Media Informatics

Bachelor Thesis

Curator:
Design Environment for Curating Tabletop Museum Experiences

by

Benjamin Sprengart

8 September 2009

Supervisor:    Prof. Dr. Dieter Wallach
Co-Supervisor: Prof. Judy Kay
Abstract

The purpose of museums is education, study and enjoyment. They aim to make the knowledge and collections accessible to people of all ages and backgrounds. Particularly with regard to special enjoyment, museums install computer terminals and PCs to provide more information and boost their attractiveness.

The use of tabletop displays in museums contributes to this purpose by inviting visitors to share their experiences offered by the tabletop. In contrast to vertical displays, users can approach a tabletop display from various angles and gather around the big display which often forms a centre of attraction.

However, it is a real challenge for exhibition designers and their assistants to create a virtual exhibition for tabletop displays. Depending on the specific topic, different presentations for different target groups should be designed to meet their various backgrounds, interests, and goals. Also, the contents of a virtual exhibition may need to be changed when new acquisitions are made or because the physical exhibition has changed. Exhibition designers are generally not programmers - so there are potential benefits if they can easily create a virtual exhibition.

In this thesis, Curator, a design environment for creating interactive museum experiences for tabletop displays is developed. The program allows exhibition designers and their assistants to react on the needs of the museum’s visitors and to compose virtual exhibitions in an easy manner. With its facilities to compose collections of digital exhibits quickly, it enables the staff to curate many variations of tabletop experiences, to enrich the attractiveness of the museum and to enthral the visitors.
Declaration Of Honour

Herewith I, Benjamin Sprengart, born on 24th September 1985 in Dudweiler, declare honourly that this Bachelor Thesis named

‘Curator: Design Environment for Curating Tabletop Museum Experiences’

is completely my own work and I have not been assisted. No part of this assignment is taken from other people’s work without giving them credit. All references have been clearly cited. I understand that an infringement of this declaration leaves me subject to disciplinary action.

Sydney, September 7, 2009
Place, date

Signature
Contents

Abstract ................................................................. iii
Declaration Of Honour ................................................... v

1. Introduction .......................................................... 1
   1.1. Problem Statement ................................................. 1
   1.2. Motivation ......................................................... 1
   1.3. Goals .............................................................. 2
   1.4. Thesis Structure .................................................. 3
   1.5. Definitions ....................................................... 3
       1.5.1. Tabletop .................................................... 3
       1.5.2. Data-Set and Project ..................................... 3

2. Background .......................................................... 5
   2.1. Tabletop Hardware ................................................ 5
       2.1.1. Overhead Projector Tabletops ............................... 5
       2.1.2. Rear-Projection Tabletops ................................ 6
   2.2. State of the Art in Existing Museum Tabletops ............... 8
       2.2.1. Dangerous Australians ..................................... 8
       2.2.2. Dialog Table ............................................... 10
       2.2.3. Churchill Lifeline ......................................... 10
       2.2.4. etx ......................................................... 11
       2.2.5. floating.numbers .......................................... 12
   2.3. Focus and Cruiser ............................................... 12
       2.3.1. Cruiser .................................................... 13
       2.3.2. Cruiser End-User Plug-Ins ................................. 13
       2.3.3. Focus ....................................................... 16
   2.4. JSR-296: Swing Application Framework ........................ 17
       2.4.1. The Swing Application Framework in Overview .......... 18
       2.4.2. Application Life Cycle ................................... 19
       2.4.3. Managing Resources ....................................... 20
       2.4.4. Actions ................................................... 22
       2.4.5. Tasks ...................................................... 24
       2.4.6. Persistent Session State .................................. 25
       2.4.7. Local Storage ............................................. 26
       2.4.8. The SingleFrameApplication Class ........................ 26

3. Curator .............................................................. 27
   3.1. Analysis .......................................................... 27
       3.1.1. Requirements Analysis .................................... 28
3.1.2. Quality Requirements ........................................ 28
3.1.3. Target Group of *Curator* ................................. 30
3.1.4. Low Fidelity Interface Prototypes ......................... 32
3.2. Software Design .................................................. 34
3.2.1. Selection of the Underlying Programming Language .... 34
3.2.2. The Workflow of *Curator* ................................ 34
3.2.3. Overall Structure ............................................ 37
3.2.4. Classes of the *spot* Package .............................. 38
3.2.5. User Interface Components ................................. 46
3.2.6. Other Classes of *Curator* ................................. 57
3.2.7. Other Features of *Curator* ............................... 63
3.2.8. Used Frameworks and Foreign Code ....................... 68

4. Evaluation of *Curator* ........................................... 69
4.1. Usability Evaluation ........................................... 69
4.1.1. Usability Testing with DECIDE ............................ 69
4.1.2. Pilot Study .................................................. 70
4.1.3. Usability Testing Session ................................. 71
4.1.4. Evaluation of Testing ...................................... 73
4.1.5. Achievements of the Usability Testing Sessions .......... 78
4.2. Time Evaluation ............................................... 80
4.2.1. Study description .......................................... 80
4.2.2. Results .................................................... 80

5. Conclusion .......................................................... 83
5.1. Summary & Future Work ....................................... 83
5.2. Conclusion ....................................................... 84

A. Images ................................................................. 87
A.1. Background ...................................................... 87
A.2. *Curator* .......................................................... 87

B. User Studies ......................................................... 91
B.1. User Tasks of Pilot Study .................................... 92
B.2. Participant Information Sheet ............................... 93
B.3. Participant Consent Form .................................... 95
B.4. User Tasks of the Main Study ............................... 96
B.5. Questionnaire .................................................. 98

C. Possible Extensions .................................................. 103
List of Figures .......................................................... 105
List of Tables ............................................................ 107
Listings ................................................................. 109
Nomenclature ........................................................... 111
1. Introduction

1.1. Problem Statement

Current tabletop technologies are often designed for only one purpose which makes it very difficult for curators to create a different collection of digital content for interactive tabletop displays. This thesis aims to address the problem by developing a design tool, Curator, for curating a virtual museum tour on a tabletop running on Cruiser with Focus, as well as a design environment that is quick and easy to use by people with basic IT skills.

1.2. Motivation

According to the International Council Of Museums (ICOM), purposes of museums are education, study and enjoyment [Ambrose, Paine, and (ICOM), 1993]. So museums aim to make knowledge and collections accessible to people of all ages and backgrounds.

As Brüninghaus-Knubel [2004] has shown, there are many ways a museum can meet the demands of education and study. Recently, museums have tended to offer computer terminals and free-standing PCs using specially designed software to add more pleasure to exploring the places of interest.

The use of tabletop displays in museums contributes to this purpose by additionally offering a potential for collaboration. Because of the horizontal orientation and bigger screens, visitors can explore the tabletop displays together. Some displays allow multiple input by multiple users which may improve this collaboration. Compared to vertical displays, users are less restricted by the limits of the human body as a horizontal display can be accessed from all sides, unlike vertical displays [Geller, 2006].

However, there is a lack of specialised systems which are designed to be used in museums. Systems are often built for only one situation and cannot be changed when, for instance, the exhibition is finished.

Cruiser, as a novel tabletop surface computing interface, offers the advantages mentioned above, is ready for any display type and supports multi-touch input. Because of its plugin-based design, Cruiser is highly flexible and customisable so that it is suited for use as an interface in museums. Focus, a plug-in for Cruiser, is an associative, similarity-based file system that allows to browse files by interest. In combination with Focus, Cruiser is a powerful interface for museums.
However, creating new data-sets for *Cruiser* applications is a significant effort for non-programmers. If the data-set for *Focus* must be changed, developer’s knowledge is needed. But not every composer of these data-sets is necessarily a programmer, so that it is very hard for them to create and change the data-set.

Hence, a design environment for creating data-sets for *Focus* is needed. So this thesis aim is to develop and describe a design environment for curating tabletop museum experiences.

### 1.3. Goals

The main goal of *Curator* is to simplify and quicken the creation of data-sets for *Focus*. *Focus* is described in detail in Section 2.3.

A *Focus* data-set comprises a background map or an image, the basis of a data-set. Images, video or audio files represent the content of the data-set. Finally, the coherence between map and content is stored in an XML file.

*Focus* interprets a touch on special marked areas on images as a search, retrieving If done so, *Focus* displays a number of files connected with this special area, which we call *spot*. It is also possible that another image with marked spots comes up. Typically, this is a more detailed image of the one the user clicked on. Consequently, it is possible to navigate from one image to another or to browse the contents of a spot.

However, it is tedious to create these data-sets manually. Therefore, pixels on the image at points of interest must be picked out first using a program that is able to do so. An XML file that can be interpreted by *Focus* must be created and all files which should be attached to a point of interest must be stored there and copied in a folder named after this spot. This process takes a lot of time, is not convenient, not suitable for people who are not familiar with XML files and is susceptible to mistakes.

*Curator* the application developed in this thesis, is a design environment for these data-sets and allows users to compose data-sets for *Focus* quickly and easily. It offers features to define spots on a chosen image and attach content to them. If the user is satisfied with his work, *Curator* writes all the data collected by the user to an XML file and copies all required files to a specified folder. It is also possible to load a data-set saved earlier, to revise it.

*Curator* makes it easier and faster to create data-sets for *Focus*. Consequently, it will be possible to spend less time on the actual creation process while improving the quality of data-sets and exhibitions.

In summary, the goals of this thesis are:

- to create an application that simplifies and hastens the creation of data-sets for *Focus*, and
- to evaluate its usability and effectiveness.
1.4. Thesis Structure

This thesis consists of two parts.

The first part of the thesis, Background, provides background knowledge. To start with, it introduces two common types of tabletop displays. Section 2.2 describes existing tabletops in museums as well as the Cruiser surface computing interface with its plug-in Focus. The next section describes the ‘Swing Application Framework’, the underlying framework used for the application developed in Chapter 3.

The second part comprises Chapter 3, Curator, and Chapter 4, Evaluation of Curator. Chapter 3 describes the implementation of Curator, a design environment for curating tabletop experiences. As a first step, the requirements of the application are defined. The Software Design Section of Chapter 3 gives insight into the implementation of Curator. Chapter 4 deals with user studies evaluating Curator.

1.5. Definitions

1.5.1. Tabletop

The expressions tabletop, tabletop display, and horizontal display are used equivalently in this thesis.

A tabletop in the context of this work is a system offering a horizontal image displayed by a computer on a surface as well as the possibility of direct manipulation of this image by the use of fingers or other pointing objects.

1.5.2. Data-Set and Project

In this thesis, the expressions data-set and project are used in the context of Focus and Curator.

A data-set, created with Curator and used by Focus comprises three components:

- A background image, which could be a map for instance. Points of interest (spots) are exported on this image.
- Folders named after the created spots, with all linked files or folders in it.
- An XML file that contains information about the spots as well as their linkages.

A project is a group of logically connected data-sets. Typically, data-sets of a project are stored in the same folder. For instance, it is possible to have a main map of inner Sydney, that is a data-set, and data-sets for each Circular Quay, Central Business District, and Kings Cross. There are markings (spots) in the inner Sydney data-set for each area Circular
Quay, Central Business District, and Kings Cross with links to their data-set, or rather their background image file. Together, all these data-sets form a project.
2. Background

2.1. Tabletop Hardware

This section introduces two common ways of realising tabletop displays. All systems described in Section 2.2, State of the Art in Existing Museum Tabletops, use one of the approaches described in this section. It provides the technical background for the next chapters.

Generally, there are many ways to build tabletop displays. Two common, low-cost ways, overhead projector tabletops and rear projection tabletops, are described below in detail. Input methods ready for multi-touch recognition can be found in Schöning, Brandl, Daiber, Echtler, Hilliges, Hook, Löchtefeld, Motamedi, Muller, Olivier, et al. [2008].

2.1.1. Overhead Projector Tabletops

Overhead projector tabletops are mainly composed of several components: one or more projectors, a surface to project on, and a (usually hidden) computer. Depending on the input processing method a camera is also needed.

Figure 2.1.: Overhead projector tabletop construction
2.1.2 Rear-Projection Tabletops

The construction is shown in Figure 2.1. Over a table or a surface, the ‘Image Surface’, the projector hangs from the ceiling. It is connected to a hidden computer. Depending on the chosen method, either vision technology (seeing), capacitance sensors (touch) or specialised systems based in hardware are used for input processing [Geller, 2006]. Using vision technology, a camera located next to the projector records all movement registered on the image surface. Capacitance sensors react on touch. In both cases, the input signals are transferred to the computer which processes and evaluates the signals and communicates the result to the tabletop application.

Table 2.1 shows some advantages and disadvantages of overhead projectors in combination with vision technology for input recognition. The biggest advantage is that every surface can be used as image surface. Thus, tables resistant to vandalism can be chosen as image surface. The issue that user’s shadows interfere with the tabletop experience does not cause substantial interface problems [Geller, 2006].

Table 2.1.: Construction-specific advantages and disadvantages of overhead projector tabletops with vision technology

<table>
<thead>
<tr>
<th>Advantages</th>
<th>Disadvantages</th>
</tr>
</thead>
<tbody>
<tr>
<td>Every non-transparent surface usable for image projection</td>
<td>No ‘all-in-one’ system</td>
</tr>
<tr>
<td>Scalable: simultaneous use of several projectors and cameras</td>
<td>Shadows on image surface</td>
</tr>
<tr>
<td>Image surface could offer tactile feedback</td>
<td>Static construction: fixed to specific location</td>
</tr>
<tr>
<td>Multitouch as well as object recognition possible</td>
<td>Vision technology is not as accurate as capacitance sensors</td>
</tr>
<tr>
<td>Image surface or table can be build to be durable against vandalism</td>
<td></td>
</tr>
</tbody>
</table>

2.1.2. Rear-Projection Tabletops

Rear-projection tabletop displays basically use the same technologies as Overhead Projector Tabletops. They are also composed of a projector, a projection surface, a computer, and, depending on the chosen input technology, a camera. This construction is similar to their overhead pendants, except that the whole system is able to be stored in a closed system and the image is projected against a special surface.

