendConnectionLines():` This method delivers a list of 2D screen coordinates including their related client identifier to a possibly connected Multi-Display Environment. In case of the `DeskothequeManager` this is, of course, the Deskotheque.

The collaboration related view control widgets are implemented as Eclipse RCP chevron

```
// single vertex for visual link rendering
struct ConnectionLineVertex {
  // groupware client that provides this vertex
  string clientID;

  // screen x-coordinate
  int x;

  // screen y-coordinate
  int y;
};

// registration information provided to groupware clients
struct GroupwareInformation {
  // unique display ID in Deskotheque
  int displayID;

  // indicates whether the display is private
  bool isPrivate;

  // identification string of the groupware
  string groupwareID;

  // Deskotheque unique XID consisting of server-hostname + window-X-ID
  string deskoXID;
};
```

Figure 4.5: *Ice descriptor of the data transport classes utilized by the Deskotheque interface.*

menu. This menu is intended for context sensitive actions on views (see Figure 4.6).

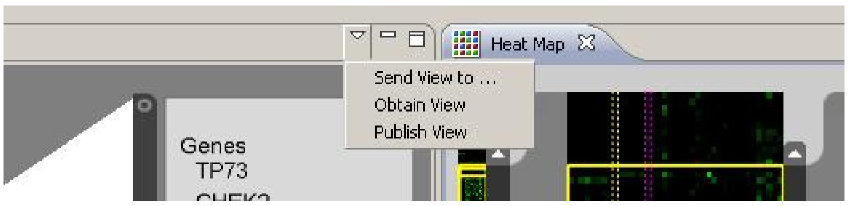


Figure 4.6: *Eclipse RCP's so called chevron menu is used for the initiation of view related collaboration tasks.*

The Deskotheque environment supports currently only Ubuntu 7.04 as operating system. A meaningful setup consists of several workstation. One setup is established in a laboratory at the Institute for Computer Graphics and Vision of the University of Technology in Graz. The development of an application in this environment involves deployment, start and shutdown on each of the workstations for each turnaround cycle. Furthermore, the setup is needed by other researchers as well. To make development easier and for being independent from the operating system a mock-up implementation of Deskotheque featuring only the required parts of the interface is introduced with this work. Same as Deskotheque this mock-up application uses also Ice for RMI access. In contrast to Deskotheque it does not support multiple workstations or multiple users. Multiple displays connected to a single workstation can be used, but cannot be differentiated as separated display areas. The whole display area of the workstation is used as one single display area. A workplace with a dual monitor setup was used most times during the development phase. The arrangement of multiple applications across the provided display area was sufficient for all the development tasks.

The implementation of the view related methods of the mock-up deliver predefined return values. The server application is always taken as the client for the home display (`getHomeGroupwareClient()`), while the first connected client is returned as available public display (`getPublicGroupwareClient()`). Visual Links are supported with the help of an transparent overlay window on the workspace. The mock-up does not use any line bundling algorithm. Visual connection lines are rendered as red lines from the first to each of the other provided screen coordinates.

Chapter 5

Results

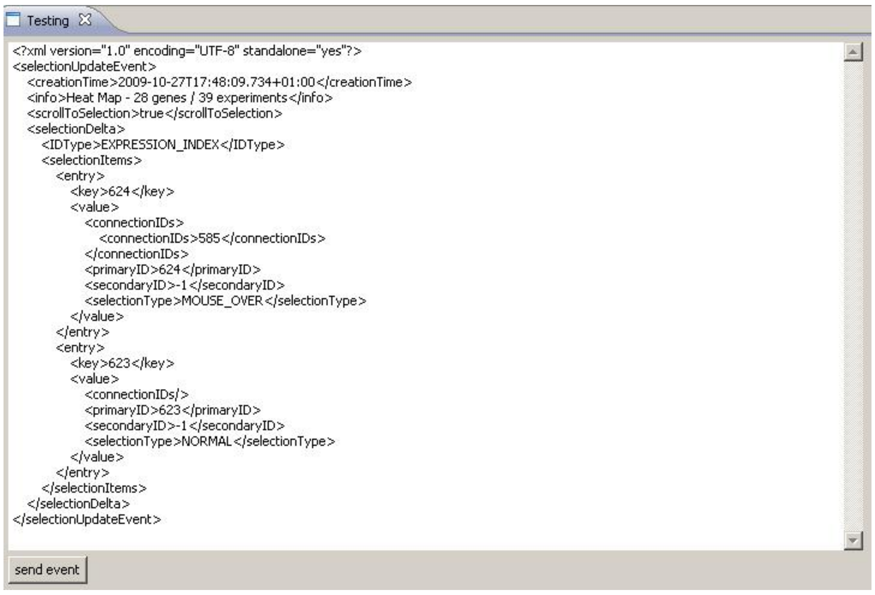
In this chapter the findings of this work are presented. Section 5.1 and Section 5.2 describe the benefits resulting from the applied refactoring process of components of the Caleydo framework implementation. Experiences with newly provided features are covered by Section 5.3 and Section 5.4. The two sections at the end of this chapter concern themselves with resulting collaboration possibilities.

Caleydo is being developed in cooperation with the Institute of Pathology at the Medical University of Graz. Feedback from researchers of this institute are a major source for user experience analysis for the development. The development results of this work are not yet part of a Caleydo release. Therefore, we have no user feedback from domain specific researchers. The feedback provided in the following sections are given by the developers of the Caleydo application.

5.1 Event System

The new implementation of the event system forces developers to strictly seperate events for different processes. The seperation in different event distribution divisions has been dropped. This makes it easier for a developer who inspects the application's runtime behavior for example for debugging reasons. The occurrance of a particular event implicates directly to the triggered workflows. Event sources or different event trigger mechanisms by choosing a particular event division do not have any side effects on the workflow anymore.

