FAKULTÄT
FÜR !NFORMATIK
Faculty of Informatics

A State-of-the-Art Report on Docking Frameworks

BACHELORARBEIT

zur Erlangung des akademischen Grades

Bachelor of Science

im Rahmen des Studiums

Medieninformatik und Visual Computing

eingereicht von

Benjamin Kowatsch
Matrikelnummer 0828124

an der
Fakultät für Informatik der Technischen Universität Wien

Betreuung: Dipl.Ing. Bilal Alsallakh, Dr.techn.

Wien, 16.02.2015

(Unterschrift Benjamin Kowatsch)

(Unterschrift Betreuung)

Abstract

Modern user interfaces make extensive use of multiple windows and views that enable users to explore different pieces of information and solve different tasks. Efficient usage of coordinated and multiple views requires a window management framework that allows users to interactively arrange their views within the available screen-space. A variety of software libraries have been developed to support window management in various software environments, both for desktop and web-based interfaces. These libraries often feature docking frameworks that support window management using drag and drop. Examining all of these libraries is a tedious task for software developers, who need to select an appropriate library for their purpose without the need for extensive tests and experimental implementations. This thesis helps developers in exploring, comparing and selecting docking frameworks by functionalities. After introducing the reader to theoretical foundations, an overview of state-of-the-art frameworks is given, along with a discussion of their features and technical details of their implementations. Additionally, I provide reference implementations of a multiple view user interface using selected frameworks, to test and demonstrate their features and allow developers to extend these implementations. Finally, I conclude this work with a comparison of the presented docking frameworks regarding their features and capabilities.

Contents

CHAPTER 1

Introduction

User interfaces in modern operating systems and software programs make extensive use of multiple windows and views that allow users to explore different pieces of information and solve different tasks. Such multiple views are particularly important for information visualization systems to support analyzing complex and multi-faceted data sets. The views can be tiled, resized, repositioned, focused, opened, closed and docked to another view. Docking means that views can be snapped to each other, either via drag'n'drop or using another interaction method. This is especially important in applications that use multiple windows to allow the user to customize the interface. Figure 1.1 gives an example for docking functionality illustrating only one way how docking can be performed. In this example the user drags a view, for example "Files", and drops it to the up arrow key. After that, the "Files" view will be snapped above the "Properties" and "Code" view. Of course there are other ways of achieving such functionality. For example, a view could be directly dropped on the border line of another view, subsequently snapping both views to each other.

Currently there are many docking frameworks available, either commercial or open- source. At the moment, no comprehensive survey has been conducted on the features and functionality of these frameworks. Therefore it is hard to choose a suited framework for certain requirements - and to compare the strengths and weaknesses of these frameworks for a specific task. This thesis gives a state-of-the-art report on docking frameworks. It should serve as a solid orientation for readers who are searching for a scientific comparison and overview of docking with Java- and web-technologies.

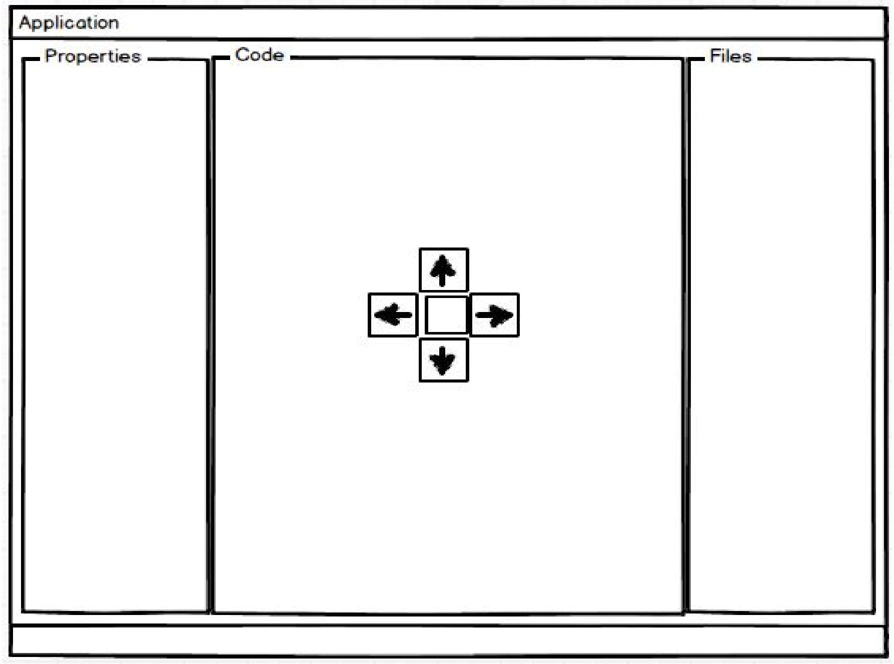


Figure 1.1: Docking functionality with arrows indicating to which view a dragged window will be snapped.

The following list gives an overview of the contributions of this thesis:

- theoretical foundations as well as an overview of researched frameworks, see chapter 2
- prototype implementation of selected frameworks, see chapters 4 and 5
- comparison of the implemented frameworks, see chapters 6 and 7

Based on this thesis, it should be possible to determine whether the use of a specific framework fits the specified requirements of a software project. Additionally, the thesis provides theory to understand how window management works and which techniques exist to properly handle multiple views in order to assist the user with his/her task(s). Furthermore, the thesis moves the emphasis to Information Visualization with both the practical and the theoretical part.Therefore, theory will be concluded in a manner to be relevant to Information Visualization and the prototype will use Information Visualization for demonstration purposes.

CHAPTER 2

Related work and theoretical foundations

Before getting into practical details of docking frameworks, it is important to understand which principles back up the design decisions of user interfaces that actually use docking. In the following sections window and display space management, interaction with multiple views, multitasking with multiple views, and visualization with multiple views will be covered.

2.1 Window and Display Space Management

Today's most common operating systems make use of window managers. As [Myers, 1988] points out, window managers are software packages that help the user to separate contexts on one or more computer screens. The main goal is to help users to keep an overview while handling multiple activities. This is mainly achieved through the higher level interface and the basic features of window managers, the desktop. The desktop acts as a metaphor, using a computer is like doing operations on a physical desk. Additionally, window managers provide a variety of basic features for interactions that allow rearranging the user interface. These features include windows in general, popup menus, icons, tiled or overlapping windows and setting the focus on a certain window, to mention just a few. [Myers, 1988] discusses and defines the most important features. Overlapping allows windows to be placed on top of each other, either completely or only partially. In opposite to overlapping, tiling places windows next to each other, therefore not allowing windows to overlap. Additionally, window managers often make use of icons. It allows replacing an application with a small icon when the application is not running. Another important feature is the management of the window focus, referring to the window that is attached to the keyboard and receives input. Lastly, most window managers make use of some kind of pointer, the most common being a pointer controlled by an input device, for example a computer mouse. This pointer specifies locations on the screen and can therefore be used to interact with the elements of the specified location.

Window and display space management can be done in several ways from a user point of view as [Hutchings and Stasko, 2004] found out via a user study. They investigated different window managers that let users build up their own window arrangement. For example systems that allow the user to arrange sets of windows, each set belonging to a specific task, and switching between these sets. Another example allows peeling and rotating of windows as in Microsoft Windows when ALT+TAB key combination is pressed. [Hutchings and Stasko, 2004] interviewed different users on their regular workspaces. They found out that users can be put into three categories while handling multiple windows.

The first category is called Maximizers. Users of this category tend to maximize every window (Figure 2.1).Switching windows in this category happens mostly via the keyboard sequence ALT+TAB.

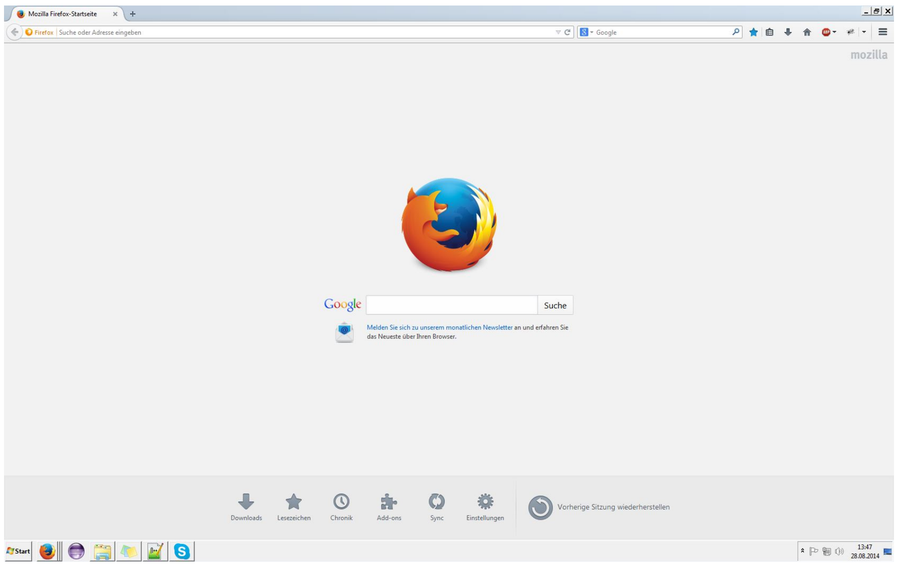

Mozilla Firefox-Startseite
Suche oder Adresse eingeben
Google
mozilla
Suche
Melden Sie sich zu unserem monatlichen Newsletter an und erfahren Sie das Neueste über Ihren Browser.
Downloads
Lesezeichen
Chronik
Add-ons
Sync
Einstellungen
Vorherige Sitzung wiederherstellen
Start
13:47
28.08.2014

Figure 2.1: Desktop screenshot: Maximizer category

Near Maximizers is the second category. This category uses one smaller window which they frequently use besides their maximized windows (Figure 2.2) or they leave a small area of desktop icons uncovered. Near Maximizers switch their windows with mouse clicks.

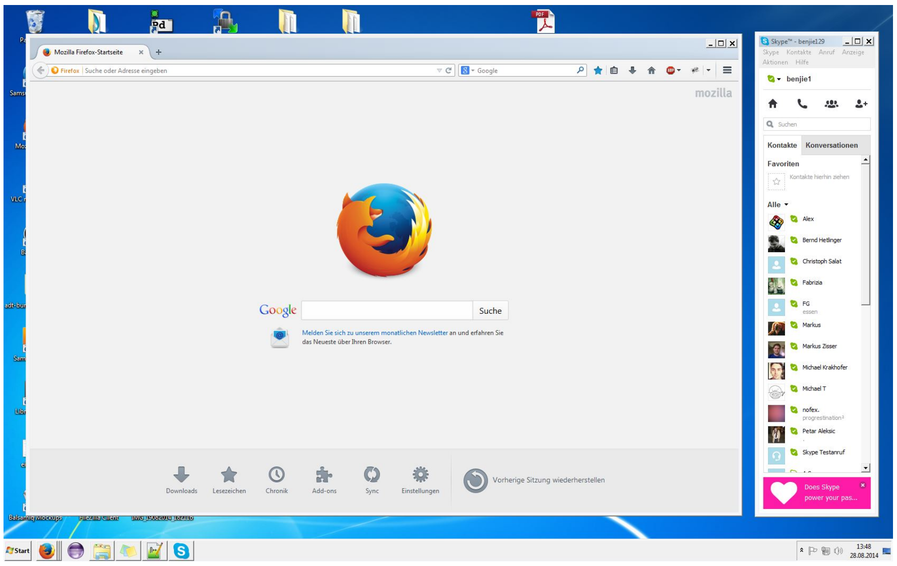


Figure 2.2: Desktop screenshot: Near Maximizer category

The last category is called Careful Coordinators. These users have many windows visible simultaneously and no window is maximized (Figure 2.3). Every window has an important function and users arrange their windows based on their functions.

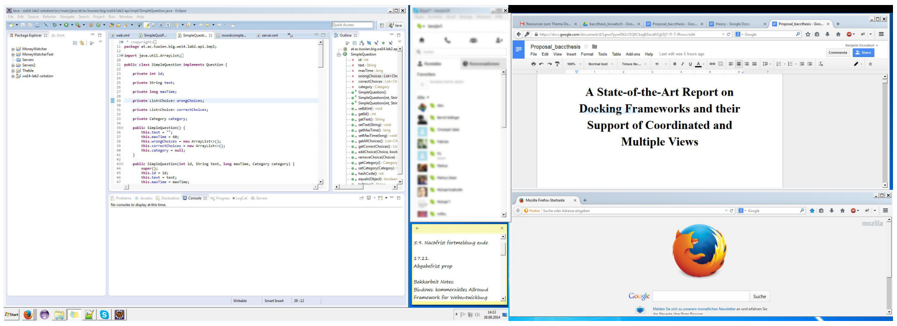


Figure 2.3: Desktop screenshot: Careful Coordinator on two monitors

Additionally, [Hutchings and Stasko, 2004] analyzed in detail common practices that have been established by users regarding window management and how these practices are influenced by the environment. For example, some users were very sensitive about their privacy and did not want E-Mails or Instant-Messages to be shown on the screens while not actively working with them. Therefore they were using several techniques for hiding these windows. Furthermore, users were not strictly tiling their windows, instead they overlayed windows over each other to just see the important information they needed. They did this to make full use of their screen-size. Another interesting practice was the use of empty space that actually was not empty. Some users left empty space on their screens where they placed files or icons for quickly launching applications. Windows were also used as reminders, especially by people that often got interrupted while working and had to abandon their task. Some of the users even used multiple reminder windows. Input devices often define how users interact with windows and screens. For example, one user had two monitors but it was very annoying to traverse from one screen to the other because of the limited space for moving the mouse. This caused the user to nearly never actively use the second monitor.