In contrast to overhead-projector tabletops, the projection surface of a rear-projection tabletop is less flexible: it must be semi-transparent as well as able to display a sharp image. To be durable against vandalism, the surface must also be resistant to scratches, pressure, environmental influences, etc.
The main benefit of this system is a ‘all-in-one’-look and feel for the museum visitors. When interacting with or pointing at the tabletop, no shadows of pointing devices will hide a part of the screen. It feels more like ‘magic’, the association with a computer is minimal. Therewith, the barrier to enjoying the device by people not familiar with computers can be lower.

Rear-projection is not limited to horizontal projection surfaces. As Han has shown, Frustrated Total Internal Reflection (FTIR), an infrared based technique to localise touches on a Plexiglas surface, can be used for realising multi-touch input in bigger surfaces which can also be used vertically.

Table 2.2 indicates some advantages and disadvantages of rear-projection tabletops. When creating a rear-projection tabletop for museums, it can be designed to match the theme of the whole exhibition. Since it is a closed system, there are more design possibilities than for an overhead projection tabletop. However, a special image projection surface which is highly resistant against scratches and is needed.

Table 2.2.: Construction-Specific advantages and disadvantages of rear-projection tabletops

<table>
<thead>
<tr>
<th>Advantages</th>
<th>Disadvantages</th>
</tr>
</thead>
<tbody>
<tr>
<td>Can be designed to conceal the use of a computer</td>
<td>Capacitance sensors must be invisible</td>
</tr>
<tr>
<td>Closed system</td>
<td>Vandalism measures needed</td>
</tr>
<tr>
<td>No shadows that reduce joy of use</td>
<td>Special image projection surface needed</td>
</tr>
<tr>
<td>Can be designed for portability</td>
<td>Scalability depends on structural shape</td>
</tr>
</tbody>
</table>
2.2. State of the Art in Existing Museum Tabletops

In 1972, the advantages of publicly accessible computer kiosks were demonstrated with the first electronic bulletin board [Colstad and Lipkin, 1975]. Newman and Wellner [1992] first suggested tabletop interfaces for use by an individual in 1992. Since then, public tabletop displays have also been introduced in museums.

Geller [2006] explored in 2006 several tabletop displays in museums. Regarding the museum experience, using a tabletop display has several advantages compared with a common computer with keyboard and mouse.

First, most museums are designed for general use [Ambrose et al., 1993]. That is why the benefits of interactive displays in museums should not be limited to people familiar in computer interaction. Tabletop displays help to reduce the barrier of handling a computer because they do not look like a computer and interaction with them does not feel like using an office computer with a conventional keyboard and mouse controls.

Second, horizontal displays offer a more-familiar, collaborative environment [Geller, 2006]. Most of the displays are designed for use from several angles, allowing more people to interact simultaneously if this is supported by hardware. This is also in strong contrast to common desktop computers which are generally used by only one user at the same time. Compared to vertical displays, horizontal ones offer a larger interaction area because visitors can gather around the table and not just in front of them.

This section introduces various existing tabletop systems in museums. It provides the background for Chapter 3 and gives an insight into other existing tabletop displays in a museum environment. It is recommended to read the Section Tabletop Hardware on 5 first because it contains basic knowledge of tabletop hardware important to this section.

2.2.1. Dangerous Australians

*Dangerous Australians* is part of the permanent exhibition ‘Surviving Australia’, displayed in the Australian Museum in Sydney, Australia. The six meter long table enables the visitors of the Australian Museum to explore Australia’s deadliest top ten animals.
On the screen which is divided into water and ground areas, lethal Australian animals like the funnel web spider, saltwater crocodiles, brown snake and box jelly fish move around and try to attract the visitor’s attention. They allow him to touch it and read information on this animal. The animals also interact with people: for instance, a shark attacks someone who keeps his hands nearby for too long on the table. The application then provides information how to help people wounded or poisoned by the attacker.

Technically, this exhibit uses several projectors from above (see Section 2.1.1) with cameras and infrared light. It is able to accept input from multiple visitors. The table itself is solid.
However, it reacts quickly when hovering a bit over the image surface. The software is developed for this purpose only and runs on Mac OS X [lightwell, 2008].

A movie of *Dangerous Australians* can be found on the enclosed disc.

### 2.2.2. Dialog Table

The Dialog Table is a multi-touch table designed for the Walker Art Center in Minneapolis. The table’s aim is to encourage visitors to engage with and discuss art.

Dialog Table is part of the permanent exhibit. It is designed as a rear-projection device: Using a system of mirrors the images are projected from the inside of the table onto the back of two semi-transparent displays (see Figure 2.6).

A video showing visitors interacting with the Dialog Table in the Walker Art Center can be found on the enclosed disc.

### 2.2.3. Churchill Lifeline

The Churchill Lifeline (see Geller [2006] and Figure 2.7) is, like the Dialog Table, a permanent exhibit. It is located at the Churchill Museum and Cabinet War Rooms in London. Created by Small Design Firm, Churchill Lifeline is a 15 meter long table that allows up to 26 people at the same time to browse over 6000 letters, documents and photographs related to Winston Churchill’s life.

Thirteen projectors beam the documents in chronological order from above onto the table. Touchstripes on both long sides allow visitors to select a single event they want to see more information about. The image surface is not directly interactive.
2.2.4. etx

etx, located in the Indianapolis Museum of Art, in contrast to Dialog Table and Churchill Lifeline, operates not by direct input but by reflective-taped wooden paddles tracked by infrared cameras. Up to three visitors can use three paddles with different effects to the table together:
2.2.5 floating.numbers

- ‘?’-paddle displays further details,
- ‘Arrow’-paddle shows the physical location of the exhibit in the building and
- ‘X’-paddle shows connections with other exhibits in the collection.

Figure 2.8.: etx, Indianapolis Museum of Art (© Courtesy Indianapolis Museum of Art, Pervasive Technology Labs at Indiana University)

Floating.numbers

Floating.numbers is a tabletop display shown in the Jewish Museum in Berlin. As part of a temporary exhibition on this nine by two meter table, a continuous stream of digits flows from one short end to the other. As shown in Figure 2.9, visitors can touch numbers floating near them to get further insight into the meaning or story of the chosen number.

Like in Churchill Lifeline, the display of floating.numbers is projected from above. Because of the capacitance sensors the table uses, multiple users can interact with the table simultaneously. Unlike Churchill Lifeline, floating.numbers has its sensors on the whole table so that there is no limited interaction area (see Geller [2006]).

2.3. Focus and Cruiser

Curator is a design environment for curating tabletop museum experiences. It helps to produce data-sets for the Focus system, a plugin for Cruiser. This section introduces the reader to Focus and Cruiser with the aim to get familiar with their principles and behaviours and prepares him for the following chapters.
2.3.1. Cruiser

*Cruiser* is a multi-user, gestural, collaborative, plugin based software framework for tabletop displays developed by Trent Apted, University of Sydney [Apted, Kay, Assad, and Sydney, 2006].

*Cruiser* has gestures to select, move, rotate and resize, called *rosize*¹, objects such as photographs. The framework also allows the user to ‘flip’ objects. If a photograph has been flipped, it is possible to attach other photographs to it simply by moving over the flipped one and releasing it. Objects can also be ‘flicked’: this is done when releasing an object whilst it is moving quickly. This is a convenient method to move objects over a long distance on a big table.

The current version of *Cruiser* supports many plugins that extend the functionality of the framework. Below, a few of the end-user plugins and functionalities are described because they illustrate the functioning of *Cruiser* and help to understand *Curator*.

2.3.2. Cruiser End-User Plug-Ins

**Black Hole** The *Black Hole* is an object that provides similar functionality to trash cans available in several operating systems. However, it has a ‘sphere of influence’ that affects the

---

¹ ‘rosize’: Rotating and resizing at the same time in one gesture. A rosize gesture is performed in the *Focus* video on the disc of this thesis.
size of objects near to it. Moving it closer to the middle of the hole, the photo gets smaller until it is hidden. Figure 2.10 illustrates this process.

When the photo is flicked near the hole, it will be ‘caught’ by the gravity of the Black Hole and finally get ‘sucked’ into it. To get objects out of it, users can reduce the size of the Black Hole. Dwelling on it permanently deletes the content.

![Black Hole sucking in an image](image)

**Frame** The Frame is a semi-transparent object which allows users to capture an image of the objects covered by the Frame. The new image will appear on the table.

Figure 2.11 shows how to create a new image using the Frame. The size of the Frame can be adjusted by dragging the borders of it. Like other objects, a Frame can be resized, moved and flicked. Dwelling on the frame captures the new image and brings it up on the screen.
Storage Bin  The Storage Bin is an object that supports grouping of other objects in it. It arranges the content in a grid. Storage Bins can contain other Storage Bins, similar to a folder structure in normal file systems. Of course it is possible to drag an object out of a Storage Bin again. Figure 2.12 shows a Storage Bin with three pictures in it.

Figure 2.11.: Frame creating a screen shot

Copier  A Copier works similarly to a real-world photocopier. When moving an object over it, the Copier stores a copy of this object in it. This copy can be dragged out of the copier as often as desired, unlike the real-world complement.

Figure 2.12.: Storage Bin
2.3.3 Focus

Focus, developed as a plugin for Cruiser by Anthony Collins, University of Sydney, is a novel tabletop interface for collaboratively accessing multiple remote file systems [Collins, 2008]. Focus is an associative, similarity-based file system that enables browsing files by interest. The underlying system retrieves automatically files across multiple platforms and presents only relevant files.

When dwelling on a document, Focus searches for similar files based on tags, file and folder names and time stamps, and shows them to the user. A history bar, shown at the top border of Figure 2.14, lists all the items a search was done for so that the user can understand why the items shown by the system are on the table. Also he can dwell on these items in the list to go back to a previous search.

With its extended functionality, Focus offers the end-user to click on defined positions on images or maps. These special positions, called ‘spots’ have two functions if the user dwells on it: First, they can open another map with spots, for instance to offer a closer look to an area or to navigate in a direction. Second, additional files or URLs can be attached to a spot. In this case, dwelling on this special position raises attached documents. In Figure 2.14, these spots are marked with a red circle.

2 When a spot is formatted in typewriter (Spot), the class Spot.java is named. However, a spot describes the position marked on the image by the Spot class.
For instance, the system offers a map of inner Sydney with three special positions in Sydney’s Central Business District, The Rocks and Kings Cross. Dwelling on the The Rocks spot opens another map which is closer and offers more details of the area. This second map has two spots, one next to The Rocks and one at the Sydney Opera House. Dwelling again on one of these spots, for instance the Opera House spot, loads attached images, video and audio files.

A video of Focus can be found on the disc of this thesis.

![Figure 2.14: Focus with Sydney data-set](image)

This is only one example of Focus. Uncountable data-sets and possibilities are offered with this plugin. Conceivably, it could also offer a data-set which offers a tour through the human body. Semi-transparent and X-ray images could give good insight into the inner points of a human being. Dwelling on organs could provide information on functioning, showing states of diseased organs, and many more.

### 2.4. JSR-296: Swing Application Framework

This section ‘JSR-296: Swing Application Framework’ introduces to the powerful ‘Swing Application Framework’, the chosen framework for the implementation of Curator. The chapter Curator, which deals with the implementation of the application, presupposes the knowledge procured with this section.
Curator is based on the Swing Application Framework. This is why it uses most of the features SAF comes with. This section illustrates the functionality used in Curator, such as Tasks, Application Life Cycle and Resource Management, to introduce the reader into these topics and build up background knowledge which is important for the following chapters.

2.4.1. The Swing Application Framework in Overview

With Swing, Java offers an Application Programming Interface (API) with a set of customisable graphical components the look and feel of which can be changed at runtime. Sun Microsystems, the creator, calls it a next-generation GUI toolkit [Loy and Eckstein, 2002].

The Swing Application Framework [SAF] was developed to simplify building applications in Java using Swing. Although a lot of Swing applications have the same core elements like resource, action and session state management as well as startup and shutdown, developers have to do this implementation on their own.

The Java Specification Request (JSR) 296’s aim from May/June 2006 is to provide these developers with a simple API:

‘This JSR will provide a simple application framework for Swing applications. It will define an infrastructure which is common to most desktop applications. In doing so, Swing applications will be easier to create.’ [JSR].

![JSR-296 architecture](©java.net)

JSR 296 will be part of the next Java version, Java SE7. The integrated development environment (IDE) NetBeans since version 6.1 has already integrated the prototype implementation of JSR 296.
According to O’Connor [2007], this prototype implementation of the Swing Application Framework provides basic functionality for

- Application lifecycle, notably GUI startup and shutdown.
- Support for managing and loading resources, like strings, formatted messages, images, colors, fonts, and other types common to desktop application
- Support for defining, managing, and binding actions, including actions that run asynchronously (in the ‘background’).
- Persistent session state: support for automatically and selectively saving a GUI state from one run of an application to the next.

The functionality of Swing Application Framework is described in detail in the following sections. It helps to understand implementation details of Curator.