A simple event visualization tool is available for testing and development purposes (see Figure 5.1). Events can be visualized in its serialized XML representation. The input of user-defined XML documents is also possible, to trigger desired events on demand without the use of according user interface elements. For example, this feature makes it possible to seperate the development of GUI elements and execution workflows in two tasks, that can be achieved by different development teams. The teams have to agree on an interface definition, based on the event system and the event's XML representation. Afterwards both can fulfill their part of the development and test it with the provided event investigation tool.



```xml
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<selectionUpdateEvent>
  <creationTime>2009-10-27T17:48:09.734+01:00</creationTime>
  <info>Heat Map - 28 genes / 39 experiments</info>
  <scrollToSelection>true</scrollToSelection>
  <selectionDelta>
    <IDType>EXPRESSION_INDEX</IDType>
    <selectionItems>
      <entry>
        <key>624</key>
        <value>
          <connectionIDs>
            <connectionIDs>585</connectionIDs>
          </connectionIDs>
          <primaryID>624</primaryID>
          <secondaryID>-1</secondaryID>
          <selectionType>MOUSE_OVER</selectionType>
        </value>
      </entry>
      <entry>
        <key>623</key>
        <value>
          <connectionIDs/>
          <primaryID>623</primaryID>
          <secondaryID>-1</secondaryID>
          <selectionType>NORMAL</selectionType>
        </value>
      </entry>
    </selectionItems>
  </selectionDelta>
</selectionUpdateEvent>
```

Figure 5.1: *Event investigation user interface showing an XML representation of a `ViewActivationEvent`. The input field is used to display the most recent event and to enter user defined events that can be propagated to the application by pressing the send event button.*

5.2 Synchronization

Recent Caleydo releases suffered from the problem of concurrent access to data structures that are not thread safe. The introduction of event queueing together with the thread dependent processing of events within views is the first step to a stable application. The second step is adding event queues that move the event processing to defined threads for all data managing classes. Because of these changes concurrent modification exceptions are a matter of the past.

5.3 Application State Persistence

The addition of new features to the application often comes along with some new bugs. Some of them result in an undefined application state that the application cannot recover from. This was often criticized in recent users' feedback. For the users to be able to save the current state and continue at a later time by loading it again, data sets and views of the application need to be serialized. This feature is expected to lead to a much higher user acceptance of Caleydo, as there is no more frustration for restarting an analysis process from scratch after an application failure. The auto save feature provides an even higher safety for the current project. As the overall save process does not take more than 50ms on standard workstations[1], the auto saving of the project in the background will not be noticed by the user.

[1]measured on a system with an Intel Core2Duo 2.4GHz processor

5.4 Single User Experiences

During most of the development phase, standard setup was to use one or more connected Caleydo applications run by a single user (the developer) on a single workstation. In setups with multiple applications on a single workstation no additional latency due to the sending of events over network was experienced. There were two exceptions:

Heatmap: When hovering the mouse over the tiles in a heatmap a selection update event was sent on each mouse move event received from the GUI. This behavior was not necessary at all, as a selection update event is only necessary if the mouse pointer hovers over an element other than the currently selected one. The vast amount of events did not only result in lag when using a distributed application setup, higher response times were experienced while using a single application as well. This unintentional behavior of the application concerning these selection update events could be tracked down by using the event investigation tool described in Section 5.1. The application was changed to only send events when a selection change occurred.

Parallel Coordinates: The parallel coordinates view provides brushing functions. Applying a brush leads to a filtering of the data set. As Caleydo is capable of handling thousands of gene expressions, a brush that selects all but a few genes from the current view results in the filtering of thousands of data entities. The current implementation calculates the visibility state for each data single entity and puts it in one single event. The resulting XML representation of such an event has a size of multiple kilobytes containing thousands of XML nodes. The serialization and deserialization of XML documents of that size can take up to several seconds. The usage of brushes in the parallel coordinate view of a distributed application setup produces a latency that is usually not tolerated by users. This is especially true when a user tries to manipulate the parameters of a brush. In a single application setup the parameters are applied in real time. On a distributed system the operation leads to a latency of a few seconds per view update on the local as well as on the remote applications. The serialization process slows down the local application while the deserialization process slows down the remote applications.
We suggest to alter the information payload of the event resulting from the brush operation. Not the result of the calculation should be sent but the parameters defining the brush. The calculation itself would then be performed by the respective applications. The filter calculation would be executed after sending or receiving the event and would take a similar amount of execution time on similar workstations. The effort to serialize and deserialize events with thousands of data entities would not be needed anymore.

Visual Links and their rendering within the Deskotheque mock-up has been tested with one and two Caleydo applications on a single workstation with a dual monitor setup. Views have been distributed over the entire display area[2] of 3360x1050 pixels and spatial dimension of 100x30cm. Figure 5.3 shows a screenshot of two Caleydo instances and the mock-up rendering Visual Links. Users reported a better awareness of related entities

[2]using two 22" TFT displays placed side by side

especially on views placed far away from the current mouse pointer position. This statement is underlined by an experience during a testing phase. During tests of the cross application Visual Links, a bug was detected, related to the selection of entities within the pathway view. It was detected, because Visual Links resulting from a specific selection in the heatmap were not always drawn to the pathway view.

Against our expectations, the 2D to 3D projection of Visual Link coordinates does not take unreasonable time. The time consumed for projecting 20 vertices to their 2D position in the OpenGL canvas is below the measurement resolution of 1ms. Based on this fact it can be stated that the calculation of Visual Link coordinates will not have any reasonable impact on the overall performance of the application.