2.2 Multitasking with multiple views and docking

When working with computers, users do not necessarily rely just on one document or one window for completing a single task, as [Shibata and Omura, 2012] point out. Therefore users in different work areas usually perform multitasking when working with computers. 2.1 demonstrated how window managers support users while multitasking. [Shibata and Omura, 2012] present a framework which allows docking of windows. Its purpose is to manage multiple windows and support multitasking. To achieve the design goal, the framework provides four main features. First, it allows constructing of and switching between workspaces, so that the user does not have to remember which windows belong to which workspace, like in conventional window managers. To keep the user interface for switching tasks between workspaces, [Shibata and Omura, 2012] decided to use docking as the preferred interaction method. Docking eliminates the need to define group or hierarchical relations between the windows, as they can simply be docked to each other in a workspace context. The second main feature is the possibility to save and restore workspaces. Instead of setting up the whole environment with every startup of the computer, users just have to use one action. Furthermore, [Shibata and Omura, 2012] state that it is important to make use of the fact that many people already have larger display space and multiple monitors available. Therefore, they avoid overlapping and use tiling of windows instead. Tiling further supports the design decision to use docking windows as an interaction technique. Lastly, the Docking Window Framework (DWF) they propose supports multiple window operations that affect all docked windows. For example enlarging one window makes the other docked windows smaller. Evaluating their framework with a user study, [Shibata and Omura, 2012] found out that users were able to finish their tasks faster by about 23 %, primarily because of the fact that users were able to set up a window layout suited for their tasks more quickly.

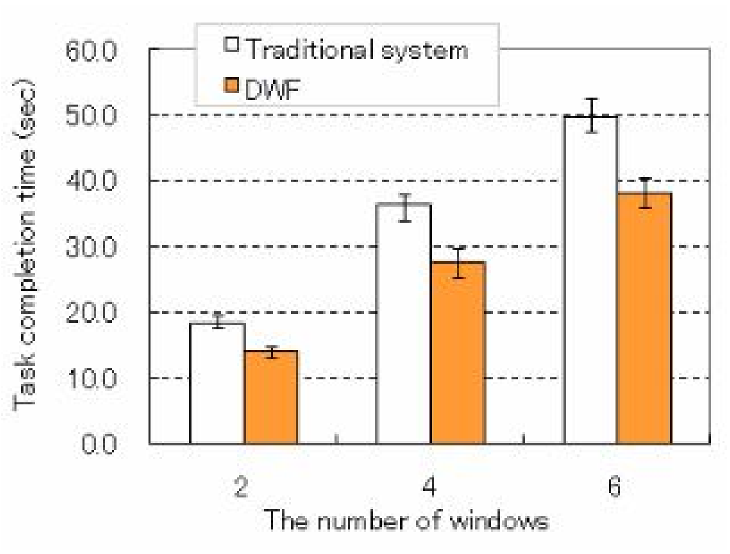


Figure 2.4: Task completion time in window arrangement tasks both in a traditional system and in Docking Window Framework (DWF) [Shibata and Omura, 2012].

This practical example of a window manager system shows how docking can enhance and make the workplace of users more efficient.

2.3 Visualization with multiple views

As pointed out in the previous chapters, multiple views are very useful in different areas and for different tasks. The reference implementations of selected docking frameworks, provided as part of this thesis,l use visualizations for demonstration purposes. Therefore it is important to show which rules apply to the usage of visualizations in this context. [Wang Baldonado et al., 2000] introduced eight guidelines for using multiple views in Information Visualization which are derived by their own experience and by analyzing existing systems in that area. Additional input was given at a workshop [1] on information exploration environments. [Wang Baldonado et al., 2000] split the guidelines in two groups. The first group discusses if the use of multiple views is appropriate while the second group discusses how a system with multiple views should be designed. The remainder of this section provides a summary of these rules.

Rule of Diversity

Use multiple views when there is a diversity of attributes, models, user profiles, levels of abstraction or genres.

[Wang Baldonado et al., 2000] present diversity as one of the most important reasons to use

[1]CHI '98 Workshop on Innovation and Evaluation in Information Exploration Interfaces

multiple view systems. A popular example is if different levels of detail are present and multiple views serve as kind of progression through the dataset.

Rule of Complementarity

Use multiple views when different views bring out correlations and/or disparities.

This rule proposes the use of multiple windows if it helps to understand complex relationships among different data sets or to find hidden relations between views.

Rule of Decomposition

Partition complex data into multiple views to create manageable chunks and to provide insight into the interaction among different dimensions.

Decomposition helps the user to comprehend otherwise cognitively overwhelming content. The rule of decomposition can be compared with the principle of "divide and conquer". [Wang Baldonado et al., 2000] suggest that this rule could simply be implemented by letting the user see one dataset after another. A more sophisticated way combines the rule of complementarity with the rule of decomposition to give insight into multiple dimensions, thus making it easier to compare datasets.

Rule of Parsimony

Use multiple views minimally.

Generally, multiple views demand a steeper learning curve and increased cognitive attention from the user. Additionally, multiple views require more screen space, so they should only be used if there is a strong reason to do so.

Rule of Space/Time Resource Optimization

Balance the spatial and temporal costs of presenting multiple views with the spatial and temporal benefits of using the views.

It is crucial for the designer to know how much space and time is available to the user, and how much time and space is consumed for each of the views to be displayed. Different factors have to be considered, for example download times for images in multiple views or if it is even possible for the user to see all images at once because of limited display space.

Rule of Self-Evidence

Use perceptual cues to make relationships among multiple views more apparent to the user.

Because of the difficulties in recognizing and processing of relationships in multiple views, it is important to give proper cues to shift the recognization of these relationships from cognition to perception. Examples for commonly used cues are highlighting of information, spatial arrangement of views or coupled interaction of views.

Rule of Consistency

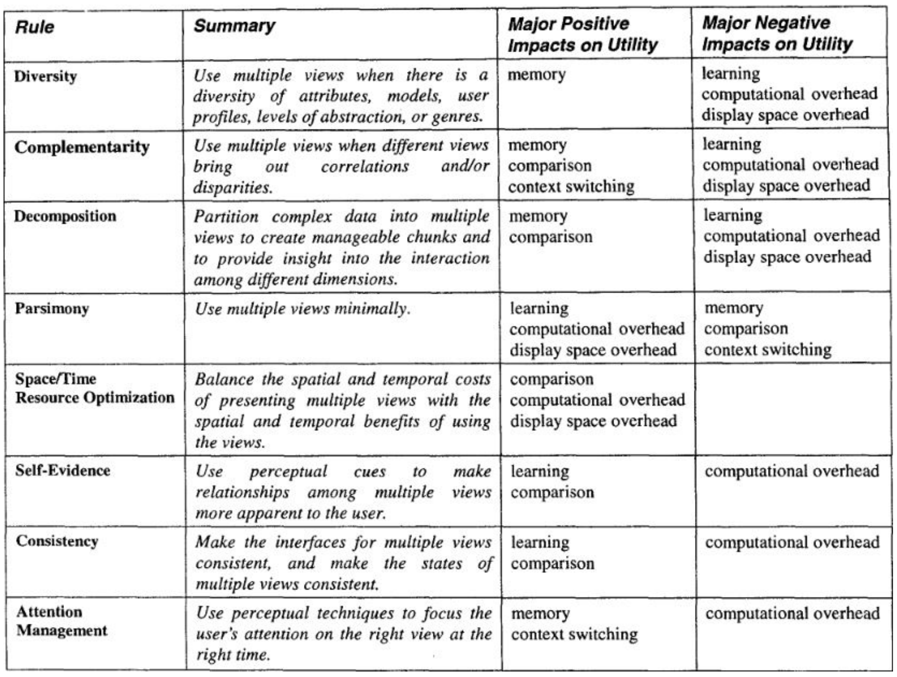
Make the interface for multiple views consistent and make the states of multiple views consistent.

In general, multiple views should provide consistency especially regarding system state and interface affordances. As an example of system consistency, displaying a region in one view should result in displaying the same region in another related view. Making interface affordances consistent makes it easier for the user to learn how to handle a multiple view system.

Rule of Attention Management

Use perceptual techniques to focus the user's attention on the right view at the right time.

Systems that follow this design guideline try to draw the attention of the user to the right views at the right time. Therefore perceptual techniques like animation, sounds, highlighting and movement are used similar to the techniques used in the rule of self-evidence.

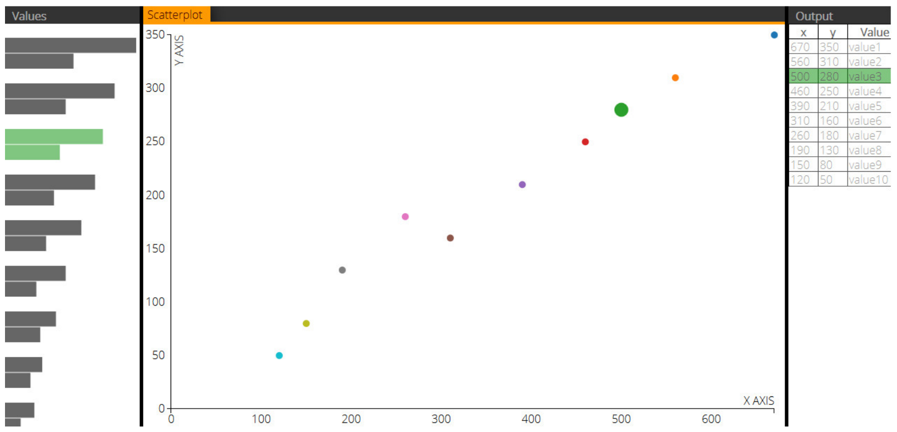


Rule	Summary	Major Positive Impacts on Utility	Major Negative Impacts on Utility
Diversity	*Use multiple views when there is a diversity of attributes, models, user profiles, levels of abstraction, or genres.*	memory	learning computational overhead display space overhead
Complementarity	*Use multiple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

Figure 2.5: Design guidelines for multiple views ([Wang Baldonado et al., 2000])

2.4 Interactive data exploration with multiple views

[Roberts, 2007] surveyed coordinated and multiple views in exploratory visualization and mentions different interaction techniques. In general, interaction includes filtering of data and what is displayed, changing the mapping of information, navigation in the information like zooming, and changing the placement of windows on the screen. [Roberts, 2007] classifies interaction into two categories. The first category is called indirect manipulation and makes use of dynamic queries like sliders, buttons and menus. As an example, sliders are mentioned which make it possible to have an easy way to express what is visualized and how the viewed data is constrained. The second category is direct manipulation. Techniques in this category allow the user to directly filter elements using the visualization itself. The general approach is to use brushing, where for example selecting and highlighting parts of the visualization or data in one view leads to the same highlighting in another view. Additional techniques are manipulators and widgets directly associated to an object to change its properties.



x	y	Value
670	350	value1
560	310	value2
500	280	value3
460	250	value4
390	210	value5
310	160	value6
260	180	value7
190	130	value8
150	80	value9
120	50	value10

Figure 2.6: Example for direct manipulation with brushing.

CHAPTER 3

Method

The following chapters 4, 5, 6 and 7 contain the results of my research regarding the different frameworks. I found most of these frameworks with the help of developer forums such as www.stackoverflow.com, my own Google search and with the support of my advisor Bilal Alsallakh.

The following keywords were used for my search: docking, docking frameworks, window management tools, drag and drop frameworks, Java docking frameworks, web docking frameworks, JavaScript docking frameworks, window management docking frameworks. This search has revealed ten frameworks that are based on web technology and fourteen frameworks based on Java technology.

The frameworks were then filtered by their last update time, so that only frameworks where the last update happened after 2010 were relevant. Furthermore, I excluded frameworks whose download count (if available, for example on Google Code or GitHub) was relatively low. As you will notice, most of the presented frameworks are open-source software. I did not exclude commercial frameworks per se but as testing possibilities with these frameworks were limited, only a couple of them are listed. As one can see, I checked on Java and "native" web-based frameworks (HTML, CSS, JavaScript) only and my thesis does not include analysis of frameworks based on Microsoft Silverlight, Flash or similar technologies.