### 2.4.2. Application Life Cycle

Swing Application Framework (SAF) offers several life cycle methods to support applications. These methods are (in order):

- **launch**: Must be called to launch the application. The application will be launched in the Event Dispatch Thread (EDT). It also starts the application’s life cycle and will call initialize, startup, and ready.

- **initialize**: SAF calls this method. This method is used to process any initial configuration not related to the user interface (UI), but needs to be done before displaying the UI. An example would be to check a database connection or to access system properties.

- **startup**: SAF calls this method. This method should create and display the user interface.

- **ready**: SAF calls this method. Functionality based on a created and visible UI can be performed in this method.

- **exit**: Must be called to invoke the application’s exit functionality.

- **shutdown**: SAF calls this method. Clean up like closing data connections or saving files can be done here. Further details on exiting an application are described below.

Many programs contain complex functionality. Thus it is important, for example, to ask the user if he wants to save unsaved documents or close open transfers. JSR 296 offers application shutdown functionality to do so.

To refer to this functionality, the developer can add an ExitListener to the application. An ExitListener has to provide two methods, public boolean canExit(EventHandler eo) and public void willExit(EventHandler eo). First, the application will call the canExit
method of each ExitListener. In this method, the ExitListener should clarify if it is ready to exit. If so, it should return true, false otherwise.

If every ExitListener is ready, the application will call each ExitListeners willExit method, where, for instance, an opened document can be saved before the program will exit.

### 2.4.3. Managing Resources

The Swing Application Framework also offers functionality for resource management. Resource management is an easy way to publish applications in multiple languages and country settings. It is possible to create several properties files for each language the application should support. For instance, a JLabel can profit of this technology by predefining a properties file with its text. To change and improve usability, country-specific colours can be described in a properties file for a country, too. With resource management it is also possible to access, for example, images.

Resources can be accessed in three ways instead of accessing them directly. Using the ResourceManager and ResourceMap helps managing resources. ResourceMaps include, besides the resources, a link to the parent chain of ResourceMaps including a Map for the class file, the application subclass, and all subclasses up to the Application class. The ResourceManager creates such Maps and can be accessed, like Maps, using the ApplicationContext.

<table>
<thead>
<tr>
<th>Key</th>
<th>default language</th>
<th>de_DE = Deutsch (Deutschland)</th>
</tr>
</thead>
<tbody>
<tr>
<td>xmlFilePanel.border.title</td>
<td>Name of XML-File</td>
<td>XML-Datei</td>
</tr>
<tr>
<td>folderPanel.border.title</td>
<td>Target Folder</td>
<td>Zielordner</td>
</tr>
<tr>
<td>Filename</td>
<td>XML file &quot;&quot;</td>
<td>Die XML-Datei &quot;&quot;</td>
</tr>
<tr>
<td>Filename2</td>
<td>&quot; does not exist yet, do... &quot;</td>
<td>&quot; existiert noch nicht,...</td>
</tr>
<tr>
<td>createFile</td>
<td>Create File</td>
<td>XML-Datei erstellen?</td>
</tr>
<tr>
<td>errorFolder</td>
<td>Folder could not be created... Der Ordner konnte nicht er...</td>
<td></td>
</tr>
<tr>
<td>errorFile</td>
<td>File could not be created... Die Datei konnte nicht er...</td>
<td></td>
</tr>
<tr>
<td>Foldername</td>
<td>Folder &quot;&quot;</td>
<td>Der Ordner &quot;&quot;</td>
</tr>
<tr>
<td>Foldername2</td>
<td>&quot; does not exist yet, do... &quot;</td>
<td>&quot; existiert noch nicht,...</td>
</tr>
<tr>
<td>createFolder</td>
<td>Create Folder?</td>
<td>Ordner erstellen?</td>
</tr>
</tbody>
</table>

Figure 2.17.: Example of a ResourceBundle with default language (English) and localised language (German)
2.4.3.1. Manual Management

Using the ResourceMap (a read-only generalisation of ResourceBundles) and the ResourceManager obtained with the ApplicationContext, resources such as menu entries can be read from a properties file. Depending on the country and language of the underlying operating system the matching properties file will be chosen. For instance, if the Java Virtual Machine (JVM) runs on a German computer, the properties file with 'de.DE' in its name will be loaded. If no properties file for the underlying operation system's country and language is found, the default properties file will be chosen.

Listing 2.1 shows how to read a String from a properties file.

```java
ResourceMap resourceMap = org.jdesktop.application.Application.getInstance(CuratorApp.class).getContext().getResourceMap(CuratorView.class);
infoMenuItem.setText(resourceMap.getString("showSpotsInfoMenuItem.text"));
```

Listing 2.1: Manual access to a properties file using a ResourceMap

2.4.3.2. Component Resource Injection

Besides the possibility to manually access resources, SAF also offers resource injection to components. Resource injection means that the framework initialises values of components instead of components itself. With this method, Swing Application Framework will inject resources throughout the whole container.

To be able to do so, the name of the component must be set using the component’s setName(String name) method. In the ResourceBundle, the properties of the component must be named as follows: [component name].[property name]. For example, the text of a JLabel named ‘descriptionLabel’ must be stored as ‘descriptionLabel.text’. Also, descriptionLabel.setText("descriptionLabel") must be invoked so that the Swing Application Framework can inject correctly.
2.4.4 Actions

Table 2.3: Relationships among class file, `ResourceBundle` name, and `ResourceBundle` file name

<table>
<thead>
<tr>
<th>Class</th>
<th>ResourceBundle Name</th>
<th>ResourceBundle File Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>options.Options</td>
<td>options.resources.Options</td>
<td>options/resources/Options.properties</td>
</tr>
<tr>
<td>arbeit.Spot</td>
<td>arbeit.resources.spot</td>
<td>arbeit/resources/spot.properties</td>
</tr>
</tbody>
</table>

2.4.3.3. Field Resource Injection

Resources can also be injected to fields using the `@Resource` annotation. When invoking the `injectFields` method Swing Application Framework will automatically inject resources to any field marked with this annotation. This method will be invoked using a `ResourceMap` object. Unlike the component resource injection, field resource injection will not traverse hierarchies.

Naming conventions must be respected, too. If a resource with the name `myString` should be injected in the class `myClass`, the corresponding entry in the `ResourceBundle` should be `myClass.myString`.

```java
public class MyLabel extends JLabel{
    @Resource
    String text;

    public MyLabel(ApplicationContext ctxt) {
        initComponents();
        ResourceMap resource = ctxt.getResourceMap(MyLabel.class);
        resource.injectFields(this);
    }
}
```

Listing 2.2: Example of a Field Resource Injection

2.4.4. Actions

Swing Application Framework supports Java application developers in creating actions. It sometimes happens in programs that the same action should be performed by, for example, either a mnemonic, a button or a menu entry. To help developers handle these actions, JSR 296 introduces the `@Action` annotation.
Figure 2.19: Renaming a point of interest by double-clicking either the Spot itself or its entry in the list

If this annotation marks a function or method, it is registered in the ActionMap in the ApplicationContext under the name of the function or method. Also, it is possible to provide a different name if desired. Listing 2.3 shows an example of how to use the @Action annotation.

```java
// ...
public void init(ApplicationContext ctxt){
    ResourceMap resource = ctxt.getResourceMap(ResizeFontPanel.class);
    resource.injectComponents(this); // not needed for actions
    ActionMap map = ctxt.getActionMap(this);
    helpMenuItem.setAction(map.get("showHelp"));
    helpButton.setAction(map.get("showHelp"));
}
// ...

@Action
public void showHelp(){
    // ...
}
```

Listing 2.3: Example: Using the @Action Annotation for Associating Actions

First, a method to associate an action is needed. In Listing 2.3, showHelp() is the function to be called when the corresponding action fires.
Second, the framework requires to associate the graphical user interface (GUI) elements with the action. Therefore, it is necessary to get the ActionMap from the ApplicationContext (line 5 in Listing 2.3).

To associate a GUI element with the desired action, its setAction method has to be invoked with the action as argument. The ActionMap instance provides this action by calling its get("action name") method. Altogether, associating an action with an GUI element looks like this: guiElement.setAction(actionMap.get("actionName"));

2.4.5. Tasks

To prevent an expensive task from blocking the Event Dispatch Thread and with it the GUI, such large tasks, like copying files, should run in their own thread. For such cases, Sun provides the SwingWorker class. Swing Application Framework extends this class and supports the developer in executing tasks and monitoring their state.

Overriding the Task class of the SAF is a convenient method to create a Task. By overriding the doInbackground() method, the desired expensive functionality can be provided. For instance, there is also a succeed method which is called, as the name suggests, in case of success. In case of failure, the failed(Throwable t) method is called. In any case, finished will be invoked. It will run on EDT, so it can be used to update GUI elements.

When the developer provides a method which he marks with the @Action annotation and returns a Task, the framework will automatically run this Task.

Also, the Task class provides functionality to inform interested classes on properties like the task progress. So the programmer can use the method setProgress(float/int val) or setProgress(float/int val, float/int min, float/int max) to define the current progress of the Task. The latter method is very convenient because value does not need to be normalised, only the borders have to be provided. A second method to indicate the state of the progress is the setMessage(String message) method. It allows the developer to tell all interested parts of the program what the progress is doing at the moment. For instance, in a file copy Task he can inform which file is currently moving.

When extending the SingleFrameApplication class (see Subsection 2.4.8) the Swing Application Framework offers an easy way to show a Task’s message and progress. Because the SingleFrameApplication implements a JLabel to inform of messages, a JProgressBar to show the progress, and a second JLabel to show that a progress is in action, it interacts very well with the methods mentioned above. When starting a Task using the execute(Task t) method of the ApplicationContexts TaskService, SAF will automatically listen to the Task’s information and update the GUI accordingly. As per [Cooper, Reimann, and Cronin, 2007], this is a good way to inform advanced users without interrupting them.
2.4.6. Persistent Session State

The Swing Application Framework offers the possibility to save and restore the session state, the ‘graphical window configuration of the application’ [O’Connor, 2007]. This means that the window size, window position, selected tabs, internal frames, column widths and other attributes are in the same state when starting the program as they were when it was exited.

When subclassing the `SingleFrameApplication` class (see Subsection 2.4.8), session storage is implemented by the framework. No further work needs to be done by the developer. So it
is convenient for the user to find the application at the position and in the state he or she left it.

However, when subclassing the `Application` class, the session state must be saved and restored manually by the developer. This can be done in the application’s life cycle methods, see Section 2.4.2. The framework’s using guideline [O’Connor, 2007] suggests to restore the state in the `startup` method and store the state in `shutdown`. The methods `restore(java.awt.Component root, String fileName)` and `save(java.awt.Component root, String fileName)`, to be found in the `SessionStorage` class, will actually save and restore the session state.

### 2.4.7. Local Storage

The `LocalStorage` class is used by the `SessionStorage` class to persist its data. Internally, this class uses the `XMLEncoder` class or the `XMLDecoder` class to store data in an XML file and to restore them from it. The `ApplicationContext` provides this class (`getLocalStorage()`), so that programmers can also use the `LocalStorage` class to persist other data than those from `SessionStorage`.

### 2.4.8. The SingleFrameApplication Class

Subclassing the `SingleFrameApplication` class offers a lot of advantages. A `JFrame` is provided in the main window and overrides some life cycle methods with default behaviour. The `SingleFrameApplication` also injects resources and implements a rudimentary `WindowAdapter`. This listener contains an `ExitListener` which calls the `exit` method. Subclassing this class adds basic functionality to saving and restoring sessions (see Subsection 2.4.6).

---

3 To simplify the reading, the male form is used
3. Curator

This chapter describes Curator, the application developed in this thesis, in detail. It starts with the analysis phase in Section 3.1 which includes the requirements of the developed program, a description of the target user group, and low fidelity interface prototypes. The second section, Software Design, illustrates the architecture and program structure.

3.1. Analysis

The analysis phase forms the basis of the entire software project. With the information researched in this section, the design of the software can be outlined. This section identifies the requirements of Curator and defines quality requirements. Possible end-users of the program are described as well. Interface paper prototypes illustrate how Curator could look like later.

All requirements, functional as well as non-functional, named in this section were found by interviewing the developer of Focus and other students, who created the data-sets before. For a better overview of the process, they have been surveyed when creating a data-set. User

Figure 3.1.: User story cards helped to formulate requirements
stories, shown in Figure 3.1, helped to formulate the requirements, too. With data-set files as a reference, the notes from the survey and the user story cards, the requirements in this section could be formulated.

3.1.1. Requirements Analysis

The requirements analysis identified the following functional requirements for Curator. Simultaneously, it provided an overview of the needed, desired and nice-to-have functionality of the application. The use cases on page 38 show an overview of the final functionality implemented in Curator.

Table 3.1.: Functionality requirements

<table>
<thead>
<tr>
<th>Requirement</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>/R1/</td>
<td>Mark spots on a map or on an image</td>
</tr>
<tr>
<td>/R2/</td>
<td>List spots in a separate XML file</td>
</tr>
<tr>
<td>/R3/</td>
<td>Attach files and URLs to a spot</td>
</tr>
<tr>
<td>/R4/</td>
<td>Change the map or image to mark spots on</td>
</tr>
<tr>
<td>/R5/</td>
<td>Copy file attachments to a desired place</td>
</tr>
<tr>
<td>/R6/</td>
<td>Provide information on a selected spot</td>
</tr>
<tr>
<td>/R7/</td>
<td>Move and delete spots</td>
</tr>
<tr>
<td>/R8/</td>
<td>Delete attachments</td>
</tr>
<tr>
<td>/R9/</td>
<td>Guide first time users</td>
</tr>
<tr>
<td>/R10/</td>
<td>Add tags to file attachments</td>
</tr>
</tbody>
</table>

Requirements /R1/ and /R2/ are needed for the success of Curator. If these are not fulfilled, the use of Curator is impossible. Because the rest of the requirements is highly dependent on /R1/, /R1/ is absolutely necessary. Without /R2/, a further use of Curator is impossible.