5.5 Deskotheque

At the time of the final state of this thesis, Deskotheque did not yet provide the required support of the defined Caleydo-Deskotheque interface. Deskotheque development is not within the range of this work, but is assigned to a different research team. Therefore no experiences and user studies of Caleydo running in a multi-user Multi-Display Environment can be provided here. Because of the good experiences concerning improved awareness in dual monitor setups (see Section 5.4) we are looking forward to the full integration of the Caleydo application within Deskotheque.

5.6 Co-located and Distributed Collaboration

To test the influence of network transmission time, a setup with two workstations in a 100MBit cable local area network was chosen (see Figure 5.2). One Caledyo application was started at workstation A. This Caleydo application was configured as a server. Two Caleydo applications have been started on workstation B, both configured as a client application connected to the server on workstation A.

The first test focused on the user and on feedback concerning the responsiveness of the different applications in the setup. Against our expectations, there was a slight but noticeable amount of latency in updates of the view resulting from user interaction on remote applications.

The second test focused on measurements of the network latency. 20 selection update events triggered from the parallel coordinate views were sent from one of the applications on workstation B, and the time it took until the particular events were retrieved at the second application on that workstation was measured. As both applications were configured as client, the events were sent to the server application on workstation A and then sent back again to the second application on workstation B. As the measured timestamps of sending and receiving the events were taken on the same workstation, no time synchronization was necessary. The average measured event transmission time was 362ms. Considering that two hops were needed to transport the event from one client application to the other, a single transmission from one application to another takes 181ms in average.

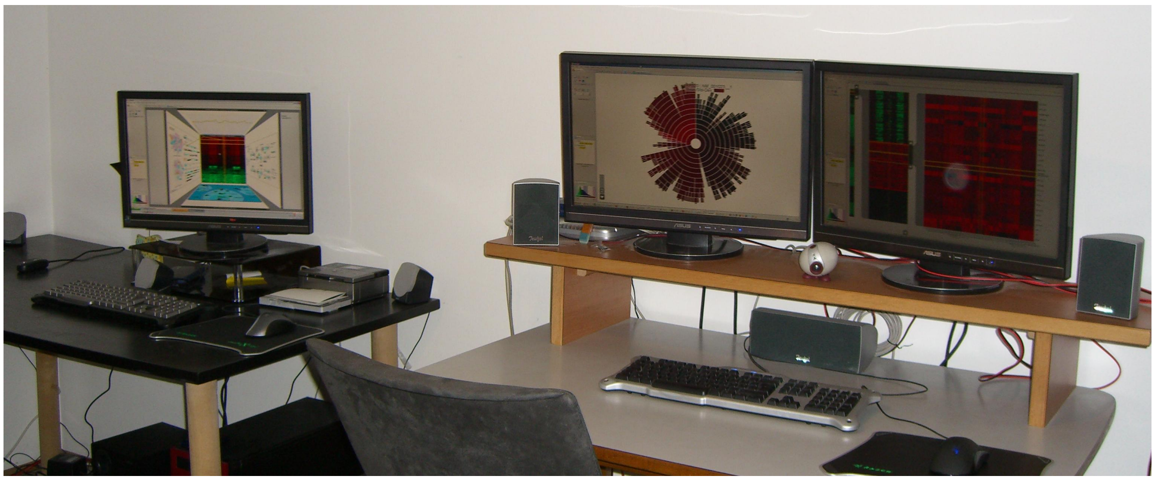


Figure 5.2: *Test setup with two workstations. The left workstation (workstation A) runs the Caleydo server application. The right workstation (workstation B) runs two clients connected to the server. Each application runs in full screen mode on an individual display.*

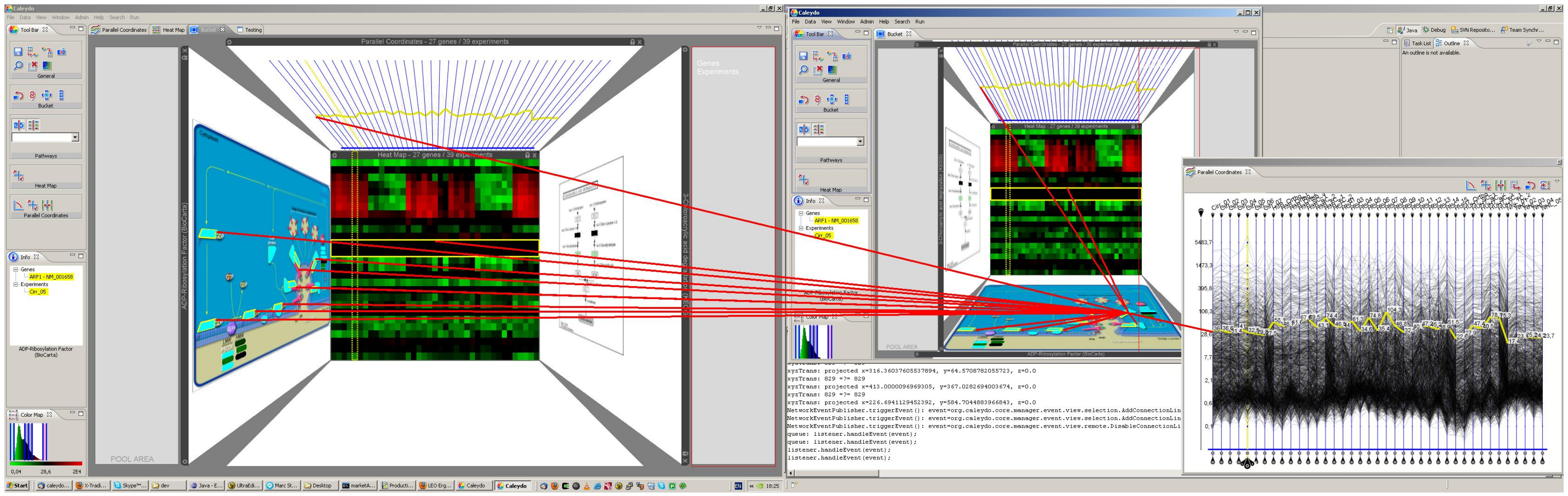
In our opinion, this is too long within a local area network, so we decided to investigate further.