CHAPTER 4

Overview of web-based frameworks

This chapter presents different web based frameworks that can be used for docking. A comprehensive overview will be given on available features of these frameworks, in addition to a reference implementation of these features. Browser compatibility is tested for each framework, along with other problems encountered in the tests. The browsers used for testing were Internet Explorer 11.0.9600.17280, Firefox 32.0.3, Opera 24.0.1558.64 and Google Chrome 37.0.2062.124.

4.1 Dock Spawn

General

DockSpawn [Code Respawn, 2012] is a framework that was specifically developed for docking and its related features. It provides functionality for docking, floating dialogs and tabbed panels. This framework is open source software released under the MIT License [1]. Included in the software package of the framework is a demo which demonstrates how all provided features can be implemented. Furthermore, DockSpawn has a very extensive documentation on its web page.

Features

Docking:

Docking in DockSpawn works with predefined boxes that are oriented like a compass around a center-box. These predefined boxes are located in the center position of the screen. Additionally, four boxes appear on the border of the screen which are again oriented like a compass. Dropping the dragged panel in a box that is located around the center box results in docking the dragged panel and adjusting either the width or height to fit to the center panel. Dropping the dragged panels in the boxes on the border of the screen results in the dragged panel docked to the center

[1] http://opensource.org/licenses/MIT

panel again but taking up the full screen width or height. If the dragged panel is dropped in the center box, then it becomes a tabbed panel. Figure 4.1 is a screenshot of the docking process in the reference implementation of DockSpawn. Every view can be resized in every direction and the docked views are resized accordingly.

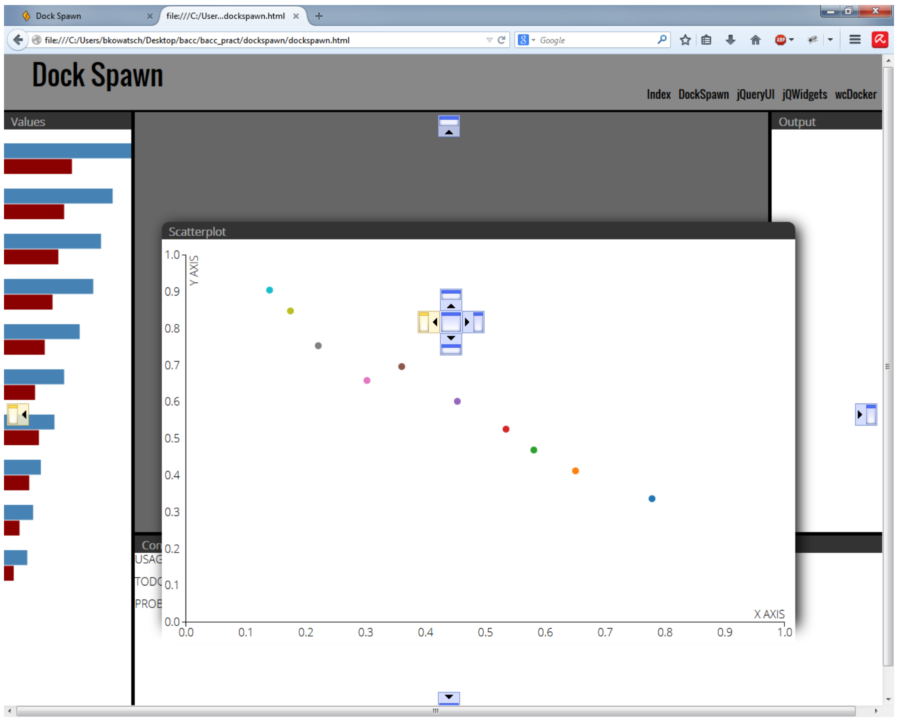

Dock Spawn
file:///C:/Users/bkowatsch/Desktop/bacc/bacc_pract/dockspawn/dockspawn.html
Dock Spawn
Index DockSpawn jQueryUI jQWidgets wcDocker
Values
Output
Scatterplot
Y AXIS
X AXIS

Figure 4.1: Floating view and docking boxes in Dock Spawn.

Floating Dialogs:

DockSpawn supports dialogs that float anywhere on the screen, but not outside of the browser. A panel has to be dragged onto the preferred position on the screen to make it float. Floating dialogs can be docked again using the docking boxes.

Tabbed Panels:

As mentioned, dropping a panel in the center box makes it a tabbed panel. A tabbed panel can be docked to another panel again by clicking the tab and then dragging and dropping it to the desired location.

Style:

DockSpawn ships with a predefined CCS file for styling. Changing this style can be done by examining and changing the corresponding CSS classes. In order to retrieve these classes, developer tools like firebug and its inspect tools can be very helpful.

Save/Load Functionality:

DockSpawn provides functionality for saving and restoring layouts of multiple views. However, my testing as well as Dock Spawn's issues list on GitHub [1] have revealed that this feature does not work at all. All entries in the issues list as well as my own testing have raised the problem of an undefined variable in Dock Spawn's source code for save/load functionality.

Problems

Browser compatibilty:

All features, except save/load functionality that does not work at all, are compatible with all browsers, but there are problems regarding the implementation, as described in the next paragraph.

Documentation:

Dock Spawn offers a detailed API documentation on its webpage [2], as well as a short code snippet to set up some views. Additionally, a demo can be found which demonstrates all features.

Implementation:

The reference implementation of Dock Spawn required 63 lines of JavaScript code and 21 lines of HTML code to achieve the desired functionality, including a workaround for resizing the scatterplot when resizing a panel. This workaround was necessary because content of one panel overlapped into panels that were located in the south of the screen. Content that would overlap from left to right was correctly clipped even when resizing panels. Another bug occured when using the save/load functionality of DockSpawn. This issue is raised by several other users on GitHub because of a seemingly undefined variable in Dock Spawn's source code, as mentioned above in the features overview.

4.2 JQuery UI Layout

General

The reference implementation is based on a plugin called jQuery UI Layout [Balliano, Dalman, 2014]. This plugin has many features and allows to implement docking views but unfortunately without the possibility to drag-and-drop docked views to other regions. Nevertheless, there are possibilites like sliding, hiding and collapsing which can work as a replacement for the drag-and-drop functionality. Additionally, if drag-and-drop functionality is not required for a project, jQuery UI Layout can be the better choice because of its wealth of UI features. The plugin is licensed under the GPL [1] and MIT [2] licenses and is an open-source project. jQuery UI Layout

[1]https://github.com/coderespawn/dock-spawn/issues

[2]http://www.dockspawn.com/

[1]http://www.gnu.org/copyleft/gpl.html

[2]http://opensource.org/licenses/MIT

also provides extensive documenation on its webpage and demos on many different aspects of the plugin. jQueryUI Layout depends on two libraries, jQuery and jQuery UI.

Features

Docking:

jQuery UI Layout provides five regions, north, south, east and west. Every region can be nested to create sophisticated layouts with multiple views. The framework does not support drag and drop of views and subsequently snapping them to each other. However, the layout itself is highly customizable and there are many features like collapsable, hidable or slidable panels which can be more important features for specific projects than other features.

Style:

jQuery UI Layout comes with very little predefined styling. The only elements that are notably styled out-of-the-box are borders, splitters and resizers. As the elements can be easily accessed with auto-generated CSS classes or with custom CSS classes, it is easy to find the corresponding classes to apply CSS styling. Additionally, there is a default layout which can be activated by setting the option applyDefaultStyles to true.

Other features:

As mentioned above, jQuery UI Layout provides functionality for collapsable, hidable and slidable panels. Additionally, panels can be resized and decorated with a collection of options [3] for each panel. It is also possible to define callback functions for events triggered by showing, hiding, opening, closing and resizing panels. Lastly, the framework supports keyboard shortcuts (called “hotkeys “) as well as headers and footers for every region (and every nested region).

Problems

Browser compatibilty:

Testing the above-mentioned features did not lead to any errors in either of the tested browsers.

Documentation:

The most helpful resource for developing with jQuery UI Layout was the official documentation [3]. Aside from that, there are also plenty of demos with different layouts from where the code can be inspected with an inspector tool.

[3] `http://layout.jquery-dev.com/documentation.cfm`

Implementation:

The reference implementation has 42 lines of JavaScript code and 28 lines of html code. Implementing all the features did not lead to any major problems except that jQuery UI Layout has its own (yet simple) syntax which one has to learn in order to make proper use of the framework.

4.3 wcDocker

General

wcDocker [Houde, 2014] is a framework that is specifically designed for docking. Its features are tailored towards multiple views, like scrolling panels, tabbed panels, saving of multiple views layout and usual interaction techniques like drag and drop, resizing and floating. From its webpage on GitHub [1] one can navigate to a demo [2] where all the features are demonstrated. The framework is open source and licensed under the MIT License [3]. The API documentation of wcDocker is also located on the webpage. As the framework was undergoing active development during the time of writing this thesis, it is recommended to examine the features of the demo [2] too.

Features

Docking:

The docking feature of wcDocker,in my opinion, has a more intuitive drag-and-drop approach than for example Dock Spawn when arranging multiple docked views. That means that there are no boxes, like in DockSpawn, that define whether the view is positioned to north, south, east or west. Instead, a view sticks to another, depending on the location where it is dropped in the other view, either in the north, south, east or west.

[1] `https://github.com/WebCabin/wcDocker/wiki/wcDocker`
[2] `http://docker.webcabin.org/`
[3] `http://opensource.org/licenses/MIT`

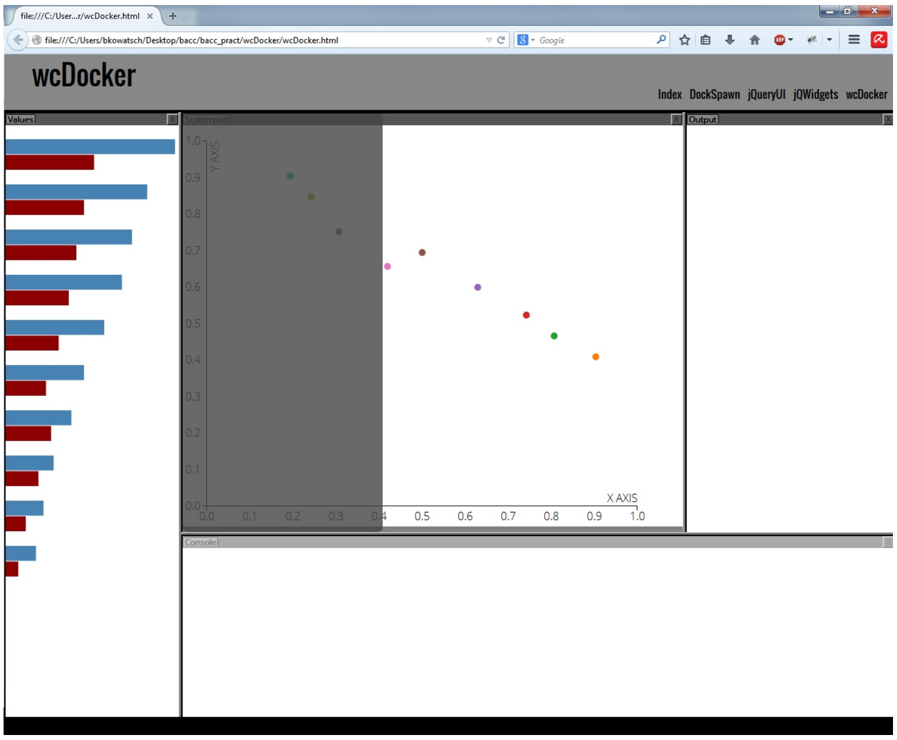


Figure 4.2: Docking the Console view to the left/west of the Scatterplot view.

Figure 4.2 shows an example of the docking process in wcDocker. The dragged view becomes transparent and a grey overlay appears to identify the new location of the view. If the overlay reaches a location where it can be positioned to another view, it becomes darker. Resizing one view also affects all other surrounding views.

Context Menu:

wcDocker provides a context menu via right click with standard options and the possibility to add custom options. Standard options include adding a registered panel, detaching a panel to make it float, closing a panel and flashing it for visual recognition.

Floating Dialogs:

Panels can either be dropped to a location where they do not snap to other views or detached via the context menu to make them float. It is not possible to let the dialog float outside of the browser window.

Tabbed Panels:

Each panel created by the framework has one tab. Other views can be dragged to the title bar to have multiple tabbed panels in one view or they can have multiple tabs initially. These tabs can also be interchanged between multiple views.

Style:

wcDocker provides three themes out-of-the-box that mostly change the coloring of backgrounds. All styles can be changed by modifying the corresponding CSS classes.

Save/Load Functionality:

The Save/Load feature either stores or reloads a previously saved layout. As either case needs just one line of code, this is a quick and easy way to enable the user to save different layouts for different tasks. As the layout is saved in JSON format, handling different layouts can be done by simply saving the JSON data to a variable and enabling access to them via an interface option.

Other features:

wcDocker offers the possibility to add custom buttons to panels that appear in the upper-right area. Additionally panel buttons and tabs can have their own icon image. The framework also has an event system in place to react on panel change events.