Requirements /R3/ as well as /R4/ extend the functionality of Curator.

Requirements /R5/ to /R10/ improve the comfort and usability of Curator. They are desirable but not inevitably needed to ensure the usage of the application.

3.1.2. Quality Requirements

To ensure a focus on the right aspects during the software development phase, the quality requirements shown in Table 3.2 were set. These requirements should help the developer and author of this thesis to organise his time.

A special focus on usability as well as efficiency is required to meet the goal to develop a program which is quick and easy to use. The portability and flexibility aspects are very important, too. The operating system of the end-users is not known yet and moreover, a development for a single operating system would hamper a widespread use of Curator and
with it Focus. With an eye on the maintainability aspect during development, the way is paved for further improvements and extensions by succeeding students.

Reliability and correctness of output are also important so that Focus can interpret them correctly. A development with regard to extensibility makes it easier to implement new features, for instance the ability to use different spots.

Since Curator is no application which is permanently running, average memory management is required. Moreover, no expensive tasks are expected in the program; so programming for an average algorithm performance is adequate.

Regarding the appearance requirement of Curator, consistency with other programs is desirable. An appearance and behaviour matching the user’s operating system and other native-looking applications makes the program easy to learn by the user and helps to feel comfortable with it.

The security aspect is not relevant to this thesis. Curator will be implemented as an offline application and no security-sensitive data such as passwords or user details will be accessed. Concerning the use of sensible files, the user is in charge of the publishing of such files for curating museum experiences.

<table>
<thead>
<tr>
<th>Requirement</th>
<th>very important</th>
<th>important</th>
<th>average</th>
<th>not relevant</th>
</tr>
</thead>
<tbody>
<tr>
<td>Reliability</td>
<td></td>
<td>X</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Usability</td>
<td>X</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Appearance</td>
<td></td>
<td>X</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Efficiency</td>
<td></td>
<td></td>
<td>X</td>
<td></td>
</tr>
<tr>
<td>Portability and Flexibility</td>
<td></td>
<td>X</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Maintainability</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Correctness of Output</td>
<td></td>
<td>X</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Security</td>
<td></td>
<td></td>
<td></td>
<td>X</td>
</tr>
<tr>
<td>Extensibility</td>
<td></td>
<td>X</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Performance</td>
<td></td>
<td></td>
<td></td>
<td>X</td>
</tr>
</tbody>
</table>

Since the development of Curator covers all phases of a software project in a short period of time, this chapter Curator as well as the creation of Curator itself focus on the implementation phase. Due to the number of functional requirements and the weighting of the quality requirements, testing has been conducted only on Mac OS X, the operating system developed on. However, this does not affect the quality requirement ‘Portability and Flexibility’, since it should be portable to other systems (see Section 3.2.1).

Also, user studies of Curator were rescheduled to the end of the implementation phase. Due to the lack of time it was not possible to develop interface prototypes of the application, do user studies of these prototypes, evaluate the data and build up the interface based on this
data. However, several low fidelity interface prototypes have been created and assessed (see Section 3.1.4).

3.1.3. Target Group of Curator

Up to now, data-sets for Focus have been created manually either by the developer of Focus or by other students.

Since there is neither a prior tool like Curator nor Focus is introduced in museums yet, it is difficult to define a specific target group. It is conceivable that working students or part-time employees may use Curator as a main target group. It may also be possible that the exhibition designer himself wants to create data-sets using the application developed in this thesis. In any case it is most likely that they will have basic computer skills but no programming skills. Basic computer knowledge assumes regular use of computers including file managers, office suites, and e-mail programs. The target operating system the end user works with is not yet known. It is to be expected and advisable that the users of Curator are given the opportunity to see Focus so that they have a better insight to the system. This insight would be increased if the target group could use Focus.

Fictive persons who represent the target group, so called Personas, help to empathise with the end users throughout the entire software engineering. Figures 3.2 and 3.3 represent the primary and secondary personae of Curator.
Primary Persona

David, 23, is a first year archaeology student at the University of Sydney, Australia. Since he started studying half a year ago, he has been permanently out of money. This is why he successfully applied for a job as a working student at the Australian Museum. Apart from the fact that the museum is in walking distance from his home, he is happy to gather some study-related experience and good payment.

He is now responsible for creating and updating the labels of the exhibits, for preparing new exhibitions of his boss, an exhibition designer, and for managing the new tabletop system his department bought two months ago. Since he is not only interested in ancient artifacts but also in innovative electronic devices such as his Nintendo Wii, David particularly enjoys interacting with the new table.

However, he is disappointed by the fact that it is hard to update the tabletop system with new data. Without Frank, his predecessor, who was willing to introduce him, David would never have found out how to remove the old exhibition from the table and set up the new data for the kid’s floor exhibition ‘The Human Body’. He is pleased with his Apple MacBook and the fact that he is allowed to use it for editing the labels for the exhibition, but he does not want to go further and start programming or the like. Since he is paid per assignment and not per hour, he wished there was a less time-consuming way to set up the new exhibition.

Secondary Persona

Alice, 39, has been working in the Australian Museum in Sydney, Australia, for thirteen years now. Two years ago, she was promoted to the chief exhibition designer. Amongst her colleagues she is known as a creative and innovative person who is always looking for ideas to pep up her exhibitions.

When she heard about the tabletop display system of the University of Sydney, Alice was excited about its opportunities and possibilities to be used in their exhibitions and contacted the university and ordered a device. The first exhibition with the tabletop display was a great success and she received acknowledgement from all her colleagues.

At the moment, her IT working student Frank is managing the complex set-up of the tabletop with the data she passes to him. But he will quit soon and be replaced by another non-IT student. Fortunately, Frank will instruct him. However, Alice misses the possibility to configure the data presented on the tabletop in case of emergency by herself - even if she has only little time to do so.
3.1.4. Low Fidelity Interface Prototypes

This section shows paper interface prototypes of Curator which explore the interface design outline.

Figure 3.4 shows the first developed prototype. It has a prominent area where the image or map with spots is displayed. On the top left, information of the selected spot are displayed. Below this panel, all attachments of this spot are listed. An instrument bar at the bottom allows to drag new spots to the map. This interface lacks a button to delete a spot as well as an overview list of spots.

![First Curator paper prototype](image)

Figure 3.4.: First Curator paper prototype

The second prototype, shown in Figure 3.5, implemented a button to delete selected spots to make this frequently used task more accessible than a menu entry. Also, all created spots are now listed, so that it is easy to select multiple entries of the list to link attachments to all of them. To guarantee that all information is in sight, a three column design has been chosen (map, spot list and information, attachment list). The instrument bar has moved to the top of the window as it is more natural to drag from top to bottom than from bottom to top.
The third paper prototype shows small changes. To provide more space to the image area, the information panel and the linkage list are now in a tabbed pane. To allow the user to customise the layout of Curator, the right column is detachable and can be positioned everywhere on the desktop. This prototype is illustrated in Figure 3.6.

Since the third prototype offers more space for the background image it is also suited for smaller screens. Even though spot list, attachment list and information panel are combined in one bar, the layout feels cleaner compared to the second prototype. Besides, an attachment list as high as the program is needed only in few cases so that it’s size can be reduced. Hence, the third prototype is chosen for the implementation.
3.2. Software Design

This section describes the architecture and program structure of Curator as well as its components in detail. It starts with a short justification why the programming language Java was chosen to create Curator. Then this section gives an introduction to the overall structure of the program and depicts all parts of it, UI components, the Spot package, other components, other features, and applied frameworks in detail.

3.2.1. Selection of the Underlying Programming Language

The quality requirements (see Section 3.1.2 on page 28) showed that portability and flexibility are very important for the program. This section explains why Java is the language chosen for Curator.

Java programs run in their own so-called Virtual Machine. This Java Virtual Machine (VM or JVM) is responsible for the execution of the byte code so that the code written by the programmer only has to be compiled for this Virtual Machine. The Virtual Machine is implemented for many operating systems (OS) so that the program can (mostly) be run independently of the OS. This makes it very easy for the developer to create applications for more than one OS. Java also offers a implementation depending look and feel for several operating system which is similar and consistent to native applications. Because the OS of the end user of Curator is not known yet, the advantage of Java’s Virtual Machine compared to other programming languages is huge.

Java applications can also be started from the internet, using Java Web Start [JWS]. So the user can start the application on any computer, as long as Java SE6 is installed on this machine. However, security requirements must be met.

Java provides a lot of built-in frameworks which reduce the developer’s workload. Also, as described in the first part of this thesis, see Section 2.4 on page 17, using Java forms a sample application of JSR 296 at the same time.

These are the four main reasons which led to Java as the programming language for Curator.

3.2.2. The Workflow of Curator

Curator is designed to simplify and quicken the curating process of museum tabletop experiences.

To help unexperienced users to work with Curator, the application offers a workflow to create data-sets for Focus. Figure 3.7 shows the workflow of Curator, Figure A.3 on page 89 shows the full activity diagram of the main creating workflow. A video of Curator in action can be found on the disk of this thesis.
Figure 3.7.: Workflow of Curator
Curator’s workflow starts, once the program is loaded, with loading an image, the base of each Curator and Focus data-set. As shown in image #1 of Figure 3.7, the application provides a menu entry to open an image. Advanced users may also use mnemonics (Command + O for Mac OS X, Ctrl + O for other systems) for a faster access.

#2 in Figure 3.7 shows the dialog to choose an image file as background image for the data-set. Further input from the user is required: the image must be chosen using a dialogue; Figure 3.26 on page 68 shows more details.

After the image, on which all spots will be displayed, is loaded, the user can start creating Spots. Image #3 shows how to do this. By dragging the blue spot icon from the top left to the desired place on the image area of the main window, a spot will be created with its tip at the user’s mouse cursor.

The workflow now intends to link attachments to this spot. Image #4 of Figure 3.7 shows a picture which is dragged from Mac OS X’s file manager ‘Finder’ onto the sidebar of Curator. It is also possible to drag other data types, multiple files or a URL string into the application. All currently selected spots will be linked to these attachments.

In a next step, the spot can be renamed to help the user identify it. Image #5 of Figure 3.7 illustrates a double-click on the spot itself and the dialogue box which asks the user to enter a new name for the chosen spot. This dialogue box could also be opened by double-clicking the spot in the list in the sidebar. Even without renaming the spot, a data-set can be used because the system gives default names to the spots.

Now the user is finished with a spot and can either create a new one or save the work and quit the program.

The procedure described above is the workflow suggested by Curator. However, the user is not bound to this sequence. Renaming a spot may be done first instead of linking attachments, once a spot is created. It depends on the user’s preferences what he likes better doing first. The only reason the user guidance system suggests this sequence is that attachments are more important to the program than the name is. The user study will help to identify the order of events the users prefer. If the usability study shows that users like to rename spots first, the workflow and the guidance system should be changed accordingly.

More advanced users may also create all desired spots first and then link attachments to multiple spots. This may considerably save time.

Figure 3.8 in Section 3.8 gives an overview of all major Use Cases of Curator. They serve as a reference for the volume of the program.
3.2.3. Overall Structure

This section provides an outline of the architecture and program structure of the Curator application.

Curator mainly comprises four parts:

- **Spot** components, specified in Section 3.2.4 on page 38,
- User interface components, see Section 3.2.5 on page 46,
- other components like the main class and helper classes and methods, see Section 3.2.6 on page 57, and
- existing frameworks used in Curator, see Section 3.2.8 on page 68.

These four parts assure a proper division in user interface related classes, classes that are related to Spots and classes needed otherwise in the program. Section 3.2.8 provides a short overview of the frameworks used in Curator.

Apart from these parts, Section Other Features of Curator explains features of Curator that are not covered in the sections mentioned above.

To give an overview of the functionality of Curator, Figure 3.8 shows the Use Cases.
3.2.4 Classes of the spot Package

This section deals with the package `bachelorarbeit.spot`. This important package comprises ten POJOs (Plain Old Java Object), interfaces and enums. Figure 3.9 and Table 3.3 show the contents of the package.

Table 3.3 is ordered alphabetically in contrast to the descriptions in the following sections, which are organised in a different order to ease the understanding while reading.
3.2.4.1. The Enum Event

Event is used in the SpotEvent class to classify events connected with Spot.

This enum has seven fields. Table 3.4 shows details for each field.
### 3.2.4.2. The Spot Class

This abstract class, inheriting from `javax.swing.JComponent`, has three main tasks:

- Preparing UI functionality for deriving classes
- Listener functionality
- Information encapsulation

Besides these tasks, `Spot.java` has additional methods to support easier access, for instance, methods of the included `SpotInformation` class.

#### UI Functionality

Because a `Spot` is derived from `javax.swing.JComponent`, it is able to be displayed in Swing applications. The class cannot be used as an UI component itself, though, because of its abstract attribute. A class deriving from `Spot` is needed. This guarantees that visual variations of `Spots` can exist. The programmer’s effort to implement a new visual class is minimal. Section 3.2.6.5 shows the implementation of an example.

To keep the effort of creating new visual classes to a minimum, `Spot.java` provides basic features like tooltips, different states of activation (active, passive and mouse-over) and ‘Drag’n’Drop’ functionality.