We repeated the test with a slightly different setup. The heatmap view was used to select elements. Each selection within a heatmap view results in the subsequent propagation of four events. Instead of again taking a number of measurements and calculating the average we decided to inspect the events related to one selection operation in detail. The result of this experiment is presented in Table 5.1. The times measured for the first three events are within our expectations and are far from being noticed by users. But the time until the fourth event was received took considerably longer.

#	**Event type**	**time**
1	Clear gene selection	16ms
2	Set Gene Selection	31ms
3	Clear Experiment Selection	31ms
4	Set Experiment Selection	187ms

Table 5.1: *Time needed between sending and receiving the four events involved in the selection of one element in a heatmap view.*

After a further investigation of the source code and some additional time recordings the problem could be tracked down to the code section within the `NetworkEventReceiver` class, which is responsible for reading from the network. Whenever a complete XML document of a serialized event cannot be received within the return of the first blocking read on the socket, another blocking read is initiated. The minimum time for another call of the blocking read was measured to be at least 150ms. We assume that an optimization of the network related part would solve this problem.


Caleydo
File Data View Window Admin Help Search Run
Tool Bar
General
Bucket
Pathways
Heat Map
Parallel Coordinates
Testing
Parallel Coordinates - 27 genes / 39 experiments
Heat Map - 27 genes / 39 experiments
ADP-Ribosylation Factor (BioCarta)
3-Chloroacrylic acid degradation (KEGG)
Genes
Experiments
Info
ARF1 - NM_001658
Cirr_05
Color Map
POOL AREA
Outline
An outline is not available.
18:25

```
xyzTrans: projected x=316.36037605537894, y=64.5708782055723, z=0.0
xyzTrans: 829 =?= 829
xyzTrans: projected x=413.0000096969305, y=367.0282694003674, z=0.0
xyzTrans: 829 =?= 829
xyzTrans: projected x=226.6941129452392, y=584.7044883966843, z=0.0
NetworkEventPublisher.triggerEvent(): event=org.caleydo.core.manager.event.view.selection.AddConnectionLin
NetworkEventPublisher.triggerEvent(): event=org.caleydo.core.manager.event.view.selection.AddConnectionLin
NetworkEventPublisher.triggerEvent(): event=org.caleydo.core.manager.event.view.remote.DisableConnectionLi
queue: listener.handleEvent(event);
queue: listener.handleEvent(event);
listener.handleEvent(event);
listener.handleEvent(event);
```

Figure 5.3: *Visual Links in a dual monitor setup rendered by the Deskotheque mock-up. One Caleydo application is placed on each of the monitors. The application on the left runs in full screen mode. The application on the right has a detached parallel coordinate view. Visual Links connecting elements from all visible views are rendered by the Deskotheque mock-up application.*

Chapter 6

Conclusion and Future Work

Distributed systems and collaboration are topics that today's workers and researchers face nearly every day. Workplaces with more than one display device are getting common. This thesis shows how the InfoVis system Caleydo can make beneficial use of MDE and CSCW concepts. The implemented extensions to the framework are the basis for a distributed and collaborative InfoVis environment. The ability to serialize views and process related data structures makes it possible to run parts of the software synchronized on multiple workstations. The implementation of a groupware interface to Deskotheque enables the support for co-located collaboration.

Synchronization features that were absolutely needed for the distributed use case also brought more stability when running it as single workstation application. In our opinion a consistent use of the MVC design pattern, especially in view related parts of the application, would not only provide a better structuring of the source code. It would also make parts of the software less coupled and easier to distribute on multiple workstations.

The current Visual Link implementation of the mock-up runs into undesired behavior, when view windows overlap or the user switches to other applications. The reason for this is that neither the Caleydo application nor the mock-up is currently aware of overlapping windows. An integration of Visual Links to the window manager or closer interaction of the Visual Link generating software with the window manager could solve this problem. The introduction of Visual Links counters awareness problems that come along with larger display areas. Visual Links across application and window borders open up new vistas for any kind of visualization software.

There is no feedback yet for the new features from users from the biological and medical domain. The usability of Caleydo, as a software which uses multiple views for data visualization, is limited by the display area. Especially during the use of single monitor setups, it is often necessary to switch views and thereby losing the direct visual relationship to entities in hidden views. Although dual monitor setups are an advantage, they are not always sufficient for the desired look and feel. The compound views, Bucket and Jukebox try to counter the problem of limited display space with the price of lower detail level in each particular view. In contrast, a Multi-Display Environment would make it possible to build up a "real" Bucket with multiple display devices.

First steps in using Caleydo as collaboration software showed that Caleydo is not capable yet to handle multiple user interactions. Especially the behavior when hovering the mouse over views and selecting entities always leads to conflicts. The support of multiple selections, one for each user would be beneficial and brings a higher awareness of user actions into the application.

The tests performed on distributed setups did not lead to conflicting and unrecoverable application states among the particular applications. However, this might happen during a more intensive collaborative usage. Especially for co-located collaboration we propose to adopt the local-lag mechanism. Response times in local area networks are usually very small, so the local-lag delay can be set to a very small value. That is why we do not expect that the user will experience any penalties because of the of the application's responsiveness.