Problems

Browser compatibilty:

All features described above were tested in the major browsers mentioned. As this framework is regularly updated, I also tested the already existing and additional features on the demo page regularly during writing this thesis [1]. Compatibility is one of the major features outlined by the developer of this framework.

Documentation:

On the GitHub page [2] you can find basic installation and feature guides. To actually implement all the features I found it most efficient to consult the API documentation (also linked on GitHub) and inspect the demo code [1].

Implementation:

The reference implementation required 119 lines of JavaScript code and 1 line of HTML code. Implementing the features of the framework mainly involves registering a panel to a container DIV element and consequently defining settings and additional features as well as the positioning of the panels.

[1] `http://docker.webcabin.org/`
[2] `https://github.com/WebCabin/wcDocker`

4.4 jQWidgets

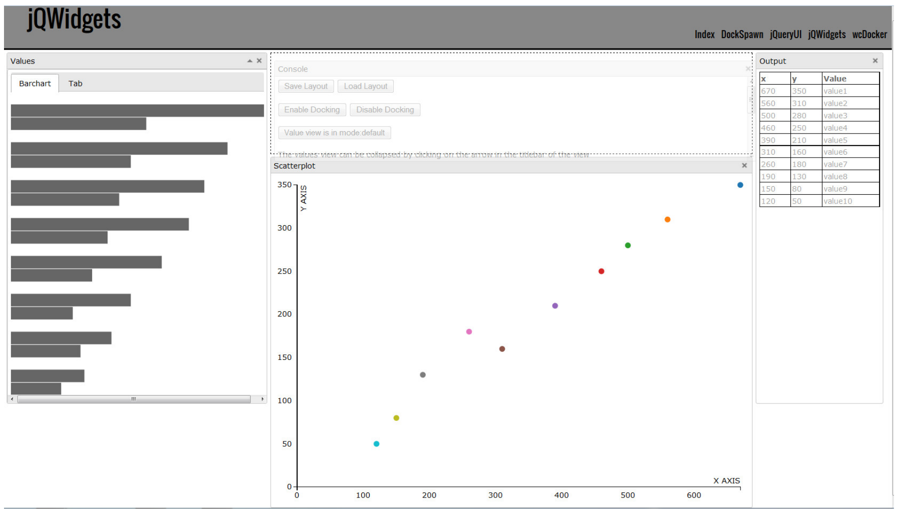
General

jqWidgets [jQWidgets, 2014] is a docking framework that offers a wide range of view-management features besides docking. On its webpage one can view all features and get a first overview of the functionality in form of a demo for every feature as well as the corresponding source code to implement it. Additionally, this first webpage also provides an API documentation for all methods, variables and classes of the features. If jQWidgets is used for non-commercial use, one can use it under the Creative Commons License [1], otherwise a license has to be purchased [2]. jQWidgets also depends on jQuery with the minimum version 1.11.0.

Features

Docking:

The docking feature of jQWidgets does not work with south/east/west/north regions where views can be docked to, like for example Dock Spawn (see section 4.1. Instead, the existing views define regions (respectively anchors) that determine where views can be snapped to. As an example Figure, 4.3 describes the effect of moving the console view above the scatterplot view in the reference implementation.



The values view can be collapsed by clicking on the arrow in the titlebar of the view

x	y	Value
670	350	value1
560	310	value2
500	280	value3
460	250	value4
390	210	value5
310	160	value6
260	180	value7
190	130	value8
150	80	value9
120	50	value10

Figure 4.3: Moving the console view above the scatterplot view with jQWidgets

In Figure 4.3 moving a view above either the values, scatterplot or output view moves them downwards. This is essentially what docking in jQWidgets does with all other views: it moves them vertically, depending on where the other views are placed. It is not possible to dock or tab the views inside of another view, like in Dock Spawn or wcDocker.

[1] `http://creativecommons.org`

[2] currently available for $199-$899, depending on the license type

Floating Dialogs:

Floating Dialogs can be achieved by switching the mode of the view to floating. The *floating mode* lets one place the view independently of the docking regions and other views. There are two other modes: The *docking mode* prohibits placing views anywhere else than in the docking regions. The *default mode* is a mix of floating and docking: If a view is dragged to a position on the page where no docking region is defined, it floats, and otherwise it is docked into the docking region. Floating dialogs cannot be placed outside of the browser window.

Tabbed Panels:

This feature is specifically mentioned, as there is no possibility to dock a view inside of another. Every view can have multiple tabbed panels. As the tab feature is not part of the docking feature, it has to be imported.

Style:

The possibilities for styling the components of this framework are more advanced, compared to all other web-based frameworks. There are several themes provided on the webpage of jQWidgets and aside from that one can either adjust these existing themes or create new ones with the theme builder [1]. As an example, one can import a predefined theme and change all CSS settings of specific components like the header, the content area and so on. No other framework allows changing its CSS settings without actually writing CSS code.

Other features:

As jQWidgets has a wide range of 49 widgets, it also brings a lot of other features aside from docking (like tabs mentioned one section earlier). Analyzing all these features is beyond the scope of this thesis. However, there are some features that can be specifically useful for docking. These features include save/restore functionality of the current layout, enabling and disabling of docking, collapsable views as well as an event handler for docking events like moving a view.

Problems

Browser compatibilty:

All the above described features were tested and did not yield any error in either of the browsers.

Documentation:

While jQWidgets provides a detailed API documentation, it also has demos with their respective source code available at their webpage [1]. These demos also provide a link to the respective API reference and compared to other frameworks, this all-in-one documentation is really time saving as you have everything at one place to look up.

[1]http://www.jqwidgets.com/themebuilder/

[1]http://www.jqwidgets.com/jquery-widgets-demo/

Implementation:

The reference implementation of jQWidgets required 80 lines of JavaScript code and 45 lines of HTML code. I had no major problems implementing all the desired functionality. Only one issue that can be confusing was the fact that it is possible to flag a view as resizable, however while the view is in docked mode it can not be resized at all. The only way to resize a view is to make it float, but as soon as it is docked into the docking region, it gets resized to the size of the docking region.

4.5 Others

The following list is an overview of other web-based frameworks that offer (sometimes limited) docking features but are not implemented in the reference implementation.

Dojo Toolkit [The Dojo Foundation, 2014]	• based on JavaScript/jQuery/HTML5 • built in 2D graphics API with charting components like Bar, Line, Pie, Scatter, etc. • special API for data that allows drag and drop, filter, pagination, etc. on tables respectively data grids • resizing, tabbing, multiple views, drag and drop, special docking functionality withl dojox API
Sencha ExtJS [Sencha, 2014]	• based on JavaScript/HTML5 • commercial license, but also open-source license if creating an open source application • over 150 additional widgets • resizing, docking, multiple views
Telerik Kendo UI [Telerik, 2014]	• based on JavaScript/jQuery/HTML5 • commercial license with the option for an open-source license but with reduced number of widgets • over 70 additional widgets

EclipseRAP [Eclipse Foundation, 2014a]	• based on JavaScript/jQuery/HTML5 • open source • similar to EclipseRCP [Eclipse Foundation, 2014b] (API of Eclipse IDE Interface) but for web technologies • very sophisticated • no specialized docking feature

CHAPTER 5

Overview of Java-based frameworks

This chapter presents different frameworks for Java. Using the Java Virtual Machine for execution, these frameworks run cross-platform. For the reference implementation I used the Java Platform 8u25 (JDK as well as JRE). The computer system in use was an Intel Core i5 CPU @ 2.67 GHz with 4 GB RAM and Windows 7 SP1.

5.1 MyDoggy

General

MyDoggy [de Caro, 2010] is mainly designed for the needs of docking in a Java environment but can also be seen as a window management framework for secondary windows as its features are not strictly restricted to docking functionalities. It is licensed under the GNU Lesser General Public License [1] and can therefore be regarded as open-source software. MyDoggy is based on Swing. Downloading MyDoggy from its official webpage [2] provides one with all necessary JAR archives to make use of the framework as well as a tutorial set which introduces the user to how MyDoggy works. Following the documentation outlined on the webpage can also give you good insight of how to use MyDoggy. MyDoggy is the oldest framework as the last update happened in December 2010.

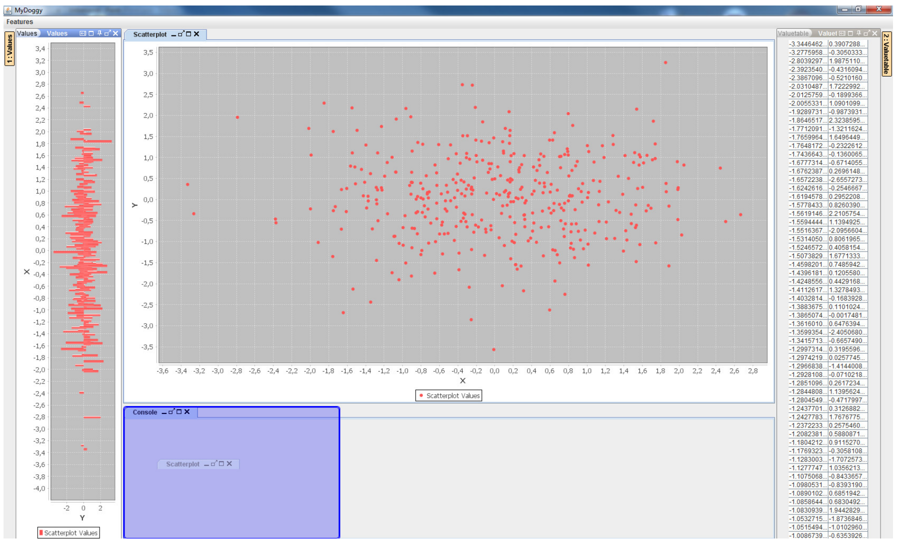
Features

Docking:

MyDoggy's docking feature evaluates the position of a view depending on where it is actually placed on the Java frame. Highlighting where a view will be placed upon dropping works with

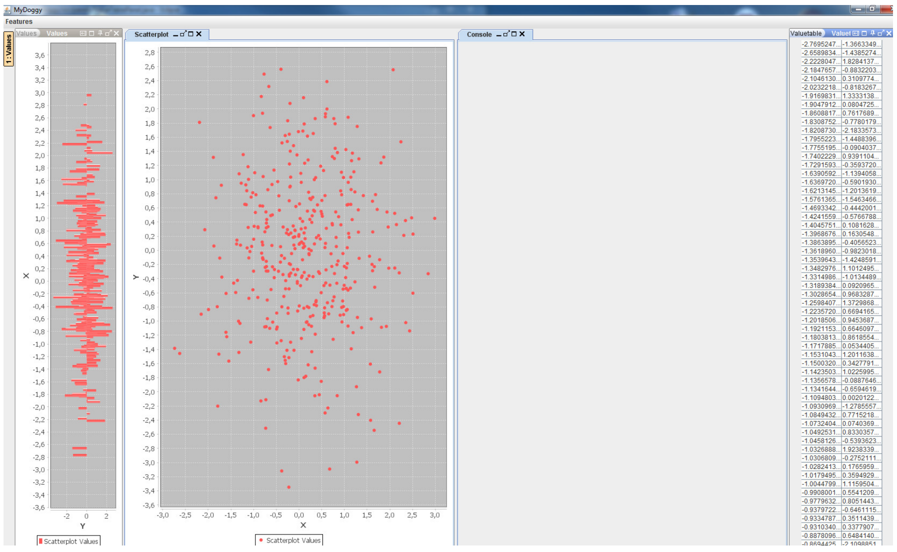
[1]`http://www.gnu.org/copyleft/gpl.html`
[2]`http://mydoggy.sourceforge.net/`

visual feedback in form of transparent regions. Figure 5.1 gives an example of the docking functionality.



-3.3446462...	0.3907288...
-3.2775958...	-0.3050333...
-2.8039297...	1.9875110...
-2.3923540...	-0.4316094...
-2.3867096...	-0.5210160...
-2.0310487...	1.7222992...
-2.0125759...	-0.1899366...
-2.0055331...	1.0901099...
-1.9289731...	-0.9873931...
-1.8646517...	2.3238595...
-1.7712091...	-1.3211624...
-1.7659964...	1.6496449...
-1.7648172...	-0.2322612...
-1.7436643...	-0.1360065...
-1.6777314...	-0.6714055...
-1.6762387...	0.2696148...
-1.6572238...	-2.6557273...
-1.6242616...	-0.2546667...
-1.6194578...	0.2952208...
-1.5778433...	0.8260390...
-1.5619146...	2.2105754...
-1.5594444...	1.1394925...
-1.5516367...	-2.0956604...
-1.5314050...	0.8061965...
-1.5246572...	0.4058154...
-1.5073829...	1.6771333...
-1.4598201...	0.7485942...
-1.4396181...	0.1205580...
-1.4248556...	0.4429168...
-1.4112617...	1.3278493...
-1.4032814...	-0.1683928...
-1.3883675...	0.1101024...
-1.3865074...	-0.0017481...
-1.3616010...	0.6476394...
-1.3599354...	-2.4050680...
-1.3415713...	-0.6657490...
-1.2997314...	0.3195596...
-1.2974219...	0.0257745...
-1.2966838...	-1.4144008...
-1.2928108...	-0.0710218...
-1.2851096...	0.2617234...
-1.2844808...	1.1395624...
-1.2804549...	-0.4717997...
-1.2437701...	0.3126882...
-1.2427783...	1.7676775...
-1.2372233...	0.2575460...
-1.2082381...	0.5880871...
-1.1804212...	0.9115270...
-1.1769323...	-0.3058108...
-1.1283003...	-1.7072573...
-1.1277747...	1.0356213...
-1.1075068...	-0.8433657...
-1.0980531...	-0.8393190...
-1.0890102...	0.6851942...
-1.0858644...	0.6830492...
-1.0830939...	1.9442829...
-1.0532715...	-1.8736846...
-1.0515494...	-1.0102960...
-1.0086739...	-0.6353926...