#### Listener Functionality

`Spot.java` provides so-called ‘listener functionality’ which enables other UI components or program parts to be informed when changes happen to a `Spot`.

This functionality includes attaching and detaching an observer which implements the `SpotListener` interface (see Section 3.2.4.6). Only the `SpotFactory` class should implement this interface, though. Reasons and handling are discussed in Sections 3.2.4.3 and 3.2.4.7.
Like any other class implementing the listener pattern, the Spot class provides a method
\texttt{inform(Event e, Spot s)} that will inform all observers of a given \texttt{Event} (see Section 3.2.4.1).

Since this class is managed by the SpotFactory (see Section 3.2.4.3), every observer to the
Factory has to be informed in case a Spot should be removed. This can be done by invoking
the \texttt{deleteMe()} method of the desired Spot. It will call its \texttt{setVisible(boolean b)} method
with a \texttt{false} parameter and delegate the call to \texttt{inform(Event e)}.

**Information encapsulation** Each Spot contains a SpotInformation object which contains
all data relevant to the Spot. SpotInformation is described in detail in Section 3.2.4.4.

3.2.4.3. The SpotFactory Class

The SpotFactory class is the heart of the bachelorarbeit.spot package, and Curator.
SpotFactory manages all events that deal with Spots.

SpotFactory is implemented as a ‘Singleton’. This means that there can be only one
SpotFactory at the same time. This fact ensures easy access throughout the whole application.
A more important advantage is that it is impossible for multiple SpotFactories to
hamper each other. For instance, if it were allowed to have two SpotFactories, they would
either have to do the same work (redundancy) or neither of them would have full overview of
all Spots. The first case leads to unnecessary memory load and would slow down the system.
In the second case, it would be a huge effort to browse the Spots. Every class that would
like to be an observer of SpotFactory must know every instance of SpotFactory so that it
can attach to them and not miss an event.
Like the Spot class, SpotFactory implements the ‘Listener Pattern’. Observers can subscribe (‘attach’) to the SpotFactory. This is explained in detail in Sections 3.2.4.6, 3.2.4.7, and 3.2.4.8.

SpotFactory has two attributes to manage Spots: private java.util.ArrayList<Spot> spots and private java.util.ArrayList<Spot> selectedSpots. Both attributes are ArrayLists in which Spots can be stored.

private java.util.ArrayList<Spot> spots contains all Spots that exist at a given moment. As soon as a Spot has been created using the Factory, it will be added to the list. This is important for several points of the program code, for instance to store all Spots in the output XML file.

private java.util.ArrayList<Spot> selectedSpots, on the other hand, contains, as the name suggests, all currently selected Spots. This attribute is not necessary, strictly speaking. However, because requesting a list of selected Spots is a very common task of Curator, it is easier to provide a separate list containing these objects than checking every Spot in spots to know if it is active or not.

Apart from a few convenience methods, SpotFactory only offers the possibility to create Spots, listens to every event of its Spots and forwards these events. However, if a Spot has been created, this event will be distributed to the observers as well. The differences between SpotListeners and SpotFactoryListeners will be explained below in detail (see Section 3.2.4.8).

To create a Spot the user must get the only instance of SpotFactory by calling the ‘Singleton Pattern’ method SpotFactory.getFactory().

For instance: SpotFactory sf = SpotFactory.getFactory(). The user is then able to create a Spot using one of the createSpot methods. sf.createSpot(new Point(100,200)) will create a Spot with the given Point (100, 200). Consequently, SpotFactory will

- create a NonResizableSpot (see Section 3.2.6.5),
- add it to the spots list (see above),
- attach itself to the list of the new Spot’s observers
- deselect all other Spots, select only the new Spot\(^1\) and add the new one to the selectedSpots list,
- and finally inform every SpotFactoryListener of the creation.

Observers of the SpotFactory class will always be up to date with every Spot.

\(^1\) This behavior is easy to understand to the end-user
3.2.4.4. The SpotInformation Class

The SpotInformation class is a wrapper class which contains all information important to the Spot class. Providing a separate class to support the Spot class offers two advantages.

First, all values are encapsulated in one object so that it is easier and faster to handle. These values are the name (name), dimension (dim), top left point of the Spot (start), active or inactive state (active), a hash value for storage and comparison reasons (hash) and all attachments which belong to the Spot.

Second, saving the wrapped information is easier. XML frameworks such as ‘Simple XML Serialization’ can store an object into an XML file with all its attributes.

Apart from the attributes mentioned above, the class is provided with a constructor which uses a Spot as argument (public SpotInformation(Spot spot)). This makes it easy to create and encapsulate data from a given Spot.

Comparing two Spots or two SpotInformations is done by comparing the hash codes only. This is because a simple comparison of the starting point and dimension leads to serious problems when a Spot is dragged by the end-user and the coordinates of start change. Spots are compared by comparing its SpotInformation.

attachObject(SpotAttachment sa), removeObject(SpotAttachment sa) and removeObject(String s) are important methods to, as their names suggest, attach a URL or a file to a Spot, i.e. to its SpotInformation. These objects can be removed either by a given SpotAttachment or by a String. removeAllAttachedObjects() clears the list with the objects, but the copies of the files remain.

public java.util.List<String> relativizeAttachedObjectsURI (URI uriToRelativize) relativize all attached objects to a given URI, if possible.

3.2.4.5. The SpotEvent Class

The SpotEvent class is used as an argument for all the methods of the SpotListener and SpotFactoryListener interfaces. It is a wrapper class and contains either the Spot which has caused the event, a list of all selected Spots at the time of the event or both. SpotEvent is used for both SpotListeners and SpotFactoryListeners.

3.2.4.6. The SpotListener Interface

The SpotListener interface is the counterpart of the ‘Listener Pattern’ implementation in the Spot class. A class that wishes to listen to a Spot must implement this interface to do so.

The interface itself offers six methods, strongly related to the enum Event. The six methods are listed in Table 3.5.
Table 3.5.: Methods a SpotListener must provide

<table>
<thead>
<tr>
<th>Method</th>
<th>Raising event</th>
</tr>
</thead>
<tbody>
<tr>
<td>public void selectedExclusive(SpotEvent s)</td>
<td>Fired if a Spot was clicked without modifiers. Only this Spot is now selected</td>
</tr>
<tr>
<td>public void selected(SpotEvent s)</td>
<td>Fired if a Spot was clicked with modifiers. This Spot could have been selected together with other Spots</td>
</tr>
<tr>
<td>public void deselected(SpotEvent s)</td>
<td>Fired if a Spot was clicked and deselected.</td>
</tr>
<tr>
<td>public void renamed(SpotEvent s)</td>
<td>Fired if a Spot was renamed.</td>
</tr>
<tr>
<td>public void deleted(SpotEvent s)</td>
<td>Fired if a Spot was deleted using its deleteMe() function.</td>
</tr>
<tr>
<td>public void uriChanged(SpotEvent s)</td>
<td>Fired if a URI of a Spot was added or removed.</td>
</tr>
</tbody>
</table>

3.2.4.7. The SpotFactoryListener Interface

The SpotFactoryListener interface is the counterpart of the ‘Listener pattern’ implementation in the SpotFactory class. A class that wishes to listen to the SpotFactory and with it its Spots (see Section 3.2.4.8) must implement this interface to do so.

The interface itself offers only six methods, strongly related to the enum Event. The six methods are listed in Table 3.6.

Table 3.6.: Methods a SpotFactoryListener must provide

<table>
<thead>
<tr>
<th>Method</th>
<th>Raising event</th>
</tr>
</thead>
<tbody>
<tr>
<td>public void selected(SpotEvent s)</td>
<td>Fired if a Spot was clicked with modifiers. This Spot could have been selected together with other Spots</td>
</tr>
<tr>
<td>public void deselected(SpotEvent s)</td>
<td>Fired if a Spot was clicked and deselected.</td>
</tr>
<tr>
<td>public void renamed(SpotEvent s)</td>
<td>Fired if a Spot was renamed.</td>
</tr>
<tr>
<td>public void deleted(SpotEvent s)</td>
<td>Fired if a Spot was deleted using its deleteMe() function.</td>
</tr>
<tr>
<td>public void created(SpotEvent s)</td>
<td>Fired if a Spot was created by the SpotFactory. Only this Spot is now selected</td>
</tr>
<tr>
<td>public void uriChanged(SpotEvent s)</td>
<td>Fired if a URI of a Spot was added or removed.</td>
</tr>
</tbody>
</table>
3.2.4.8. The SpotListener and SpotFactoryListener Interfaces: a Comparison

At a first glance, the SpotListener and SpotFactoryListener interfaces look quite similar. In fact, they have five out of six methods, in particular

- `public void selected(SpotEvent s),`
- `public void deselected(SpotEvent s),`
- `public void renamed(SpotEvent s),`
- `public void deleted(SpotEvent s),` and
- `public void uriChanged(SpotEvent s),`

in common.

However, SpotListener has `public void selectedExclusive(SpotEvent s)` meanwhile SpotFactoryListener has `public void created(SpotEvent s)`.

SpotFactory is the manager class for Spot and has two list attributes which contain all Spots or all selected (active) Spots. If a Spot is clicked without any modifiers like the ‘Alt’ or ‘Control’ keys, it will be selected exclusively. This is important for the SpotFactory class, but it does not make any difference to other classes whether a Spot is selected with modifiers or not. It will receive a list of all selected Spots with the SpotEvent.

For all classes but SpotFactory it is important to be informed when a new Spot has been created. If a class subscribes to SpotFactory any subscriber will be informed as soon as another instance of Spot has been created. Now this information can be used, for instance, to display information of the new Spot.

Figure A.4 shows a sequence diagram of a spot that was clicked without modifiers.

3.2.4.9. The PersistentSpot Class

A PersistentSpot contains information about a Spot. Its attributes are the name (`name`), dimension (`dimension`), location (`location`), and a list of `Strings` with relative URLs (`relativeURLs`).

The list of relative URLs are relative to the directory chosen by the end-user in Curator’s Options frame.
3.2.5. User Interface Components

This section deals with the bachelorarbeit and bachelorarbeit.ui packages and their sub-packages. These packages include the main window and the Options window of the application, a dialogue and seven sub-classes of basic Java components. Figure 3.10 and Table 3.7 show the contents of the package. These classes are described in detail below.
Table 3.7.: User interface related classes

<table>
<thead>
<tr>
<th>Package</th>
<th>Class Name</th>
<th>Parent Class</th>
</tr>
</thead>
<tbody>
<tr>
<td>bachelorarbeit</td>
<td>BachelorArbeitView.java</td>
<td>FrameView</td>
</tr>
<tr>
<td>bachelorarbeit.ui.imagepanel</td>
<td>ImageLayerPanel.java</td>
<td>JPanel</td>
</tr>
<tr>
<td>bachelorarbeit.ui.imagepanel</td>
<td>ImagePanel.java</td>
<td>JLayeredPane</td>
</tr>
<tr>
<td>bachelorarbeit.ui.newmap</td>
<td>NewMapDialogue.java</td>
<td>JDialog</td>
</tr>
<tr>
<td>bachelorarbeit.ui.options</td>
<td>OptionsFrame.java</td>
<td>JFrame</td>
</tr>
<tr>
<td>bachelorarbeit.ui.palette</td>
<td>InformationProviderPanel.java</td>
<td>JPanel</td>
</tr>
<tr>
<td>bachelorarbeit.ui.palette</td>
<td>LinkageList.java</td>
<td>JList</td>
</tr>
<tr>
<td>bachelorarbeit.ui.palette</td>
<td>LinkagesPanel</td>
<td>JPanel</td>
</tr>
<tr>
<td>bachelorarbeit.ui.palette</td>
<td>SpotLabel</td>
<td>JLabel</td>
</tr>
<tr>
<td>bachelorarbeit.ui.palette</td>
<td>SpotList</td>
<td>JList</td>
</tr>
</tbody>
</table>

3.2.5.1. The BachelorArbeitView Class

The class BachelorArbeitView is the main window of the application Curator. All tasks the tool offers are done in this window. Figure 3.11 and Table 3.8 show the component parts of Curator and its position in the window. These parts are explained below.
3.2.5 User Interface Components

Figure 3.11.: The *Curator* application and its component parts

Figure 3.12.: Sidebar of *Curator*
Table 3.8.: Overview of Parts of Curator

<table>
<thead>
<tr>
<th>Nr.</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Create new Spot</td>
</tr>
<tr>
<td>2</td>
<td>Switch to another Project</td>
</tr>
<tr>
<td>3</td>
<td>Inactive Spot</td>
</tr>
<tr>
<td>4</td>
<td>Image</td>
</tr>
<tr>
<td>5</td>
<td>List of Spots</td>
</tr>
<tr>
<td>6</td>
<td>Linkages</td>
</tr>
<tr>
<td>7</td>
<td>SpotLabel</td>
</tr>
<tr>
<td>8</td>
<td>Button to delete a Spot</td>
</tr>
<tr>
<td>9</td>
<td>Attachments</td>
</tr>
<tr>
<td>10</td>
<td>Selected Spot is a map</td>
</tr>
<tr>
<td>11</td>
<td>Button to remove an attachment</td>
</tr>
</tbody>
</table>

**UI Description**  
1 allows the end-user to drag a Spot onto the map or image he has loaded. This Spot, class type NonResizableSpot, is a 32 by 32 pixels image. Curator is designed to support multiple types of Spots (see Sections 3.2.4.2 and 3.2.6.5 for details). Is it quite possible that other sizes or images than the one presented in NonResizableSpot may be implemented and displayed here.