For spatially distributed collaboration setups, especially when using internet connections, local-lag will not be adequate. The integration of an event ordering algorithm could be interesting for this case. The necessity to keep a history of events could also be used to introduce a multi-level undo, an often desired feature request in recent user feedbacks.

The current implementation does not provide any communication aids. In spatially distributed setups the collaborators have to use conferencing tools and messengers. If a user does not announce the sending of a view to his or her co-worker, the view suddenly pops up at the receiver. This behavior of the application might confuse the user as the source triggered actions is not obvious for him or her. For this issue, we propose the integration of a chat tool. Similar to a feature messengers use for sending files, the receiver will get a dialog where he or she can decide whether a received view should be displayed.

We are currently looking forward to the integration of the Caleydo-Deskotheque interface into Deskotheque. This will provide an interesting setup for user studies with MDEs and Visual Links. The continuous workspace of Deskotheque could also lead to a drag and drop feature for views. The currently supported view publishing feature distinguishes only between private and public views. We think that it would be beneficial for the user not need to think about the display characteristics and the requirements of a particular view. By delivering requirements parameters to the MDE software, the MDE software would be able to choose an appropriate view. This might be a high resolution view for the finely grained lines of a parallel coordinate view or the table touch top display for a toolbar. More details about this kind of support for InfoVis systems are covered in [Waldner2009] and [Streit2009b].

List of Figures

List of Tables

Bibliography

[Andrews1998] Keith Andrews and Helmut Heidegger. *Information slices: visualising and exploring large hierarchies using cascading, semicircular discs.* In *InfoVis'98: Proc. IEEE Information Visualization Symposium, Carolina, USA*, pp. 9–12. **1998**.

[Aris2007] Aleks Aris and Ben Shneiderman. *Designing semantic substrates for visual network exploration. Information Visualization*, volume 6(4):pp. 281–300, **2007**. ISSN 1473-8716.

[Baldonado2000] Michelle Q. Wang Baldonado, Allison Woodruff, and Allan Kuchinsky. *Guidelines for using multiple views in information visualization.* In *AVI '00: Proceedings on Advanced visual interfaces*, pp. 110–119. ACM Press, New York, NY, USA, **2000**.

[Beca1999] L. Beca, G.C. Fox, and M. Podgorny. *Component architecture for building web-based synchronous collaboration systems.* In *Enabling Technologies: Infrastructure for Collaborative Enterprises, 1999. (WET ICE '99) Proceedings. IEEE 8th International Workshops on*, pp. 108–113. **1999**.

[Biehl2008] Jacob T. Biehl, William T. Baker, Brian P. Bailey, Desney S. Tan, Kori M. Inkpen, and Mary Czerwinski. *Impromptu: a new interaction framework for supporting collaboration in multiple display environments and its field evaluation for co-located software development.* In *CHI '08: Proceeding of the twenty-sixth annual SIGCHI conference on Human factors in computing systems*, pp. 939–948. ACM, New York, NY, USA, **2008**. ISBN 978-1-60558-011-1.

[Boulila2004] N. Boulila, B. Bruegge, and A.H. Dutoit. *Computer supported cooperative software engineering: a framework for supporting distributed concurrent group modeling of software.* In *Proceedings of the International Conference on Applied Computing*, pp. 11–15. Lisbon, Portugal, **2004**.

[Bourqui2006] Romain Bourqui, David Auber, Vincent Lacroix, and Fabien Jourdan. *Metabolic network visualization using constraint planar graph drawing algorithm.* In *IV '06: Proceedings on Information Visualization*, pp. 489–496. IEEE Computer Society, Washington, DC, USA, **2006**.

[Card1999] Stuart K. Card, Jock D. Mackinlay, and Ben Shneiderman, editors. *Readings in information visualization: using vision to think.* Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, **1999**. ISBN 1-55860-533-9.

[Chuah1998] Mei C. Chuah. *Dynamic aggregation with circular visual designs.* In *INFOVIS '98: Proceedings of the 1998 IEEE Symposium on Information Visualization*, pp. 35–43. IEEE Computer Society, Washington, DC, USA, **1998**. ISBN 0-8186-9093-3.

[Churchill1999] Elizabeth F. Churchill and Sara Bly. *It's all in the words: supporting work activites with lightweight tools.* In *GROUP '99: Proceedings of the international ACM SIGGROUP conference on Supporting group work*, pp. 40–49. ACM, New York, NY, USA, **1999**. ISBN 1-58113-065-1.

[Cleveland1993] William S. Cleveland. *Visualizing Data.* Hobart Press, **1993**. ISBN 0963488406.

[Collins2007] Christopher Collins and Sheelagh Carpendale. *Vislink: Revealing relationships amongst visualizations. IEEE Transactions on Visualization and Computer Graphics*, volume 13(6):pp. 1192–1199, **2007**. ISSN 1077-2626.

[Convertino2005] G. Convertino, C.H. Ganoe, W.A. Schafer, B. Yost, and J.M. Carroll. *A multiple view approach to support common ground in distributed and synchronous geo-collaboration.* In *Coordinated and Multiple Views in Exploratory Visualization, 2005. (CMV 2005). Proceedings. Third International Conference on*, pp. 121–132. **July 2005**.

[Cook2005] C. Cook, W. Irwin, and N. Churcher. *A user evaluation of synchronous collaborative software engineering tools.* In *Software Engineering Conference, 2005. APSEC '05. 12th Asia-Pacific*, pp. 6 pp.–. **Dec. 2005**. ISSN 1530-1362.

[Craft2005] Brock Craft and Paul Cairns. *Beyond guidelines: What can we learn from the visual information seeking mantra?* In *IV '05: Proceedings on Information Visualisation*, pp. 110–118. IEEE Computer Society, Washington, DC, USA, **2005**.