Figure 5.1: Clicking the scatterplot tab and dragging it to the left of the console view. Result of this operation is described in Figure 5.2



-2.7695247...	-1.3663349...
-2.6589834...	-1.4385274...
-2.2228047...	1.8284137...
-2.1847657...	-0.8832203...
-2.1046130...	0.3109774...
-2.0232218...	-0.8183267...
-1.9169831...	1.3333138...
-1.9047912...	0.0804725...
-1.8608817...	0.7617689...
-1.8308752...	-0.7780179...
-1.8208730...	-2.1833573...
-1.7955223...	-1.4488396...
-1.7755195...	-0.0904037...
-1.7402229...	0.9391104...
-1.7291593...	-0.3593720...
-1.6390592...	-1.1394058...
-1.6369720...	-0.5901930...
-1.6213145...	-1.2013619...
-1.5761365...	-1.5463466...
-1.4693342...	-0.4442001...
-1.4241559...	-0.5766788...
-1.4045751...	0.1081628...
-1.3968676...	0.1630548...
-1.3863895...	-0.4056523...
-1.3618960...	-0.9823018...
-1.3539643...	-1.4248591...
-1.3482976...	1.1012495...
-1.3314986...	-1.0134489...
-1.3189384...	0.0920965...
-1.3028654...	0.9683287...
-1.2598407...	1.3729868...
-1.2235720...	0.6694165...
-1.2018506...	0.9453687...
-1.1921153...	0.6646097...
-1.1803813...	0.8618554...
-1.1717885...	0.0534405...
-1.1531043...	1.2011638...
-1.1500320...	0.3427791...
-1.1423503...	1.0225995...
-1.1356578...	-0.0887646...
-1.1341644...	-0.6594619...
-1.1094803...	0.0020122...
-1.0930969...	-1.2785557...
-1.0849432...	0.7715218...
-1.0732404...	0.0740369...
-1.0492531...	0.8330357...
-1.0458126...	-0.5393623...
-1.0326888...	1.9238339...
-1.0306809...	-0.2752111...
-1.0282413...	0.1765959...
-1.0179495...	0.3594929...
-1.0044799...	1.1159504...
-0.9908001...	0.5541209...
-0.9779632...	0.8051443...
-0.9379722...	-0.6461115...
-0.9334787...	0.3511439...
-0.9310340...	0.3377907...
-0.8878096...	0.6484140...
-0.8694425...	-2.1098851...

Figure 5.2: The result of the dragging (and subsequently) dropping operation from Figure 5.1

There are some rules to docking in MyDoggy which were not readily clear to me at first glance. Tool windows, in Figure 5.1 the left and right views (values and valuetable), can just be docked to other tool windows, whereas "normal" views, like the scatterplot or console view, can just be docked to other "normal" views. If highlighting takes up the whole view, like for example the whole console view in Figure 5.2, then these two views grow together to one view with two tabs.

Floating Dialogs:

All views can be made floating. Tool windows can directly be dragged onto “normal” views and are then floating. “Normal” views have to be detached via the context menu to become floating windows.

Tabbed Panels:

“Normal” views as well as tool windows can be tabbed just to the same window type as docking. Additionally, there is a context menu for each tab that also changes, depending on the view type.

Different modes for UI behaviour:

When setting up the “normal” views one can choose between different window modes. The main effect of these modes is to make views either be separated but docked to each other (MultisplitContent Mode), tabbed (TabbedContentMode) or floating (DesktopContentMode) at initialization.

Groups:

Tool windows can be grouped up, so if a tool window is opened, than all other tool windows that belong to the same group open up, too.

Problems

Documentation:

The documentation on the webpage describes single features like ToolWindow, ContentManager and ResourceManager in a very rudimentary way. Unfortunately there is no detailed API documentation.

Implementation:

A tutorial, which is included in the software package of the framework, compensates for the lack of a proper API documentation, and - following some basic examples in the tutorial - one can become familiar with the API of MyDoggy.

5.2 DockingFrames

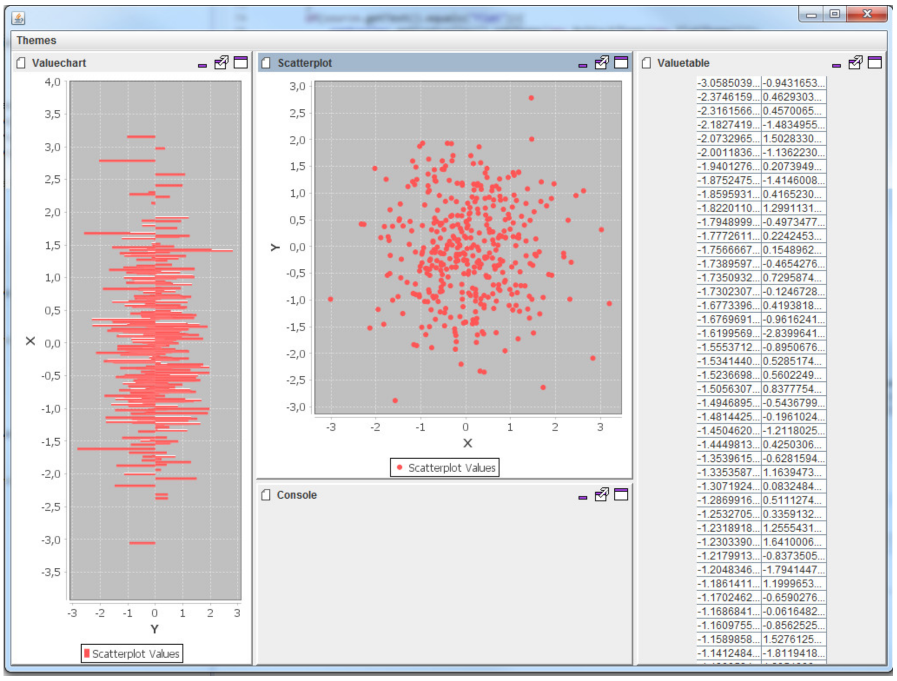
General

DockingFrames [Sigg, 2014] is an open-source framework that is dedicated for docking features. It is based on Swing and licensed under the LGPL 2.1 [1]. To use DockingFrames, the common and core library provided in the download section of the framework's webpage [1] has to be included in the project where it is to be used.

Features

Docking:

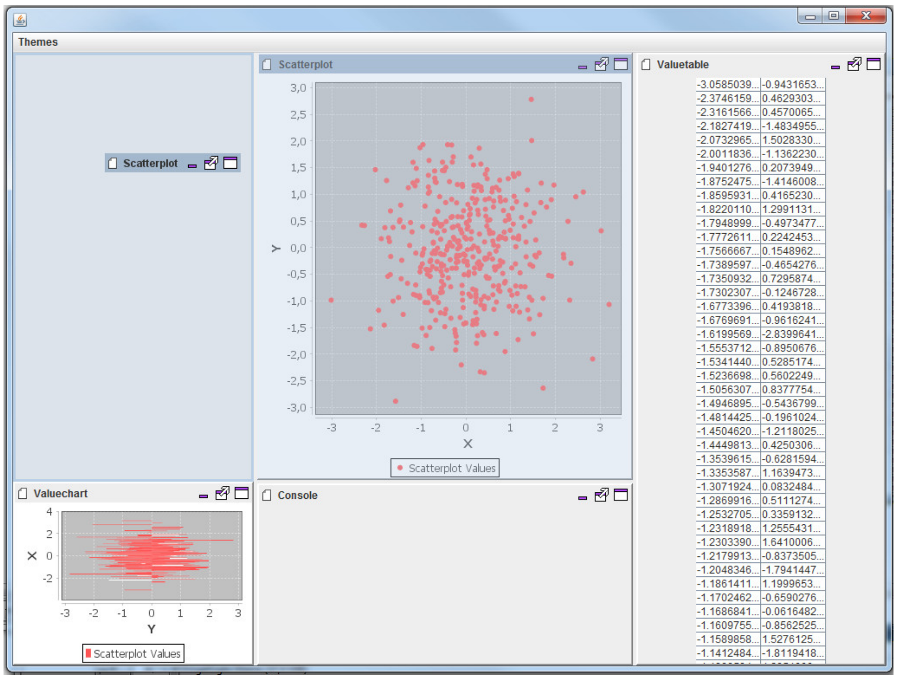
First of all, DockingFrames has a very intuitive preview feature when dragging and dropping a view to see where the view will be placed. As will be mentioned later in in this section, DockingFrames offers different themes which also change the behaviour and design of this docking preview but it works well in either theme. Aside from that, docking works like in most other frameworks, by dragging and dropping a view to another view. Then the preview feature highlights where the view is placed relatively to the view being hovered with the mouse, and adapts the other view(s) immediately while dragging the view. Figures 5.3 and 5.4 show the docking process including the preview function.



-3.0585039...	-0.9431653...
-2.3746159...	0.4629303...
-2.3161566...	0.4570065...
-2.1827419...	-1.4834955...
-2.0732965...	1.5028330...
-2.0011836...	-1.1362230...
-1.9401276...	0.2073949...
-1.8752475...	-1.4146008...
-1.8595931...	0.4165230...
-1.8220110...	1.2991131...
-1.7948999...	-0.4973477...
-1.7772611...	0.2242453...
-1.7566667...	0.1548962...
-1.7389597...	-0.4654276...
-1.7350932...	0.7295874...
-1.7302307...	-0.1246728...
-1.6773396...	0.4193818...
-1.6769691...	-0.9616241...
-1.6199569...	-2.8399641...
-1.5553712...	-0.8950676...
-1.5341440...	0.5285174...
-1.5236698...	0.5602249...
-1.5056307...	0.8377754...
-1.4946895...	-0.5436799...
-1.4814425...	-0.1961024...
-1.4504620...	-1.2118025...
-1.4449813...	0.4250306...
-1.3539615...	-0.6281594...
-1.3353587...	1.1639473...
-1.3071924...	0.0832484...
-1.2869916...	0.5111274...
-1.2532705...	0.3359132...
-1.2318918...	1.2555431...
-1.2303390...	1.6410006...
-1.2179913...	-0.8373505...
-1.2048346...	-1.7941447...
-1.1861411...	1.1999653...
-1.1702462...	-0.6590276...
-1.1686841...	-0.0616482...
-1.1609755...	-0.8562525...
-1.1589858...	1.5276125...
-1.1412484...	-1.8119418...

Figure 5.3: Initial layout created with DockingFrames.

[1]`https://www.gnu.org/licenses/lgpl-2.1.html`
[1]`http://dock.javaforge.com/download.html`



-3.0585039...	-0.9431653...
-2.3746159...	0.4629303...
-2.3161566...	0.4570065...
-2.1827419...	-1.4834955...
-2.0732965...	1.5028330...
-2.0011836...	-1.1362230...
-1.9401276...	0.2073949...
-1.8752475...	-1.4146008...
-1.8595931...	0.4165230...
-1.8220110...	1.2991131...
-1.7948999...	-0.4973477...
-1.7772611...	0.2242453...
-1.7566667...	0.1548962...
-1.7389597...	-0.4654276...
-1.7350932...	0.7295874...
-1.7302307...	-0.1246728...
-1.6773396...	0.4193818...
-1.6769691...	-0.9616241...
-1.6199569...	-2.8399641...
-1.5553712...	-0.8950676...
-1.5341440...	0.5285174...
-1.5236698...	0.5602249...
-1.5056307...	0.8377754...
-1.4946895...	-0.5436799...
-1.4814425...	-0.1961024...
-1.4504620...	-1.2118025...
-1.4449813...	0.4250306...
-1.3539615...	-0.6281594...
-1.3353587...	1.1639473...
-1.3071924...	0.0832484...
-1.2869916...	0.5111274...
-1.2532705...	0.3359132...
-1.2318918...	1.2555431...
-1.2303390...	1.6410006...
-1.2179913...	-0.8373505...
-1.2048346...	-1.7941447...
-1.1861411...	1.1999653...
-1.1702462...	-0.6590276...
-1.1686841...	-0.0616482...
-1.1609755...	-0.8562525...
-1.1589858...	1.5276125...
-1.1412484...	-1.8119418...