Between 1 and 2, a label helps the user and tells him that he can drag and drop a Spot\(^2\). This label is shown in Figure 3.22 on page 65.

2, a JComboBox, contains all data-sets the user has loaded or created during his session. It allows the user to restore a data-set he has worked on before. By selecting a data-set, it is convenient to switch from one map to another. If a map contains a linkage marked as a map and the user created a new map of a given attachment, he can easily switch between these maps. All spots are restored as well. This functionality is the same as saving the current data-set and loading the desired XML file using the menu.

3 shows a Spot that has been dragged onto a map. In this particular case, a NonResizableSpot is shown which is currently not selected. It is possible to select multiple Spots at once, for instance to attach a file or URL to all selected Spots. This is done by holding the Command button (Mac) or Control button (all OS) on the keyboard and clicking on the desired Spots (Figure 3.18 on page 57).

4 is the area where a loaded image or map is displayed. If the image is bigger than the size of the area it should be displayed in, scroll bars will appear. This behaviour guarantees a more precise positioning of a Spot than if the image is scaled.

5 is a list that contains all Spots the user has created for a specific map or image. All currently selected Spots are highlighted (see Figure 3.18).

\(^2\) Only available if selected in the Options
6 consists of two separate tabs. The Information tab shows information on the currently selected Spot or Spots. It allows to delete one or more Spots (see Section 3.2.5.5). The Linkages tab displays the files linked to the Spot(s) (see Section 3.2.5.7).

7 displays the currently selected Spot. This helps the user identify the selected Spot. Of course, this feature makes more sense in case many different Spot implementations are displayed on the map or image.

8 With this button, a user can delete all currently selected Spots. Figure 3.16 shows the dialogue that asks the user if he really wants to delete the Spots. All selected Spots are shown in this dialogue.

9 shows all attachments of the selected Spot. To help the user identify his attachments, their paths and file names are shown.

10 indicates if an attachment is a map or not. This feature is explained in Section 3.2.5.7.

11 allows the user to remove attachments from the list of the current Spot. The attachment is only removed from the list; its physical representation, if it is a file, remains untouched.

The menu of Curator is structured as followed:

- **File**
  - *Load Image...* - Loads a new image as a base for a data-set, see Paragraph Functionality on page 50
  - *Save Project* - Saves a data-set in the folder chosen under Options, see Section The Utilities Class
  - *Load Project...* - Loads a data-set, see Paragraph Functionality on page 50
  - *Options...* - Invokes the Options dialogue, see Section The OptionFrame Class
  - *Exit* - See Section The CuratorExitListener Class

- **Tools**
  - *Create new Project from Attachment...* - See Section The NewMapDialogue Class

- **View**
  - *Show Spot Information* - Toggles Curator’s sidebar, see Section Sidebar

- **Help**
  - *About...*

**Functionality** BachelorArbeitView only contains logic needed for updating components; logic like storing the created data in an XML file is externalised to other classes like
Utilities.java. Important methods are `private boolean loadProject(final File xml)` and `private boolean loadImage(File image) throws IOException`. BachelorArbeitView also contains an inner class, `class LoadTask extends Task<Void, File>`.

The class LoadTask wraps the method loadProject and adds UI functionality. This task indicates the user whether a project was loaded successfully or not (see Section 2.4.5). Apart from that, LoadTask profits form all other advantages of a Task.

The method loadProject loads a data-set which was saved in an XML file before. It enables the user to save its work and continue later.

The method loadImage loads an image from a specified file and sets it as the background-image (see Section 3.2.5.2 for details). The name of the corresponding XML file is set in the Options dialogue box to indicate the user which data-set he is currently working on.

The method tries to load the map or image associated with the chosen data-set file. After that, it restores all Spots from the XML file using a method in the Utilities class (see Section 3.2.6.4) and displays them. Spots are added using the SpotFactory class to offer the advantages described in Section 3.2.4.3. Finally, the loaded data-set is added to the JComboBox to enable the user to switch the project.

When done, the Task this method is wrapped in either displays an success or error message at the bottom of the tool.

Other functionality of BachelorArbeitView, like saving Spots, only delegates tasks to another class or updates the user interface.

To keep this class in synchronisation with the SpotFactory, it implements the SpotFactoryListener interface described in Section 3.2.4.7.

### 3.2.5.2. The ImageLayerPanel and ImagePanel classes

ImageLayerPanel has only one task: displaying an image.

Since Java lacks the possibility to easily configure the design of elements, this class has been created. JLabel however offers the use of images, but only instances of the javax.swing.Icon interface are accepted as image. So JLabel is not suitable for this task, especially handling bigger files is problematic.

ImageLayerPanel inherits from JPanel and accepts any java.awt.Image in its setImage(Image img) method. It also overwrites the paint(Graphics g) method of its parent class JPanel and draws its image.

The ImagePanel class is to display a map or an image onto which Spots can be dragged.
**ImagePanel** inherits from `javax.swing.JLayeredPane` which is, as the name suggests, like multiple `JPanel`s one upon another. It allows an easy handling of adding Spots without explicitly calling the `paint` method.

The background layer is set to be under the normal layer (`JLayeredPane.DEFAULT_LAYER`). Spots dragged onto `ImagePanel` are added as high as `JLayeredPane.POPUP_LAYER`, above the background, otherwise they would be hidden by the background.

`ImagePanel` offers a convenient method to test if Spots can be dragged onto it. This method can be used by a `TransferHandler` like `SpotTransferHandler` described in Section 3.2.6.6.

### 3.2.5.3. The NewMapDialogue Class

This class, shown in Figure 3.13, is a modal dialogue which lists all attachment to a Spot marked as a map. End-users can select an attachment and create a new data-set out of it. The following example explains the process.

![Figure 3.13: Curator's dialogue to create a map from an attachment](image)

A user has created a Spot on a map, for instance a map of Sydney with a Spot on the Circular Quay region. This Spot has a map of Circular Quay as attachment, and the attachment has been marked as a map in Curator. When he opens the menu entry ‘Tools -> Create new Map from Attachment...’, `NewMapDialogue` will be displayed. Now he can select the corresponding entry in the list and click OK. The open data-set will be saved and closed and a new data-set with the map of Circular Quay will be created. If he wants to, the user can switch between these two projects using the `JComboBox`.

This procedure helps the user to create a whole series of connected maps or images. Though he could save and close the current data-set and open a new one manually, the method described above is much more convenient.
3.2.5.4. The OptionFrame Class

This Options dialogue allows the user to view and changes settings of the Curator program.

![Options Frame](image)

Figure 3.14.: The Options Frame

The first row, ‘Name of XML-File’, shows the name of the data-set the user currently is working on. The corresponding JTextField is disabled so that the user cannot edit it by himself. This is because the XML file must have the same name as the image, it is the way Focus maps the XML file with the image. If an image called ‘Circular Quay.jpg’ is loaded, the corresponding XML file name is ‘Circular Quay.xml’.

The second row, ‘Target Folder’, contains an editable JTextField and a JButton. This text field contains, as the name suggests, the target folder where the user wants his data to be saved. All files of the data-set, the background image, XML file, and the Spot attachments will be stored in this folder.

With the JCheckBox named ‘User Guidance on Start Up’, the user can decide if he needs help to use Curator. By default, user guidance is activated to help first time users working with Curator. This guidance is described in Section 3.2.7.3.

3.2.5.5. The InformationProviderPanel Class

This class, extending javax.swing.JPanel, displays several information on the selected Spot or Spots. It also enables the user to delete all Spots he has selected.

If a user has selected one Spot, he will get information on the name, the size and the number of linkages he has attached to the Spot. In addition the Spot itself is displayed in this panel. The right-hand panel of Figure 3.15 shows how the class looks like if a NonResizableSpot is selected.
When more than one Spot is selected, the amount of Spots and an image is displayed. The left-hand side of Figure 3.15 shows the behaviour if more than one Spot is selected.

Figure 3.15.: Displayed information if multiple Spots are selected (left) and only one Spot is selected (right)

If the user wants to delete one or more Spots, he has to confirm this action. Figure 3.16 shows this dialogue. An implementation of an undo functionality would be better for the workflow, but due the time limit, this option could not be implemented. See details in Chapter 5.
To keep this class in synchronisation with the SpotFactory, it implements the SpotFactoryListener interface described in Section 3.2.4.7.

### 3.2.5.6. The LinkageList Class

This list extending JList contains all attachments of the currently selected Spot or Spots.

Displayed are the file name of the attachments or the URL if an URL was attached. If more than one Spot has been selected in the SpotList, all attachments of all Spots are shown. If an attachment has been linked with every Spot, it is shown as described above. But if an attachment is linked with only some of the Spots, a ‘*’ marks this attachment.

To keep this class in synchronisation with the SpotFactory, it implements the SpotFactoryListener interface described in Section 3.2.4.7.

Chapter 5 shows extensions for this list which would improve the usability but could not be implemented due to the limited time.

### 3.2.5.7. The LinkagePanel Class

This class, a subclass of JPanel, displays the LinkagePanel described above, a JCheckBox and a JButton to delete linkages. Figure 3.17 shows the LinkagePanel class.
With the JCheckBox, the user is able to mark an attachment as a map. This is needed for the Cruiser system to distinguish maps from other images. As a side-effect, Curator is able to list in NewMapDialogue only attachments marked as map, and ignore any other attachments.

This JCheckBox is disabled as soon as more than one linkage is selected. Otherwise, the question what should be displayed when attachments are maps and no maps must be answered. Again, due to time reasons, this functionality is not implemented.

However, if multiple linkages are selected, the user can delete them all at the same time by pressing the corresponding button.

To keep this class in synchronisation with the SpotFactory, it implements the SpotFactoryListener interface described in Section 3.2.4.7.

3.2.5.8. The SpotLabel Class

The only task of this class is to display the currently selected Spot.

This class, inheriting from JLabel, can be used as described above in Section 3.2.5.1 to help the user to identify the Spot selected last. This can be useful if there is more than one implementation of Spot.

In Figure 3.11 on page 48 a SpotLabel is implemented next to the name of the Spot, marked with number 7.

To keep this class in synchronisation with the SpotFactory, it implements the SpotFactoryListener interface described in Section 3.2.4.7.
3.2.5.9. The SpotList Class

A SpotList is able to display all Spots listed by the SpotFactory.

Selected Spots are also selected in this list. On the other hand, the user can select Spots using the list. Especially when many or all Spots should be selected, probably to attach a file to them, this is a convenient method.

To keep this class in synchronisation with the SpotFactory, it implements the SpotFactoryListener interface described in Section 3.2.4.7.

Figure 3.18 shows a few selected Spots.

![Multiple Selection of Spots](image)

Figure 3.18.: Multiple Selection of Spots

3.2.6. Other Classes of Curator

This section deals with components of the program which are not yet mentioned. Figure 3.19 and Table 3.9 show the components described.
3.2.6 Other Classes of *Curator*

![Image of other components of Curator]

Figure 3.19.: Other Components of *Curator*

Table 3.9.: Other Components of *Curator*

<table>
<thead>
<tr>
<th>Component</th>
<th>Package</th>
</tr>
</thead>
<tbody>
<tr>
<td>BachelorArbeitApp.java</td>
<td>bachelorarbeit</td>
</tr>
<tr>
<td>CuratorExitListener.java</td>
<td>bachelorarbeit.utilities</td>
</tr>
<tr>
<td>SettingsDescriptor.java</td>
<td>bachelorarbeit.utilities</td>
</tr>
<tr>
<td>Utilities.java</td>
<td>bachelorarbeit.utilities</td>
</tr>
<tr>
<td>MoveBackgroundTask</td>
<td>bachelorarbeit.utilities.Utilities</td>
</tr>
<tr>
<td>MoveSpotsLinkagesTask</td>
<td>bachelorarbeit.utilities.Utilities</td>
</tr>
<tr>
<td>NonResizableSpot.java</td>
<td>bachelorarbeit.spot.implementations</td>
</tr>
<tr>
<td>SpotTransferHandler.java</td>
<td>bachelorarbeit.spot.dnd</td>
</tr>
</tbody>
</table>

### 3.2.6.1. The BachelorArbeitApp Class

This class contains the method `public static void main(String[] args)` and is responsible for starting *Curator*.

*BachelorArbeitApp* also contains a startup method which displays the main window of the application, *BachelorArbeitView*. Details of the life cycle of the Swing Application Framework are explained in Section 2.4.2.

### 3.2.6.2. The CuratorExitListener Class

The *CuratorExitListener* class implements the `ExitListener` interface of the Swing Application Framework (see Section 2.4.2).
When the user wants to close the application, Swing Application Framework (SAF) asks all `ExitListener` if they are ready to quit the program or not. In this particular case, the `public boolean canExit(EventObject e)` method checks if the user has created Spots that should be saved before the program exits. The user will be asked to do so and the answer will be stored in the class.

When SAF finished asking all `ExitListener` if they are ready, and no `false` was returned from any Listener, SAF will call each `public void willExit(EventObject e)`. If the user wants to save his progress, this will be done now using the `Utilities`'s `save` method.

### 3.2.6.3. The `SettingsDescriptor` Class

`SettingsDescriptor` is responsible for saving and loading application settings that are not covered by the Swing Application Framework.

The class is implemented as a Singleton. It is responsible for storing the settings of the Options dialogue and restoring them when the program is started the next time. This is done with the `java.util.prefs.Preferences` class. The data is stored persistently in an implementation-depending backing-store [JDo].