[Cruz-Neira1993] Carolina Cruz-Neira, Daniel J. Sandin, and Thomas A. DeFanti. *Surround-screen projection-based virtual reality: the design and implementation of the cave.* In *SIGGRAPH '93: Proceedings on Computer graphics and interactive techniques*, pp. 135–142. ACM Press, New York, NY, USA, **1993**.

[Cruz-Neira1992] Carolina Cruz-Neira, Daniel J. Sandin, Thomas A. DeFanti, Robert V. Kenyon, and John C. Hart. *The cave: audio visual experience automatic virtual environment. Commun. ACM*, volume 35(6):pp. 64–72, **1992**. ISSN 0001-0782.

[Doleisch2007] Helmut Doleisch. *Simvis: interactive visual analysis of large and time-dependent 3d simulation data.* In *WSC '07: Proceedings of the 39th conference on Winter simulation*, pp. 712–720. IEEE Press, Piscataway, NJ, USA, **2007**. ISBN 1-4244-1306-0.

[Doleisch2002] Helmut Doleisch and Helwig Hauser. *Smooth brushing for focus+context visualization of simulation data in 3d.* In *WSCG*, pp. 147–154. **2002**.

[Eisen1998] Michael B. Eisen, Paul T. Spellman, Patrick O. Brown, and David Botstein. *Cluster analysis and display of genome-wide expression patterns. Proc. Natl. Academy of Science USA*, volume 95(25):pp. 14863–14868, **December 1998**. ISSN 0027-8424.

[Ellis1989] C. A. Ellis and S. J. Gibbs. *Concurrency control in groupware systems.* In *SIGMOD '89: Proceedings of the 1989 ACM SIGMOD international conference on Management of data*, pp. 399–407. ACM, New York, NY, USA, **1989**. ISBN 0-89791-317-5.

[Emmerich2008] Wolfgang Emmerich, Mikio Aoyama, and Joe Sventek. *The impact of research on the development of middleware technology. ACM Trans. Softw. Eng. Methodol.*, volume 17(4):pp. 1–48, **2008**. ISSN 1049-331X.

[Fekete2003] J.D Fekete, D. Wang, N. Dang, A. Aris, and C. Plaisant. *Overlaying graph links on treemaps.* In *Proc. of IEEE Symp. on Information Visualization, Poster Session.* **2003**.

[Gamma1995] Erich Gamma, Richard Helm, Ralph Johnson, and John Vlissides. *Design patterns: elements of reusable object-oriented software.* Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, **1995**. ISBN 0-201-63361-2.

[Greenberg2001] Saul Greenberg and Michael Rounding. *The notification collage: posting information to public and personal displays.* In *CHI '01: Proceedings of the SIGCHI conference on Human factors in computing systems*, pp. 514–521. ACM, New York, NY, USA, **2001**. ISBN 1-58113-327-8.

[Greif1986] Irene Greif, Robert Seliger, and William E. Weihl. *Atomic data abstractions in a distributed collaborative editing system.* In *POPL '86: Proceedings of the 13th ACM SIGACT-SIGPLAN symposium on Principles of programming languages*, pp. 160–172. ACM, New York, NY, USA, **1986**.

[Hauser2002] Helwig Hauser, Florian Ledermann, and Helmut Doleisch. *Angular brushing of extended parallel coordinates.* In *INFOVIS '02: Proceedings on Information Visualization*, pp. 127–130. IEEE Computer Society, Washington, DC, USA, **2002**.

[Heer2006] Jeffrey Heer and Maneesh Agrawala. *Software design patterns for information visualization. IEEE Transactions on Visualization and Computer Graphics*, volume 12(5):pp. 853–860, **2006**. Student Member-Jeffrey Heer.

[Heer2005] Jeffrey Heer, Stuart K. Card, and James A. Landay. *prefuse: a toolkit for interactive information visualization.* In *CHI '05: Proceedings of* *the SIGCHI conference on Human factors in computing systems*, pp. 421–430. ACM, New York, NY, USA, **2005**. ISBN 1-58113-998-5.

[Heer2007] Jeffrey Heer, Fernanda B. Viégas, and Martin Wattenberg. *Voyagers and voyeurs: supporting asynchronous collaborative information visualization.* In *CHI '07: Proceedings of the SIGCHI conference on Human factors in computing systems*, pp. 1029–1038. ACM, New York, NY, USA, **2007**. ISBN 978-1-59593-593-9.

[Sharp2002] Jenny Preece Helen Sharp, Yvonne Rogers. *Interaction Design: Beyond Human-Computer Interaction.* John Wiley & Sons, **2002**. ISBN 0470018666.

[Henry2007] N. Henry, J.-D. Fekete, and M.J. McGuffin. *Nodetrix: a hybrid visualization of social networks. Visualization and Computer Graphics, IEEE Transactions on*, volume 13(6):pp. 1302–1309, **Nov.-Dec. 2007**. ISSN 1077-2626.

[Hericko2003] Marjan Hericko, Matjaz B. Juric, Ivan Rozman, Simon Beloglavec, and Ales Zivkovic. *Object serialization analysis and comparison in java and .net. SIGPLAN Not.*, volume 38(8):pp. 44–54, **2003**. ISSN 0362-1340.

[Hornecker2008] Eva Hornecker, Paul Marshall, Nick Sheep Dalton, and Yvonne Rogers. *Collaboration and interference: awareness with mice or touch input.* In *CSCW '08: Proceedings of the ACM 2008 conference on Computer supported cooperative work*, pp. 167–176. ACM, New York, NY, USA, **2008**. ISBN 978-1-60558-007-4.

[Inselberg1985] Alfred Inselberg. *The plane with parallel coordinates. The Visual Computer*, volume 1(4):pp. 69–91, **1985**.