Figure 5.4: Dragging the scatterplot view above the “Valuechart“ view triggers the preview function and arranges all views in the way they will look if the view is dropped at this location.

Floating Dialogs:

As soon as a view is moved outside of the main application window it becomes floating. Views can also be set floating programmatically.

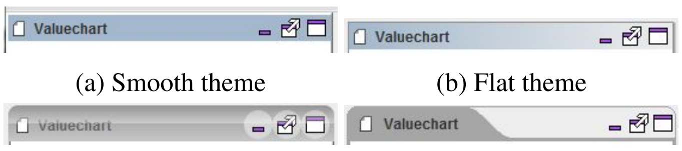
Tabbed Panels:

Tabbed panels are also possible and can be activated, like in most other frameworks, programmatically or during runtime by dragging and dropping one view into another view.

Themes:

Four different and easy to change themes distinguish DockingFrames from the other frameworks when it comes to customization. Figure 5.5 shows how the different themes look like.

One can also change the color schemes of the different themes. Combined with different look-and-feel options offered by Swing, this enables very special and distinctive designs for styling different views.



(a) Smooth theme (b) Flat theme

(c) Bubble theme (d) Eclipse theme

Figure 5.5: Themes in DockingFrames.

Problems

Documentation:

On the webpage of the DockingFrames project [2] various forms of online resources can be found. The API documentation as well as guides to set up a project with DockingFrames (which also goes into the detail of the framework architecture) and code examples can be found here. Also the Google Code repository of DockingFrames [3] provides a wide range of examples.

Implementation:

Implementing the demo for the prototype was not that hard but it can be demanding when starting to learn how the framework works and diving into the more advanced features like customizing the so-called DockStation, which handles options for the look and feel. The author of the framework explains the complexity with being the only fulltime developer on the project, backwards compatibility and flexibility of the framework [4].

5.3 VLDocking

General

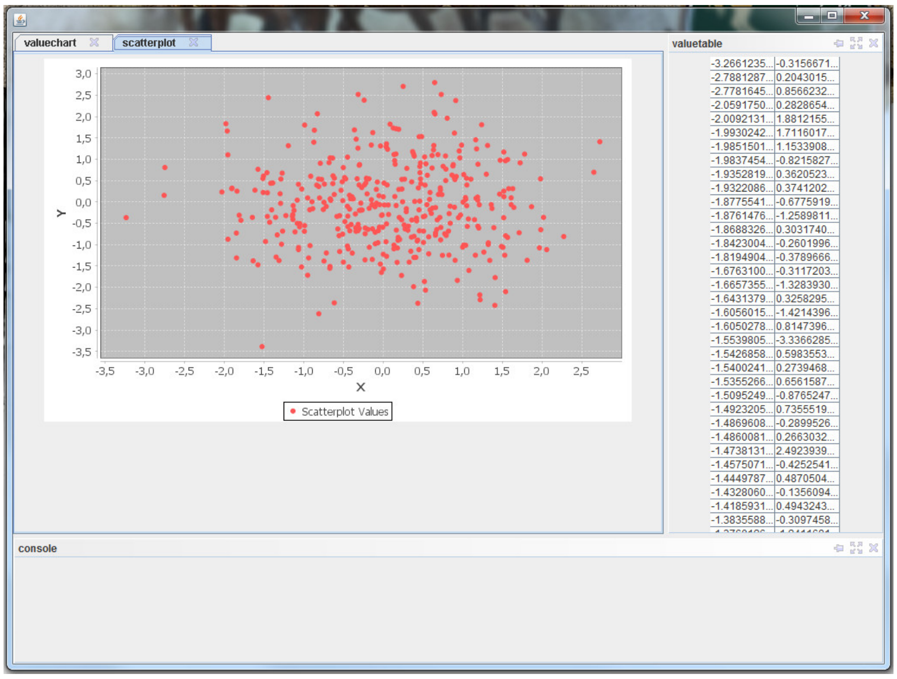
VLDocking [Chamontin, 2013] is an open source framework dedicated for docking needs and is licensed as LGPL [1]. It is one of the older projects as the last update happened in June 2013.

Features

Docking:

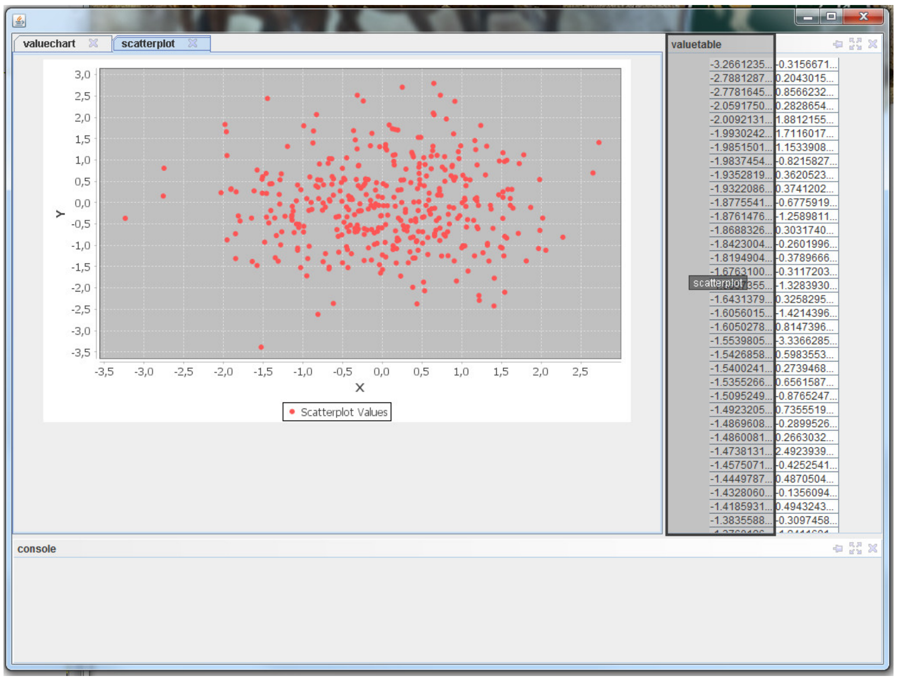
Docking works similar to most other frameworks. Dragging and dropping a view triggers a highlighting function that displays if the view is positioned left, right, up or down, depending on which region is highlighted in the other window. If the highlighting covers the whole view, then the view will be tabbed. Figure 5.6 and 5.7 show dragging and dropping of a view and the highlighting function.

[2] `http://dock.javaforge.com/doc.html`
[3] `https://code.google.com/p/docking-frames/source/browse/`
[4] `http://dock.javaforge.com/help.html`
[1] `https://www.gnu.org/licenses/lgpl.html`



-3.2661235...	-0.3156671...
-2.7881287...	0.2043015...
-2.7781645...	0.8566232...
-2.0591750...	0.2828654...
-2.0092131...	1.8812155...
-1.9930242...	1.7116017...
-1.9851501...	1.1533908...
-1.9837454...	-0.8215827...
-1.9352819...	0.3620523...
-1.9322086...	0.3741202...
-1.8775541...	-0.6775919...
-1.8761476...	-1.2589811...
-1.8688326...	0.3031740...
-1.8423004...	-0.2601996...
-1.8194904...	-0.3789666...
-1.6763100...	-0.3117203...
-1.6657355...	-1.3283930...
-1.6431379...	0.3258295...
-1.6056015...	-1.4214396...
-1.6050278...	0.8147396...
-1.5539805...	-3.3366285...
-1.5426858...	0.5983553...
-1.5400241...	0.2739468...
-1.5355266...	0.6561587...
-1.5095249...	-0.8765247...
-1.4923205...	0.7355519...
-1.4869608...	-0.2899526...
-1.4860081...	0.2663032...
-1.4738131...	2.4923939...
-1.4575071...	-0.4252541...
-1.4449787...	0.4870504...
-1.4328060...	-0.1356094...
-1.4185931...	0.4943243...
-1.3835588...	-0.3097458...

Figure 5.6: Layout created with VLDocking.



-3.2661235...	-0.3156671...
-2.7881287...	0.2043015...
-2.7781645...	0.8566232...
-2.0591750...	0.2828654...
-2.0092131...	1.8812155...
-1.9930242...	1.7116017...
-1.9851501...	1.1533908...
-1.9837454...	-0.8215827...
-1.9352819...	0.3620523...
-1.9322086...	0.3741202...
-1.8775541...	-0.6775919...
-1.8761476...	-1.2589811...
-1.8688326...	0.3031740...
-1.8423004...	-0.2601996...
-1.8194904...	-0.3789666...
-1.6763100...	-0.3117203...
355...	-1.3283930...
-1.6431379...	0.3258295...
-1.6056015...	-1.4214396...
-1.6050278...	0.8147396...
-1.5539805...	-3.3366285...
-1.5426858...	0.5983553...
-1.5400241...	0.2739468...
-1.5355266...	0.6561587...
-1.5095249...	-0.8765247...
-1.4923205...	0.7355519...
-1.4869608...	-0.2899526...
-1.4860081...	0.2663032...
-1.4738131...	2.4923939...
-1.4575071...	-0.4252541...
-1.4449787...	0.4870504...
-1.4328060...	-0.1356094...
-1.4185931...	0.4943243...
-1.3835588...	-0.3097458...

Figure 5.7: Dragging the scatterplot view to the left of the valuetable view triggers left highlighting and telling the user that the view will be placed to the left of the valuetable's view.

Floating Dialogs:

With VLDocking views cannot be made floating through interacting with the windows. Other frameworks require dragging a view out of the main application window, this does not work with VLDocking. However, there is the possibility to set views floating programmatically and therefore one could make the currently focused view floating through a menu option.

Tabbed Panels:

Tabbed panels are also possible and can be activated, like in most other frameworks, programmatically or during runtime by dragging and dropping one view into another view.

Themes:

Like DockingFrames, VLDocking offers different themes that mainly change the layout of space between views. This space can be decorated with shadows, dots or a standard blank space.

Problems

Documentation:

The webpage of VLDocking [1] provides a step-by-step tutorial for all features and provides code examples as well as figures for demonstration. Other frameworks often provide demo versions where it is possible to launch a working example, VLDocking does not offer such examples.

Implementation:

I had no major problems while implementing basic docking features of VLDocking. Especially laying out multiple views can be done very quickly and requires few lines of code. The only issue that I consider a limitation is the lack of proper floating windows by default. As mentioned above, a window can be made floating only programmatically.

5.4 Eclipse RCP

General

Eclipse [Eclipse Foundation, 2014b] enables developers to reuse its libraries and API for creating user interfaces. According to the documentation [2] it is designed to serve as an open-tools platform and offers the minimalistic set of plugins that are necessary to create Rich Client Platforms (RCPs). This platform does not only feature a docking framework but a whole set of user interface elements as well as an Update Manager, Text-Editor, Cheat Sheets for guiding users, Resource Management, Console view and many other features. It is also highly modular and the plugin architecture allows flexible and specialized applications. To set up an Eclipse RCP

[1] `https://code.google.com/p/vldocking/`

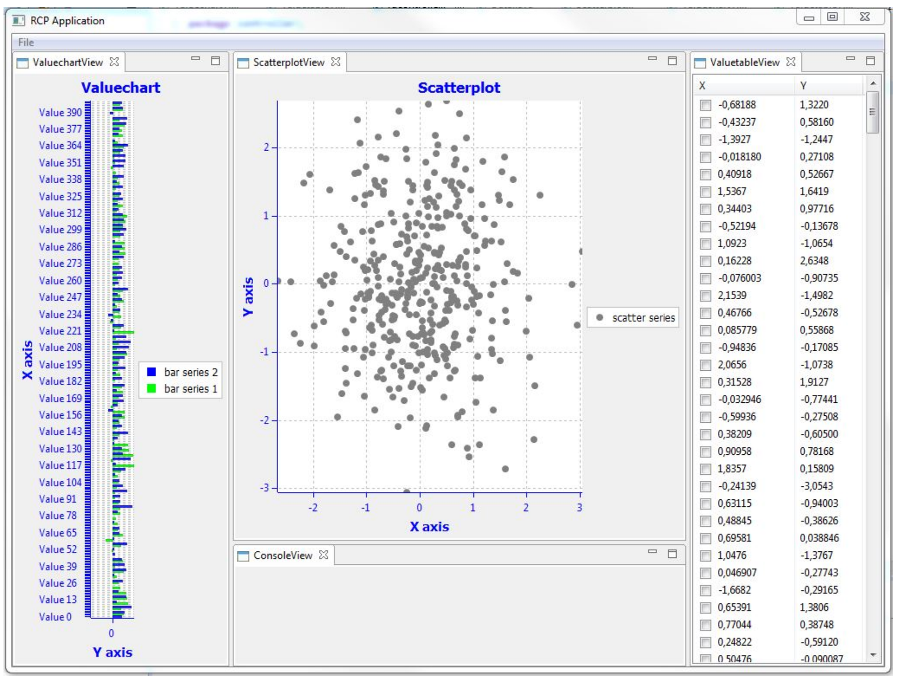
[2] `http://wiki.eclipse.org/Rich_Client_Platform/FAQ`

application one has to install the Eclipse plug-in development environment through the Eclipse IDE and afterwards create a new plug-in project with the enabled option of creating a rich client application.