### 3.2.6.4. The `Utilities` Class

`Utilities` offers a few methods which are important for `Curator` but do not fit in other classes. They are externalised to improve reusability. `Utilities` also contains two inner classes, `MoveBackgroundTask` and `MoveSpotsLinkagesTask`, which are described below in Section 3.2.6.4 and Section 3.2.6.4. The below list provides an overview of the important methods of `Utilities`.

- `public static File showFileChooser(final String[] extensions, final String description)`
- `public static ArrayList<Spot> loadSpotsFromXML(File xml) throws Exception`
- `public static void storeSpotsInXML(List spots, File target) throws Exception`
- `public static int countFiles(File source)`
- `public static Point getInfoPoint(Point p)`
- `public static int getUserOS()`
- `public static void save(File targetFolder, File targetFile, ImagePanel ip, Application app)`
The `showFileChooser(final String[] extensions, final String description)` method opens a Java dialogue which lets the user choose a file (see Figure 3.26 on page 68).

This method offers to configure the dialogue: the variable `extensions` limits the files shown or accepted by the `JFileChooser` to files which end with one of the passed `Strings`. The `String` `description` shows a description next to or instead of the file extensions.

The `loadSpotsFromXML(File xml)` method reads the passed XML file and parses it. A list of all read `Spots` will be created and returned. This method is used to load `Spots` from an XML file.

The `storeSpotsInXML(List spots, File target)` method stores all `Spots` in the file which are passed to the method. List `spots` represents the list of `Spots` to be stored. The implementation of reading from and writing to XML files is done by the `javax.xml.stream.XMLStreamReader` and `javax.xml.stream.XMLStreamWriter` classes. Listing 3.1 shows an example XML file created by `Curator`.

```xml
<?xml version="1.0" encoding="UTF-8"?>
<Spots>
  <Spot Name="The Rocks">
    <Dimension>
      <Width>32</Width>
      <Height>32</Height>
    </Dimension>
    <Point>
      <X>68</X>
      <Y>227</Y>
    </Point>
    <Hash>501153811</Hash>
    <Linkages>
      <Linkage rel="true" map="false">The Rocks.jpg</Linkage>
    </Linkages>
  </Spot>
  <Spot Name="Sydney Opera House">
    <Dimension>
      <Width>32</Width>
      <Height>32</Height>
    </Dimension>
    <Point>
      <X>323</X>
      <Y>134</Y>
    </Point>
  </Spot>
</Spots>
```

---

3 Depending on the operation system's implementation
4 This XML file was formatted to improve reading.
The `countFiles(File source)` method counts all files in a given directory, if `source` is a folder, or returns 1 if `source` is a file. This method is used in Curator to indicate the process of moving attachments to the specified path when storing Spots.

`getInfoPoint(Point p)`: If a Spot is dragged by the user and then released, the tip of the balloon icon of a NonResizableSpot will be at the position of the tip of the mouse cursor. `getInfoPoint(Point p)` adjusts the position of the Spot as described. Otherwise, the top left corner of the Spot would be at the mouse cursor’s position which could confuse the user. This method works only for a NonResizableSpot.

`getUserOS()` returns either Utilities.WINDOWS, Utilities.LINUX, Utilities.MAC or Utilities.OTHER_OS, depending on the system property ‘os.name’. This method can be used when the user operating system must be known and an operating system depending implementation is needed. For instance, Curator uses different shortcut keys on Mac OS X and Windows. `getUserOS()` enables this possibility to offer a feeling of a OS native application.

`save(File targetFolder, File targetFile, ImagePanel ip, Application app)` is the method called if a user wants to save his data-set. First, it creates `targetFile`, the XML file the Spots will be saved in, and `targetFolder`, the folder the project will created in, if these Files does not exist. Then `storeSpotsInXML`, described above, is called which stores the data in the XML file.

Instead of saving the image displayed in Curator’s ImagePanel, the ImagePanel itself with all its components will be drawn and then saved as a JPG file. The advantage is that the Spots the user created before are exported on the image. Otherwise, another program which uses the data-set would have to render the Spots itself. This image is stored in `targetFolder`.

Finally, a MoveSpotsLinkagesTask, see Section 3.2.6.4, moves all the linkages of a Spot.

The MoveBackgroundTask Class MovebackgroundTask copies the image or map of the data-set to the specified target folder. Cruiser needs this file to display it.

This inner class wraps moving the image or map of the data-set into a Task. By doing so, the Event Dispatch Thread and with it the GUI are not blocked.

This class is not used anymore because the ImagePanel and its Spots will be stored together as an image. However, if it would be necessary in future to provide the original image with
a data-set, this class could be used to do so. This is why it is described here and still a part of Curator.

**The MoveSpotsLinkagesTask Class**  This inner class is responsible for copying all linkages of every Spot to the specified target folder.

The method first counts all files which are about to be moved. This is to illustrate the duration of the process using the process bar in the main window BachelorArbeitView. So the user can estimate the duration of the process. During this count, Curator looks after Spots that have the same linkage. If so, such a linkage will be added to a HashMap.

![Sample file structure of a Curator project with two data-sets](image)

Figure 3.20.: Sample file structure of a Curator project with two data-sets

When this is done, this algorithm starts to copy the chosen files. If it finds a file which is listed in the HashMap, it will create a link instead of copying the file again. This saves hard disk space because no doublets are created. This is important especially if bigger files like videos are attached to multiple spots.

To link to another file, the algorithm will create a file named after the original file plus the ending ‘.link’. This file is a text file which contains the folder and file name of the actual file. Cruiser will recognise this link and handle the actual file instead. Using this kind of link to a file is consistent because there is no need for an operating system dependent implementation.

Figure 3.20 shows a project which comprises two data-sets (‘Main.xml’ and ‘Circular Quay.xml’). The folder ‘Circular Quay’, ‘Sydney Opera House’ and ‘The Rocks’ each represent a spot. They all include attached files. The file ‘The Rocks.jpg.link’ is link file described above. It links to the file ‘The Rocks.jpg’ in the folder ‘The Rocks’ which represents another spot.

During the copying, this Task informs the end-user about the process by updating the process bar and the message label, discussed in Section 2.4.5, shown in Figure 2.20.
3.2.6.5. The NonResizableSpot Class

This class is an example implementation of the abstract class Spot described in Section 3.2.4.2 on page 40. It contains three ImageIcon for each state of Spot, active, passive and mouseOver. Its paint method requests this status and draws this image. All the other work is done by the Spot class.

However, this class is only an example implementation of a Spot. Other Spots can be created in the same manner. It is also possible to adjust the dimensions of the Spot.

3.2.6.6. The SpotTransferHandler Class

The SpotTransferHandler class is responsible for the drag and drop feature of Curator when the user creates or moves a Spot. It packs the data which should be dragged (createTransferable (JComponent c)). This packed object can be moved or copied, depending of the task of this drag and drop operation. In case of creating a Spot, TransferHandler.COPY is used, changing the position of a Spot uses TransferHandler.MOVE.

When the user drops the data, the public boolean importData(TransferHandler.TransferSupport support) method is called. In case of TransferHandler.COPY, this method creates a new Spot at the cursor position where the drop occurred. When a Spot was moved, this method updates the position of it.

Details of the superclass of SpotTransferHandler can be found in [Sun Microsystems, 2008].

3.2.7. Other Features of Curator

This section deals with features and implementation details used in Curator which do not fit into any of the other sections described above.

3.2.7.1. Multi-Language Ability

Curator is designed to support multiple languages. At the moment, Curator is available in English and German.

Enabling multiple languages, so-called internationalisation, allows Java programs to be adapted for different languages and/or regions without the need of changing program code or recompiling. All texts, lettering and other textual elements like GUI components in a program are stored in a separate file, typically a properties file. All that has to be done to adapt a program into another language is to create a new properties file for this language. For example, if the default language properties file is ‘Spot.properties’, the German language file for Germany would be ‘Spot_de_DE.properties’. ‘de’ represents the language code, ‘DE’ the country code. Australia’s properties file would be ‘Spot_en_AU.properties’. If Java cannot
find a properties file that fits the language and country code, the default language, English in case of *Curator*, will be used.

![Figure 3.21.: Example of key/value pairs for internationalisation](image)

This file organises all strings in key/value pairs. The key represents an identifier the program uses to retrieve the value, the actual text which is to be displayed. The IDE Netbeans offers to edit all language files together in a list. Figure 3.21 shows a properties file used in *Curator* for the default language, which is English, and German.

Listing 3.2 shows how the success message of loading a project is accessed using a properties file. Section 2.4.3 discusses resource management of the Swing Application Framework in detail.

```java
protected void succeeded(Void v) {
    setMessage(java.util.ResourceBundle.
               getBundle("bachelorarbeit/Bundle").
               getString("successfullyLoadedXML"));
}
```

Listing 3.2: Example of accessing a language properties file

### 3.2.7.2. Drag and Drop

*Curator* supports drag and drop which helps the user to interact with the program. This feature is used twice in the program:

1. When creating a **Spot** or adjusting its position on the map or image and
2. when attaching files or URLs to a point of interest.

Using drag and drop to create a marked spot on a map or an image, it is very easy to position it on a map. It feels natural to move it on the map, comparable to attaching a pin on a physical board.

Another method to create a spot could be just to double-click on the position where the **Spot** should be created, but then the question how to move it to another position must be answered.
as well. Using drag and drop for both creating and moving a Spot is easy to remember for the user and more consistent. A help label next to the spot icon is shown when the program is started first or the user has enabled it in the Options dialogue.

The second use of drag and drop in Curator is attaching files or URLs to a Spot.

Users can drag files from their operating system’s file browser and drop them in the sidebar of Curator to attach them to a Spot. The user can select a number of files and directories simultaneously in his file browser. The fact that the mouse cursor indicates a copy process of these files helps the user to realise that his files will be copied to the specified target folder and not moved. The usability testing study, which is discussed in Chapter 4 on page 69, will show whether the target user group feels comfortable with drag and drop.

Chapter 5 explains another possible method to attach linkages to a Spot.

3.2.7.3. User Guidance

Curator offers the user a guidance system which should help him to use the application.

When Curator is started, the user is asked to load an image to create a new data-set. If the program is started for the first time, user guidance is activated. He can now start to drag Spots onto the map he has loaded before. In case the user does not know how to create a Spot, a big label will request him to drag the Icon left of the label and drop it onto the desired position on the map or image. Figure 3.22 shows this label.

![Figure 3.22: Label to indicate that a spot can be dragged and dropped](image)

The next step in Curator’s workflow is to attach files or URLs to the Spot the user has just created. A label, shown in Figure 3.23, indicates where this can be done.
Finally, to give the Spot a unique describing name, the user can rename it either by double-clicking the Spot itself or the entry in the list in the sidebar. By default, Spots are named in the order they were created. Figure 3.24 shows the label which indicates how to rename a Spot.

Due to lack of time, only basic icons have been created for user guidance. They should attract the user’s attention so that he is able to easily start and use Curator. The quality requirements analysis in Section 3.1.2 gives preference to other requirements.

3.2.7.4. Sidebar

Curator’s main window, BachelorArbeitView, contains a sidebar with the list of all Spots, a list of linkages and more information. This sidebar, a JToolBar, is detachable from the window. Now the user is able to move this bar to a position more convenient to him than
the default position, for instance on another monitor. It is also possible to hide this bar, for example in case of a low screen resolution and a big map or image. The image area is larger and it is easier to move spots.

Figure 3.25 shows \textit{Curator} when the sidebar is detached.

![Curator with its sidebar, detached](image)

Figure 3.25.: \textit{Curator} with its sidebar, detached

### 3.2.7.5. Apple Menu Bar

To provide a more native-looking experience when using \textit{Curator}, the application uses Apple’s Menu Bar when run on a Mac OS X machine. This is done by only one line in \texttt{BachelorArbeitView}:

```java
// use apple's menu bar, if its a mac, otherwise the app won't be affected
System.setProperty("apple.laf.useScreenMenuBar", "true");
```

Listing 3.3: Using Apple’s menu bar on Mac OS X

Figure 3.25 shows this menu bar on the very top of the image.
3.2.8. Used Frameworks and Foreign Code

This section lists and describes all third-party frameworks and code used in Curator.

3.2.8.1. ImagePreviewPanel

The class ImagePreviewPanel, developed by Urban [2005], allows to preview image files in a JFileChooser.

JFileChooser is able to show accessories when choosing a file. An accessory could be to preview a file, as an image, a video file, a PDF document or something different.

ImagePreviewPanel is used in Curator to enable the user to see a preview thumbnail of the image he wants to use as a map or an image when he starts a new data-set by loading an image. This can save a lot of time if the user does not know the name of the file he wants to load, so he can browse the file directly in the JFileChooser.

Figure 3.26 shows the dialogue to choose an image with the accessory ImagePreviewPanel on Mac OS X.

![ImagePreviewPanel](image.png)

Figure 3.26.: The ImagePreviewPanel used in an open file dialog

3.2.8.2. Swing Application Framework

The Swing Application Framework used in Curator is discussed in detail in Section 2.4.
4. Evaluation of *Curator*

In the previous chapter, the program *Curator* has been discussed in detail. A large part of it explains the graphical user interface of *Curator*. According to the requirements analysis in Section 3.1.2, usability is a very important aspect of the program. An empirical evaluation helps to find possible problems with the interface and design of *Curator*. This study helps to answer the question if the tool’s workflow is easy to understand and simplifies the process of creating a data-set.