[Isaacs2002] Ellen Isaacs, Alan Walendowski, Steve Whittaker, Diane J. Schiano, and Candace Kamm. *The character, functions, and styles of instant messaging in the workplace.* In *CSCW '02: Proceedings of the 2002 ACM conference on Computer supported cooperative work*, pp. 11–20. ACM, New York, NY, USA, **2002**. ISBN 1-58113-560-2.

[Isenberg2007] P. Isenberg and S. Carpendale. *Interactive tree comparison for co-located collaborative information visualization. Visualization and Computer Graphics, IEEE Transactions on*, volume 13(6):pp. 1232–1239, **Nov.-Dec. 2007**. ISSN 1077-2626.

[Isenberg2008] Petra Isenberg, Anthony Tang, and Sheelagh Carpendale. *An exploratory study of visual information analysis.* In *CHI '08: Proceeding of the twenty-sixth annual SIGCHI conference on Human factors in computing systems*, pp. 1217–1226. ACM, New York, NY, USA, **2008**. ISBN 978-1-60558-011-1.

[Johansen1988] Robert Johansen. *GroupWare: Computer Support for Business Teams.* The Free Press, New York, NY, USA, **1988**. ISBN 0029164915.

[Johanson2002] B. Johanson, A. Fox, and T. Winograd. *The interactive workspaces project: experiences with ubiquitous computing rooms. Pervasive Computing, IEEE*, volume 1(2):pp. 67–74, **Apr-Jun 2002**. ISSN 1536-1268.

[Kaplan1992] S.M. Kaplan and K.D. Swenson. *Operating systems support for collaborative work.* In *Object Orientation in Operating Systems, 1992., Proceedings of the Second International Workshop on*, pp. 360–363. **Sep 1992**.

[Krasner1988] Glenn E. Krasner and Stephen T. Pope. *A description of the model-view-controller user interface paradigm in the smalltalk-80 system. Journal of Object Oriented Programming*, volume 1(3):pp. 26–49, **1988**.

[Lamport1978] Leslie Lamport. *Time, clocks, and the ordering of events in a distributed system. Commun. ACM*, volume 21(7):pp. 558–565, **1978**. ISSN 0001-0782.

[Lex2008] Alexander Lex. *Exploration of Gene Expression Data in a Visually Linked Environment.* Master's thesis, Graz University of Technology, **2008**.

[Li2000] D. Li, L. Zhou, and R.R. Muntz. *A new paradigm of user intention preservation in realtime collaborative editing systems.* In *Parallel and Distributed Systems, 2000. Proceedings. Seventh International Conference on*, pp. 401–408. **2000**.

[Linebarger2005] John M. Linebarger, Andrew J. Scholand, Mark A. Ehlen, and Michael J. Procopio. *Benefits of synchronous collaboration support for an application-centered analysis team working on complex problems: a case study.* In *GROUP '05: Proceedings of the 2005 international ACM SIGGROUP conference on Supporting group work*, pp. 51–60. ACM, New York, NY, USA, **2005**. ISBN 1-59593-223-2.

[Mark2002] G. Mark, A. Kobsa, and V. Gonzalez. *Do four eyes see better than two? collaborative versus individual discovery in data visualization systems.* In *Information Visualisation, 2002. Proceedings. Sixth International Conference on*, pp. 249–255. **2002**. ISSN 1093-9547.

[Mathieu2000] John E. Mathieu, Gerald F. Goodwin, Tonia S. Heffner, Eduardo Salas, and Janis A. Cannon-Bowers. *The influence of shared mental models on team process and performance. Journal of Applied Psychology 85(2)*, pp. 273–283, **2000**.

[Mauve2004] M. Mauve, J. Vogel, V. Hilt, and W. Effelsberg. *Local-lag and time-warp: providing consistency for replicated continuous applications. Multimedia, IEEE Transactions on*, volume 6(1):pp. 47–57, **Feb. 2004**. ISSN 1520-9210.

[Nardi2000] Bonnie A. Nardi, Steve Whittaker, and Erin Bradner. *Interaction and outeraction: instant messaging in action.* In *CSCW '00: Proceedings* *of the 2000 ACM conference on Computer supported cooperative work*, pp. 79–88. ACM, New York, NY, USA, **2000**. ISBN 1-58113-222-0.

[Neuman2005] Petra Neumann, Stefan Schlechtweg, and Sheelagh Carpendale. *Arc-Trees: Visualizing Relations in Hierarchical Data.* In Ken W. Brodlie, David J. Duke, and Ken I. Joy, editors, *Data Visualization 2005, Eurographics/IEEE VGTC Symposium on Visualization Symposium Proceedings*, pp. 53–60. The Eurographics Association, Aire-la-Ville, Switzerland, **2005**.

[Ocker2009] Rosalie Ocker, Mary Beth Rosson, Dana Kracaw, and S. Roxanne Hiltz. *Training students to work effectively in partially distributed teams. Trans. Comput. Educ.*, volume 9(1):pp. 1–24, **2009**.

[Osais2006] Yahya Osais, Souhail Abdala, and Ashraf Matrawy. *A multilayer peer-to-peer framework for distributed synchronous collaboration. IEEE Internet Computing*, volume 10(6):pp. 33–41, **2006**. ISSN 1089-7801.

[Park2000] Kyoung S. Park, Abhinav Kapoor, and Jason Leigh. *Lessons learned from employing multiple perspectives in a collaborative virtual environment for visualizing scientific data.* In *CVE '00: Proceedings of the third international conference on Collaborative virtual environments*, pp. 73–82. ACM, New York, NY, USA, **2000**. ISBN 1-58113-303-0.