Features

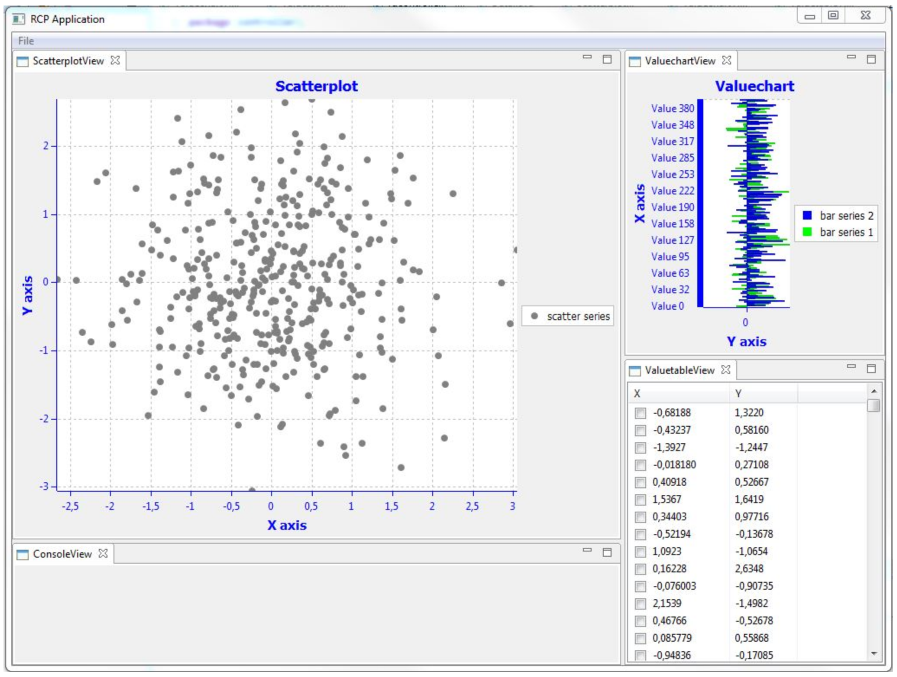
Docking:

Docking with Eclipse RCP is straightforward. If one view is dragged over another view and consequently dropped there, the display area of the other view is split in half and shares the display space with the dropped view. Figures 5.8 and 5.9 show the process of docking in Eclipse RCP.



X	Y
-0,68188	1,3220
-0,43237	0,58160
-1,3927	-1,2447
-0,018180	0,27108
0,40918	0,52667
1,5367	1,6419
0,34403	0,97716
-0,52194	-0,13678
1,0923	-1,0654
0,16228	2,6348
-0,076003	-0,90735
2,1539	-1,4982
0,46766	-0,52678
0,085779	0,55868
-0,94836	-0,17085
2,0656	-1,0738
0,31528	1,9127
-0,032946	-0,77441
-0,59936	-0,27508
0,38209	-0,60500
0,90958	0,78168
1,8357	0,15809
-0,24139	-3,0543
0,63115	-0,94003
0,48845	-0,38626
0,69581	0,038846
1,0476	-1,3767
0,046907	-0,27743
-1,6682	-0,29165
0,65391	1,3806
0,77044	0,38748
0,24822	-0,59120
0,50476	-0,090087

Figure 5.8: Initial layout of views with Eclipse RCP



X	Y
-0,68188	1,3220
-0,43237	0,58160
-1,3927	-1,2447
-0,018180	0,27108
0,40918	0,52667
1,5367	1,6419
0,34403	0,97716
-0,52194	-0,13678
1,0923	-1,0654
0,16228	2,6348
-0,076003	-0,90735
2,1539	-1,4982
0,46766	-0,52678
0,085779	0,55868
-0,94836	-0,17085

Figure 5.9: Dropping the valuechart view in the valuetable view

Floating Dialogs:

Floating Dialogs are also supported. Views can be set floating programmatically or by dragging a view outside of the application window. A view can also be reattached inside the application window.

Tabbed Panels:

Eclipse RCP also offers the possibility for tabbed panels. Views can be tabbed by dropping one view in the tab area of another view. Note that this default behaviour seems to be unique to Eclipse RCP (and Netbeans RCP, see section 5.5), as most other frameworks tab one view if another view is dropped on top of it (this would result in docking views in Eclipse RCP and Netbeans RCP).

Perspectives:

Perspectives are the way how the user can organise views on the screen. Views always exist in perspectives. Multiple perspectives with different views can be defined for different use cases within one application. Perspectives can be programmatically pre-defined or the user creates a layout of views and saves it as a custom perspective. A perspective can also be reset to its original layout to roll back layout changes made by the user.

Problems

Documentation:

There are many online resources available for Eclipse RCP. The following page `http://wiki.eclipse.org/index.php/Rich_Client_Platform` is highly recommended: It provides tutorials, examples, links to other resources as well as UI guidelines and further resources.

Implementation:

Implementing the demo for the prototype was quite easy, as Eclipse provides several templates that help understand how rich client applications are built. Also perspectives allow very sophisticated layouts with just a few lines of code, so the developer can focus on the actual features of his or her application without investing too much time in setting up the user interface.

5.5 Netbeans RCP

General

Like Eclipse, Netbeans offers its libraries and API for creating user interfaces to the developer as an open tools platform. Netbeans RCP [Netbeans RCP, 2014] has a similar architecture to Eclipse RCP based on plug-in and aims to be very modular. Also the feature list of both frameworks is nearly equal. The main differences between Eclipse RCP and Netbeans RCP are:

- Netbeans RCP is based on Swing, while Eclipse RCP is based on SWT
- different look-and-feel
- currently no perspectives for Netbeans RCP as in Eclipse RCP
- Netbeans RCP has many default menu options when creating an application
- Netbeans IDE provides built-in ways to create components (for example view) of rich client application through a GUI

Features

Docking:

Docking is similar to Eclipse RCP. If one view is dragged over another view and consequently dropped there, the display area of the other view is split in half and shares the display space with the dropped view. While dragging, Netbeans RCP shows a minimized transparent version of the dragged view. Figures 5.10 and 5.11 show the process of docking in Netbeans RCP.

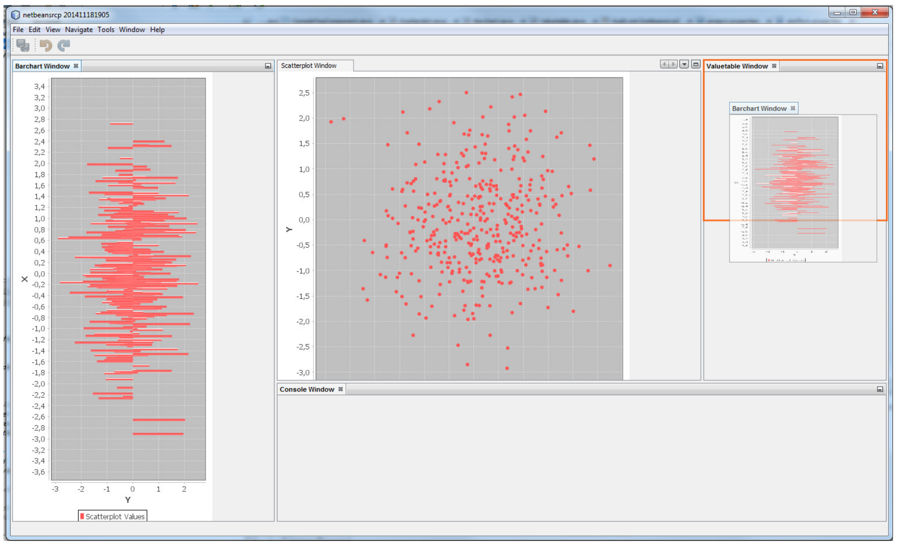


Figure 5.10: Inital layout of views with Netbeans RCP

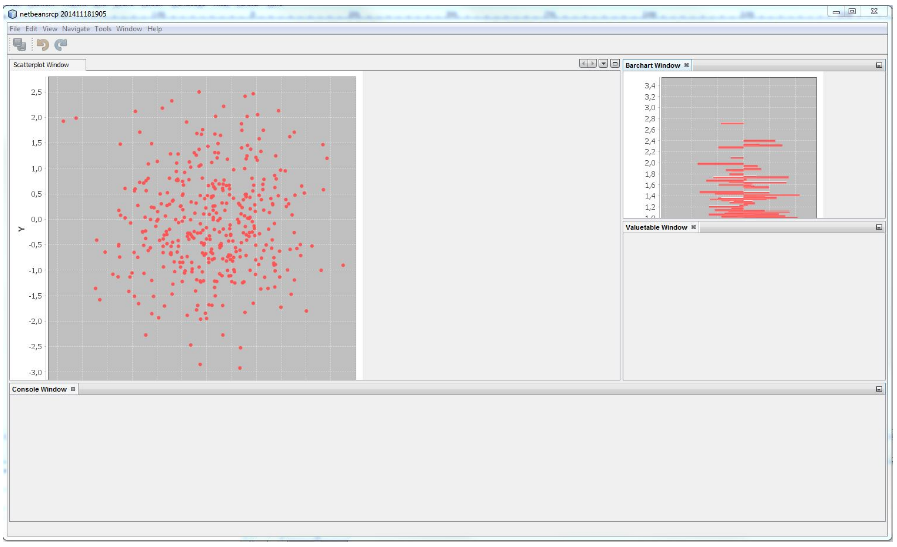


Figure 5.11: Dropping the valuechart view in the valuetable view

Floating Dialogs:

As soon as a view leaves the main application window and is dropped there, it becomes floating.

Tabbed Panels:

Panels can be tabbed by dragging one view into the tab area of another view. The user can decide on which position the new tab will be placed. If placed to the right, then the view will be the new right tab of the nearest tab and so on.

Menus:

Netbeans RCP provides default menu functionality when setting up a new rich client application. This includes a whole menu tab for managing views, a tools tab where you can set keymaps, appearance, look and feel and other useful options, a view menu tab with the functionality for setting up a toolbar with functions like undo or redo and a file menu tab with save, print and exit commands.

Problems

Documentation:

Netbeans provides very solid resources for its RCP framework, like tutorials and examples (`https://netbeans.org/features/platform/index.html`) as well as a complete API documentation.

Implementation:

Setting up an application based on Netbeans RCP is straightforward, most notably because of the possibility to create user interfaces interactively using the mouse. Adding a view to an application for example just requires two clicks. Like Eclipse RCP, this helps developers a lot to focus on the main features of their application.

CHAPTER 6

Comparison of web-based frameworks

6.1 Overview

Chapter 4 presented different web-based frameworks. In this chapter I compare these frameworks to each other. The comparison is based on a feature matrix to identify strengths and limitations in the range of features a framework provides, as well as several criteria like performance or lines of code and a list of advantages and disadvantages for every framework.

6.2 Feature Analysis

First I want to give an overview on the different features of each framework. This will be done with a feature matrix, where each column represents a framework and each row a feature. Note that the matrix is divided in two groups via a line in bold print. The group above the line includes features that are directly related to docking, whereas the group below the line represents features of general usage. Also keep in mind that some frameworks have other features, either specialized and more general, which are not included in this matrix, so if you are interested in further details, refer to chapter 4.

6.3 Criteria

The following comparisons are split up in quantifiable criteria and pros/cons for each framework. Quantifiable criteria include number of features, both docking as well as more general features (based on feature matrix), Lines of Code to achieve the functionality which is provided by the prototype implementation and performance measurement in milliseconds for executing the framework specific JavaScript code. My3 test system for the performance measurement has the following specifications: Intel Core i5 @ 2.67 GHz, 4 GB RAM, Win 7 SP1 64 Bit. Lines of Code include the function calls for the visualisations as well as the table of values and some calculations for site measurements like width and height. As these non-docking related

	DockSpawn	wcDocker	jQWidgets	jQuery UI Layout	Eclipse RAP
Splitter	+	+	-	+	+
Tabs	+	+	+	+	+
Floating Windows	+	+	+	-	+
Resizing	+	+	+	+	+
Collapsible Views	-	-	+	+	+
Docking	+	+	+	-	-
Save/Load Layout	buggy	+	+	+	+
Multiple Themes	-	+	+	+	+

Table 6.1: Feature matrix for web-based frameworks.

lines of code are roughly the same for each framework, they are included in these lines of code comparison.

6.4 Dock Spawn

Quantity Criteria Overview	
Features(docking)	5
Features(general)	1 (buggy)
Lines of Code	63 lines of JavaScript, 10 lines of HTML code
Performance	43ms in Firefox

Pros	**Cons**
simple and intuitive (boxes) UI for the user	not entirely bug-free
framework runs very fast compared to other frameworks	not many features aside from docking compared to other frameworks
few lines of code to achieve basic docking functionality	no regular updates since February 2014
predefined CSS styling with contrast rich colors	

6.5 jQuery UI Layout

Quantity Criteria Overview	
Features(docking)	4
Features(general)	2
Lines of Code	42 lines of JavaScript, 28 lines of HTML code
Performance	110ms in Firefox

Pros	Cons
tested features were bug-free and stable	no dedicated docking feature
code for framework runs fast compared to other frameworks	
once familiar with the syntax, few lines of code are needed to set up multiple views	
a wide range of general UI features	
regular updates	

6.6 jQWidgets

Quantity Criteria Overview	
Features(docking)	5
Features(general)	2
Lines of Code	80 lines of JavaScript, 45 lines of HTML code
Performance	221ms in Firefox

Pros	Cons
basic docking features do not require many code lines	more sophisticated layout and functionality does require more code compared to other frameworks
a wide range of features	no OS license for commercial use
tested features were bug-free and stable	code for framework runs slower compared to other frameworks

6.7 wcDocker

Quantity Criteria Overview	
Features(docking)	5
Features(general)	2
Lines of Code	119 lines of JavaScript, 1 line of HTML code
Performance	166 ms in Firefox

Pros	Cons
some features, like a context menu combined with docking, are unique to wcDocker	requires knowledge of some features to properly use them, for example that a right click in a view opens a context menu for panels and a right click on tabs opens a context menu for tabs
a wide range of features especially designed for docking	
tested features were bug-free and stable	
regular updates and active development of new features	

6.8 Discussion

As seen in the previous chapters, there are quite a few frameworks for either Java- and Web-Technology. During the implementation of docking frameworks based on web-technologies like HTML5, CSS, JavaScript, etc. it quickly became clear that these frameworks were in fact pretty easy to implement and often required very few lines of code. The most notable differences of the web-based frameworks lay in the diversity of features. Some of these frameworks,like for example DockSpawn or wcDocker, had very specialised docking features. In particular, the drag and drop feature with subsequently snapping windows to each other, worked very well in a web-browser environment. On the other hand, these specialised frameworks did not offer many other features, whereas jQuery UI Layout [Balliano, Dalman, 2014] for example offered a much richer set of tools for window- and display space management but was lacking docking features. The D3.js framework [Bostock et al., 2011] used for web visualizations worked very well with the tested frameworks, except for some issues especially with DockSpawn [Code Respawn, 2012] and jQWidgets [jQWidgets, 2014] where the content of the visualization was overlapping other views when resizing the windows.

CHAPTER 7

Comparison of Java-based frameworks

7.1 Overview

Chapter 5 presented different Java-based frameworks. In this chapter I compare these frameworks to each other. The comparison is based on a feature matrix to identify strengths and limitations in the range of features a framework provides, as well as several criteria like performance or lines of code and a list of advantages and disadvantages for every framework.

7.2 Feature Analysis

Like chapter 6, the following matrix gives an overview of different features for Java-based frameworks. For details regarding the frameworks refer to chapter 5.

	MyDoggy	VLDocking	DockingFrames	EclipseRCP	NetbeansRCP
Splitter	+	+	+	+	+
Tabs	+	+	+	+	+
Floating Windows	+	+	+	+	+
Resizing	+	+	+	+	+
Collapsible Views	+	+	+	+	+
Docking	+	+	+	+	+
Save/Load Layout	+	+	+	+	+
Sliding	+	+	+	+	+
Grouping	+	+	+	+	+
Multiple Themes	+	+	+	+	+
Layout Reset	-	-	-	+	+
Perspectives	-	+	+	+	-

Table 7.1: Feature matrix for Java-based frameworks.

7.3 Criteria

The following comparison of Java-based frameworks consists of an overview of quantifiable criteria as well as a list of advantages and disadvantages. Quantifiable criteria include number of features, both docking as well as more generalized features, Lines of Code to achieve the functionality which is provided by the prototype implementation and performance measurement in milliseconds for executing the framework-specific code on startup. My test system for the performance measurement had the following specifications: Intel Core i5 @ 2.67 GHz, 4 GB RAM, Win 7 SP1 64 Bit. Note that lines of code include the function calls for the visualizations, the table values, the bar-chart and the console view seen in the prototype implementation. As this code is nearly the same for every framework (except EclipseRCP, which uses SWT instead of Swing), the framework-specific code for the actual features is what makes the difference in the amount of code. Even if there is no specific definition of which features a docking framework should provide, developers of these frameworks all implemented the same features but with slightly different characteristics, especially regarding the drag and drop functionality while docking.

7.4 MyDoggy

Quantity Criteria Overview	
Features(docking)	6
Features(general)	4
Lines of Code	267
Performance	1056 ms

Pros	**Cons**
distinction between normal views and tool-windows	missing perspectives
a rich set of buttons, like undock, floating or hiding predefined	tool windows and normal views are differentiated and cannot be mixed together(for example tabbing toolwindow into normal view)
different UI behaviour modes desktop/tabbed/multisplit	latest release is from December 2010

7.5 DockingFrames

Quantity Criteria Overview	
Features(docking)	6
Features(general)	5
Lines of Code	309
Performance	1087 ms

Pros	Cons
last release from 13th December 2014, regular updates	more complex, so implementation is harder
limited guarantee for backwards compatibility	
different themes changing the style of views	

7.6 VLDocking

Quantity Criteria Overview	
Features(docking)	6
Features(general)	5
Lines of Code	304
Performance	816 ms

Pros	Cons
very easy to set up basic functionality and customize docking look and feel	latest release from June 2013
	floating windows just programmatically possible, not interactively

7.7 Eclipse RCP

Quantity Criteria Overview	
Features(docking)	6
Features(general)	6
Lines of Code	368 (mostly auto-generated code)
Performance	952 ms (evaluated with tracing options for Eclipse RCP applications)

Pros	Cons
full feature range of Eclipse IDE, a complete window management framework	
wide range of other features like Update Manager, Cheat sheets, etc	
different themes, possibility for custom styling via CSS	
predefined templates help setting up a project	

7.8 Netbeans RCP

Quantity Criteria Overview	
Features(docking)	6
Features(general)	5
Lines of Code	601 (mostly auto-generated code)
Performance	1493 ms (evaluated with Netbeans Profiler)

Pros	Cons
full feature range of Netbeans IDE, a complete window management framework	no out-of-the box perspective/workspace feature like Eclipse RCP
wide range of other features like Update Manager, Cheat sheets, etc	
different themes, possibility for custom styling via CSS	
Netbeans IDE has an UI toolkit for creating applications with RCP, nearly no coding required for an application	

7.9 Discussion

Compared with web-based frameworks (see chapter 4) Java-based frameworks have very similar features, which is quite interesting regarding the fact that there is no standard definition of the features of a high-quality docking framework. So Java-based frameworks do not differ very much from the features provided. The most notable difference between all the Java-based frameworks is the look-and-feel of the drag and drop, regarding docking functionality. Some frameworks offer a preview of how the layout will look after docking a window somewhere else, other frameworks just highlight the region where the dragged window will be placed and some frameworks do not show what will happen with the dragged window at all. The interaction while docking was quite fiddly with some frameworks, whereas Eclipse [Eclipse Foundation, 2014b] or Netbeans RCP [Netbeans RCP, 2014] are very easy to interact with. Also, Java-based frameworks are mostly harder to implement than their web-based counterparts in the sense that they require significantly more lines of code. Especially frameworks like Eclipse [Eclipse Foundation, 2014b] or Netbeans RCP [Netbeans RCP, 2014] offer functionality for building up a user interface through their IDEs, which makes implementing these frameworks a quite straightforward task.

CHAPTER 8

Conclusion

This work presents an overview on docking frameworks based on Java and web technology. At the beginning, the reader is introduced to the topic with figures and explanations about how docking can work in a specific environment as well as a basic overview on the thesis. After the basic introduction, I cover theoretical foundations that back up the necessity as well as the efficiency of using docking and multiple views to better understand and comprehend multifaceted data. Based on my experiences while implementing a prototype with different docking frameworks, an overview on these frameworks is given after the theoretical foundations. The thesis concludes with a comparison of the frameworks followed by a short discussion.

Overall, my research has revealed docking as a strong tool to support the user with managing different views. There are also a lot of different frameworks available to set up an application environment with docking features. While these frameworks often provide similar functionalitiy regarding basic features, there are significant differences between:

- look and feel
- docking behaviour
- positioning possiblities
- features that do not relate to docking

The features needed - apart from docking - and the docking behaviour itself mainly influence the decision for one certain framework. The look and feel can be changed individually for nearly all frameworks with different themes. Most of the frameworks allow a wide range of positioning possibilities but some are more restrictive than others. My prototype, provided as supplementary of my thesis, helps in the decision process by giving a first hands-on experience with different frameworks in action.

Apart from commercial frameworks, my thesis mostly covers open source frameworks in detail and especially my prototype is implemented with open source software (except jQWidgets [jQWidgets, 2014]). Commercial frameworks are mentioned, but not analyzed in detail.

Another interesting topic for future work could be the possibility of docking features on mobile devices. Frameworks based on web technology may be the most relevant for this topic, as they already follow the responsive design principle. It will be interesting how these frameworks handle the interaction during the docking process. Furthermore, it would be interesting to analyze frameworks based on other technologies like Microsoft Silverlight and what kind of (docking) features these frameworks provide.

APPENDIX A

Supplementaries

- Video of Dock Spawn to illustrate docking functionality
- Prototype with implementation of the following web-based frameworks:
  - Dock Spawn
  - JQuery UI Layout
  - wcDocker
  - jQWidgets
- Prototype with implementation of the following Java-based frameworks:
  - MyDoggy
  - DockingFrames
  - VLDocking
  - Eclipse RCP
  - Netbeans RCP

Bibliography

Balliano, Dalman (2014). jQuery UI Layout. http://layout.jquery-dev.com. Accessed: 2014-26-07.

Bostock, M., Ogievetsky, V., and Heer, J. (2011). D^3 data-driven documents. *Visualization and Computer Graphics, IEEE Transactions on*, 17(12):2301–2309.

Chamontin (2013). Vldocking. https://code.google.com/p/vldocking. Accessed: 2014-15-07.

Code Respawn (2012). Dock Spawn. www.dockspawn.com. Accessed: 2014-26-07.

de Caro (2010). MyDoggy. http://mydoggy.sourceforge.net/index.html. Accessed: 2014-15-07.

Eclipse Foundation (2014a). Eclipse RAP. http://eclipse.org/rap. Accessed: 2014-05-12.

Eclipse Foundation (2014b). Eclipse RCP. http://www.eclipse.org/downloads/packages/eclipse-rcp-and-rap-developers/lunasr1. Accessed: 2014-28-10.

Houde (2014). Web Cabin Docker. https://github.com/WebCabin/wcDocker. Accessed: 2014-26-07.

Hutchings, D. R. and Stasko, J. (2004). Revisiting display space management: Understanding current practice to inform next-generation design. In *Proceedings of Graphics Interface 2004*, GI '04, pages 127–134, School of Computer Science, University of Waterloo, Waterloo, Ontario, Canada. Canadian Human-Computer Communications Society.

jQWidgets (2014). jQWidgets. www.jqwidgets.com. Accessed: 2014-29-10.

Myers, B. (1988). A taxonomy of window manager user interfaces. *Computer Graphics and Applications, IEEE*, 8(5):65–84.

Netbeans RCP (2014). Netbeans Platform. https://netbeans.org/features/platform. Accessed: 2014-28-10.

Roberts, J. (2007). State of the art: Coordinated multiple views in exploratory visualization. In *Coordinated and Multiple Views in Exploratory Visualization, 2007. CMV '07. Fifth International Conference on*, pages 61–71.

Sencha (2014). Sencha ExtJS. http://www.sencha.com/products/extjs. Accessed: 2014-26-07.

Shibata, H. and Omura, K. (2012). Docking window framework: Supporting multitasking by docking windows. In *Proceedings of the 10th Asia Pacific Conference on Computer Human Interaction*, APCHI '12, pages 227–236, New York, NY, USA. ACM.

Sigg (2014). Docking Frames. http://dock.javaforge.com. Accessed: 2014-15-07.

Telerik (2014). Telerik Kendo UI. http://www.telerik.com/kendo-ui. Accessed: 2014-31-10.

The Dojo Foundation (2014). Dojo toolkit. http://dojotoolkit.org. Accessed: 2014-26-07.

Wang Baldonado, M. Q., Woodruff, A., and Kuchinsky, A. (2000). Guidelines for using multiple views in information visualization. In *Proceedings of the Working Conference on Advanced Visual Interfaces*, AVI '00, pages 110–119, New York, NY, USA. ACM.