[Partl2009] Christian Partl. *Visualizing a Cluster Hierarchy using a Radial Layout.* Master's thesis, Graz University of Technology, **2009**.

[Pirchheim2009] Christian Pirchheim, Manuela Waldner, and Dieter Schmalstieg. *Deskotheque: Improved spatial awareness in multi-display environments. Proceedings of IEEE Virtual Reality Conference, 2009*, **2009**.

[Plumlee2006] Matthew D. Plumlee and Colin Ware. *Zooming versus multiple window interfaces: Cognitive costs of visual comparisons. ACM Trans. Comput.-Hum. Interact.*, volume 13(2):pp. 179–209, **2006**. ISSN 1073-0516.

[JAXB2.0] Java Community Process. *Jsr-000222 java architecture for xml binding (jaxb) 2.0*, **2006**.

[Rekimoto1999] Jun Rekimoto and Masanori Saitoh. *Augmented surfaces: a spatially continuous work space for hybrid computing environments.* In *CHI '99: Proceedings of the SIGCHI conference on Human factors in computing systems*, pp. 378–385. ACM, New York, NY, USA, **1999**. ISBN 0-201-48559-1.

[Robertson2000] George Robertson, Maarten van Dantzich, Daniel Robbins, Mary Czerwinski, Ken Hinckley, Kirsten Risden, David Thiel, and Vadim Gorokhovsky. *The task gallery: a 3d window manager.* In *CHI '00: Proceedings of the SIGCHI conference on Human factors in computing systems*, pp. 494–501. ACM, New York, NY, USA, **2000**. ISBN 1-58113-216-6.

[Schlegl2009] Bernhard Schlegl. *Visual Analytics for Gene Expression Data.* Master's thesis, Graz University of Technology, **2009**.

[Scott2003] Stacey D. Scott, Karen D. Grant, and Regan L. Mandryk. *System guidelines for co-located, collaborative work on a tabletop display.* In *ECSCW'03: Proceedings of the eighth conference on European Conference on Computer Supported Cooperative Work*, pp. 159–178. Kluwer Academic Publishers, Norwell, MA, USA, **2003**.

[Seo2002] Jinwook Seo and Ben Shneiderman. *Interactively exploring hierarchical clustering results. Computer*, volume 35(7):pp. 80–86, **2002**. ISSN 0018-9162.

[Shneiderman1996] Ben Shneiderman. *The eyes have it: A task by data type taxonomy for information visualizations.* In *VL '96: Proceedings on Visual Languages*. IEEE Computer Society, **1996**. ISBN 081867508X.

[Shneiderman2006] Ben Shneiderman and Aleks Aris. *Network visualization by semantic substrates. IEEE Transactions on Visualization and Computer Graphics*, volume 12(5):pp. 733–740, **2006**. ISSN 1077-2626.

[Siirtola2006] Harri Siirtola and Kari-Jouko Räihä. *Discussion: Interacting with parallel coordinates. Interact. Comput.*, volume 18(6):pp. 1278–1309, **2006**. ISSN 0953-5438.

[Singhal1997] Vigyan Singhal and Alan Jay Smith. *Analysis of locking behavior in three real database systems. The VLDB Journal*, volume 6(1):pp. 40–52, **1997**. ISSN 1066-8888.

[Solomita1994] Ethan Solomita, James Kempf, and Dan Duchamp. *Xmove: a pseudoserver for x window movement. X Resour.*, (11):pp. 143–170, **1994**. ISSN 1058-5591.

[Stefik1987] Mark Stefik, Gregg Foster, Daniel G. Bobrow, Kenneth Kahn, Stan Lanning, and Lucy Suchman. *Beyond the chalkboard: computer support for collaboration and problem solving in meetings. Commun. ACM*, volume 30(1):pp. 32–47, **1987**. ISSN 0001-0782.

[Streit2007] Marc Streit. *Metabolic Pathway Visualization Using Gene-Expression Data.* Master's thesis, Graz University of Technology, **2007**.

[Streit2008] Marc Streit, Michael Kalkusch, Karl Kashofer, and Dieter Schmalstieg. *Navigation and exploration of interconnected pathways. Computer Graphics Forum (EuroVis 2008)*, volume 27(3):pp. 951–958(8), **May 2008**.

[Streit2009a] Marc Streit, Alexander Lex, Michael Kalkusch, Kurt Zatloukal, and Dieter Schmalstieg. *Caleydo: Connecting pathways with gene expression. Bioinformatics*, **2009**.

[Streit2009] Marc Streit, Alexander Lex, Heimo Müller, and Dieter Schmalstieg. *Gaze-based interaction for information visualization.* In *Proceedings of web3DW 2009 Conference, Algarve, Portugal.* **2009**.

[Streit2009b] Marc Streit, Hans-Jörg Schulz, Dieter Schmalstieg, and Heidrun Schumann. *Towards multi-user multi-level interaction.* In *Workshop on Collaborative Visualization on Interactive Surfaces (part of VisWeek'09)*. **2009**.

[JavaRMI] Inc. Sun Microsystems. *Java remote method invocation (rmi) specification*, **2006**.

[Fernanda2008] Fernanda Viégas and Martin Wattenberg. *Shakespeare, god, and lonely hearts: transforming data access with many eyes.* In *JCDL '08: Proceedings of the 8th ACM/IEEE-CS joint conference on Digital libraries*, pp. 145–146. ACM, New York, NY, USA, **2008**. ISBN 978-1-59593-998-2.

[Waldner2009] Manuela Waldner, Alexander Lex, Marc Streit, and Dieter Schmalstieg. *Design considerations for collaborative information workspaces in multi-display environments.* In *Workshop on Collaborative Visualization on Interactive Surfaces.* **2009**.