In addition, *Curator* has been developed to save time when creating a data-set for Focus. Therefore, a second study compares the times needed to create a data-set manually as well as by using *Curator*.

4.1. Usability Evaluation

In this section, a usability test is described which was performed with the goal to improve the workflow of creating data-sets using *Curator* and to find comprehension issues in *Curator*.

4.1.1. Usability Testing with DECIDE

As a usability evaluation method, Usability Testing with DECIDE (see Sharp, Rogers, and Preece [2007]) as framework was chosen. Figure 4.1 shows the meaning of DECIDE.

- **D**etermine the goals the evaluation addresses
- **E**xplore the specific questions to be answered
- **C**hoose the evaluation paradigm and techniques to answer the questions
- **I**dentify the practical issues
- **D**ecide how to deal with the ethical issues
- **E**valuate, interpret and present the data

Figure 4.1.: DECIDE as framework for Usability Testing
A specific goal is needed; in the case of this study, the goals are to improve Curator’s workflow of creating data-sets for Focus, to analyse the user’s acceptance of using drag and drop, and to find other usability issues.

This study has four main questions which need to be answered:

1. Is drag and drop easy enough to use for the target group of Curator users?
2. Is the layout of Curator easily comprehensible to the target group of Curator users?
3. Are the dialogues in Curator clear enough that the user understands their consequences?
4. Is the user happy with the workflow of Curator?

These main questions will help to answer the first goal, the improvement of the workflow of creating data-sets. The second goal, the identification of comprehension issues is covered as well. This goal is checked during the whole study.

Usability Testing [Dumas and Redish, 1999] has been chosen as an evaluation paradigm and its techniques are used to answer these questions. With this method, a few end users (see Faulkner [2003] for details) are observed using the current version of Curator.

These users represent the primary target user group of Curator, working students who are familiar with using a computer but not necessarily with programming. Due to the budget and time limits, the other main group of users of Curator, museum employees, cannot be observed.

No special equipment is used in this test. Using pen and paper, the observer writes down the behaviour, reactions and interactions of the users. To get insight into the user’s thoughts, the test person will be asked to think aloud. This technique helps the observer and the evaluator to understand why the user behaved the way he did. There will be no voice or video recording during the session because the evaluation of such records would consume more time than available for the study. An advantage of this is, however, that the user might not be confused and embarrassed by the recording. After his session, the user is asked to fill in a questionnaire (see Section B.5 on page 98).

Before the usability test starts, the user has to complete a participant consent form, shown in Section B.3. In this form, the user is asked to confirm that he understands what the project is about, that he can withdraw from the study at any time and that no information about the user will be used in any way which reveals his identity.

### 4.1.2. Pilot Study

A pilot study is a trial run of the main usability study. The aim of a pilot study is to confirm the correctness of the study. This may include user tasks, questionnaire, scenarios and the like. In the case that the tasks are infeasible or the program shows any big issues so that it
is not possible to perform the study, they can be reviewed. It is also possible that there is a major usability issue which must be fixed first. By running a pilot study, a lot of time and money can be saved.

4.1.2.1. Curator Pilot Study

The main target of this study was to confirm the feasibility of the tasks and to estimate the duration of the tasks and the studies.

This pilot study was performed with one user. The tasks can be found in Section B.1, the description of the task is equal to the tasks of the main study, see Section 4.1.3.

Cognitions and Changes The conditions and techniques of the pilot study were the same as for the main test, described above.

Task 1, accessing the Options dialogue, showed that the program must have been started by the user already or that the user should be asked to open it.

Task 3 had also to be updated, it was not obvious that XML files represent data-sets to load. This test was run on a Mac OS X machine; its Java implementation of a java.nio.chosser shows all files and folders, any files that do not fit the filter are greyed out. A solution to this problem is to refine Task 3 and to provide help.

The pilot study took approximately fifteen minutes. Considering this small amount of time, a fourth task should be added. This will help to reach the study’s goals.

4.1.3. Usability Testing Session

In this section the usability testing session with the target users is described.

A usability testing session for Curator is designed to take thirty to sixty minutes in total.

The study comprises four tasks separated into two scenarios which the test persons are asked to perform. A short introduction will help the test persons to empathise with the scenarios.

In the first scenario, the test person should imagine to work in a small advertising agency in Europe. Since he was in holidays in Australia, he is asked to present a few photos in a creative way. The test person chooses Focus because of its natural way to share pictures. To set up the data, the program Curator is chosen.

Afterwards, the first task asks the test person to open Curator and change the default directory where the data-set is stored. This task is part of Question 3. In the second task, the user will create a data-set of inner Sydney. Hence, he will load an image showing central Sydney from a folder on the desktop. Once this is done, he will create a spot at Circular Quay and attach a map of this area. To indicate that this attachment of the spot is another
map, he will select the corresponding check box. To make it easier to identify this spot, he will rename it. The task will finish when the user saves his work. This task tests the general workflow of the program as well as drag and drop and Curator’s layout. It refers to Questions 1, 2 and 4.

In the third task, the test person will load an existing data-set of Circular Quay. He will be asked to change the position of a spot, remove a linkage and attach another image instead. Drag and drop is tested again with this task.

These first three tasks introduce the test person into the program. A second scenario examines whether the user is now able to create a data-set by himself. In this scenario, the test person works as a working student in a museum. His supervisor asks the test person to curate a virtual exhibition of the human body and its organs.

The test person will be given a few images showing organs, and an outline of a body as a background image for the data-set. He will be asked to create a spot for each organ and attach all fitting files. By performing this assignment the facilitator examines whether the participant is now able to create data-sets on his own.

Procedure of the Usability Testing Session Each study involves one test person at a time. The test person will receive a participant information sheet, see Section B.2 on page 93. This sheet answers common questions about the study.

After reading the information sheet, the test person will be asked to sign a participant consent form, shown in Section B.3, handling ethical issues of the usability study. The test person signs that he knows what this study is about, that he can withdraw from the study at any
time and that his involvement is strictly confidential and no information about him will be used in any way that reveals his identity.

After this, the study itself may start. Because the study requires the test person to know where Circular Quay in Sydney, Australia, is located, the observer asks him to show this place on a printed map.

The test person will then sit down in front of a computer and have some time to familiarise himself with it. The facilitator shows a short video in which people interact with Focus. He comments on this video and explains the components of the video. This is to ensure that the test persons understand what the study is used for. Also they familiarise with Focus; it is most likely that real users of Curator have seen Focus. This video can be found on the disc of this thesis. The test person will then be given the tasks, see above and Section B.4, and will be asked to read and perform them step by step.

During the tasks, the user is asked to think aloud. The facilitator will take notes of this and of the test person’s behaviour.

When the test person has finished his tasks, he is asked to complete the questionnaire shown in Section B.5. The facilitator will thank the person for his participation.

4.1.4. Evaluation of Testing

This section illustrates the results of the usability testing session described above. All questions of Section 4.1.1 will be answered here.
Though Faulkner [2003] showed that increasing the number of participants can result in a dramatic improvement in data confidence, schedule and budget called for a limitation of the number of test users participating in this study to seven persons. According to the advice in Section 3.1.3 on page 30, all participants have seen a video of Focus (see above).

The second study, which compares the time needed to create a data-set by using Curator and by creating it manually (described below in detail), is only meaningful if the same persons performs both tasks, manually and using Curator. Since creating data-sets manually requires knowledge of the XML language, two IT students perform both time and usability testing studies. This reduces a bit the significance of the question whether drag and drop is easy enough to use for the target group.

All test persons are students with various computer skills - four students with basic computer knowledge and three information technology (IT) students, according to the target group. One person was familiar with creating data-sets for Focus manually. Four test person did not know Focus before. Six test persons used the English language setting of Curator, one German person used the German setting. All of them finished the study; none withdrew from the study.

4.1.4.1. Is drag and drop easy enough to use for the target group of Curator users?

This question is answered by the questionnaire as well as by observing the test persons.

All persons agreed to strongly agreed with the statement ‘It was easy and intuitional to me to use drag and drop while positioning spots’. Two test persons (#1) first tried to create a spot by double-clicking on the desired place on the image. With the hint that a spot looks like an info icon, he instantly dragged the spot onto the map. Another test person (#5) double-clicked first on the map, then searched the menu for an adequate entry. Finally he
read the help label and was dragged info-icon on Circular Quay. All other test persons had no problem with creating a spot.

When the test persons were asked to attach files to a spot, five of them used the guidance. Two participants looked for an adequate menu entry. The describing text for this icon however could be more precise and indicate to drag and drop files from the operating system file manager. Some participants were given a hint that they can save time when attaching the same files to spots; they found out that multiple files could be attached to multiple spots at the same time.

Test persons had no trouble moving a spot to a different position in Task 2 and intuitively used drag and drop. Positioning a spot to a precise pixel however was difficult because Curator does not allow moving spots over another spot.

All people learned fast and could easily use drag and drop in Task 3.

Some quotations from the questionnaire and study:

- ‘How do I drop files to attach them to spots?’ (#1)
- ‘Moving is so easy!’ (#1)
- ‘Icon image centring should be better. (#3)’
- ‘As a Mac user, that [drag and drop] is the way how programs work! (#5)’
- ‘At first I thought I attach files like I loaded the map. (#6)’
- ‘Difficulty making minor adjustments to icons’

4.1.4.2. Is the layout of Curator easily comprehensible to the target group of Curator users?

This question is answered by the questionnaire as well as by observing the test persons. The section does not refer to the workflow of Curator, the workflow will be discussed below.

All test persons agreed with the statement ‘I liked the overall layout of Curator’. During the observation of the test persons using Curator, it was found out that people had no trouble to orient themselves but two of them looked up for a ‘Attach file’ menu entry first.

Test person #3 missed the possibility to resize Curator’s sidebar. He used the application maximised.

Only one person was confused by the (designed) mismatch of the term ‘attachment’ used in the task description and the term ‘linkage’ in the English version of Curator. He asked where the attachments of a spot are, but could solve this issue without help.

However, the Options menu item, located under ‘File -> Options...’, was expected by Mac users under ‘Curator -> Options’. Loading an image as well as loading and saving a data-set
was found immediately by everybody. Participant #5 wanted to use the menu entry ‘Load image...’ to attach an image file to a spot.

Some quotations from the questionnaire and study:

- ‘Good simple layout.’ (#1)
- ‘I would like the ability to adjust the size of the sidebar.’ (#3)
- ‘Attachments are Linkages, that should be clearer!’ (#1 regarding the intentional fact that ‘linkages’ are named ‘attachment’ in the user tasks)
- ‘Where can I delete attachments? Ah, there is a button.’ (#6)
- ‘It just takes a little time to get used to the functions of the program. (#5)’
- ‘There is no save as...!’ (#4)

4.1.4.3. Are the dialogues in Curator precise so that the user understands their consequences?

The statements #3 and #8 asked if the user understood the Options dialogue and if the other question dialogues were clearly understandable.

The test persons agreed that the Options dialogue is easy to understand and that he knew what he had to change in Task 1 (1x strongly agree, 3x agree, 2x somewhat agree)\(^1\). All persons immediately found where to change the target folder for the data-set output of Curator. Three of the test persons created the requested folder using the operating system, the rest typed the name of the folder directly into the text field. The first row of the Options dialogue, the name of the currently opened data-set, was not directly understood by three test persons.

The users somewhat agreed to strongly agreed that the other dialogues of Curator, as the dialogue to ask if the data should be saved before leaving the application, behaved as indicated and helped to use the program. One person claimed that he was asked to save his progress before leaving after he just saved the data.

4.1.4.4. Is the user happy with the workflow of Curator?

Again, users somewhat agreed to strongly agreed with the statement ‘I liked the workflow of creating a data-set based on only one map’. All test persons first kept to the workflow suggested by Curator’s guidance system. Four of them later renamed the spots first and then attached files. Two users tried to drag a background image into the application after knowing that linkages are attached using drag and drop.

\(^1\)One person did not fill the questionnaire. Quantities refer to the filled questionnaires or observations during the study.
All users managed to attach files to a spot using drag and drop from the file system. Three of the participants accessed the menu first for an adequate entry. 

Except one user, who disagrees, all users agree to agree to strongly agree with the fact that the guidance system of Curator helped to perform the tasks of the study. This user did not see the icons during the tasks.

Some quotations from the questionnaire and study:

- ‘Features in program could be replicated in menu, i.e. “delete linkage” etc.’ (#1)
- ‘Would like to use a button compared to using drag & drop’ (#2)
- ‘I like it, but it needs to tell the user how to turn it back on’ (#3, asked to comment the guidance system)
- ‘Some users might like to rename a spot first, then attach image’ (#4)

4.1.4.5. Other Responses to the Questionnaire

This sections show the reaction of the users to the question what they liked most and what they did not like about Curator.

What did you like most about Curator? This question is to study whether users accept Curator as an easier method to create data-sets for Focus.

- ‘Certainly makes it easy to create Cruiser/Focus displays for the tabletop’ (#4)
- ‘Simplistic’ (#3)
- ‘Dragging and dropping of spots and fine positioning afterwards’ (#2)
- ‘Simplicity, once I knew how to use it’ (#1)
- ‘It’s a simple program, easy to use and to understand. (#6)’

What did you not like about Curator? Test persons may comment any negative points of the application not covered by other questions. They also may highlight any needs they encountered during the tasks.

- ‘(…) I feel everything on the user interface should be replicated in the top menu’ (#1)
- ‘Using drag & drop instead of buttons. Example: for putting attachments into Curator’ (#2)
- ‘Needs consistency, like no drag a main image into program, and no open attachment button to the linkages sidebar’ (#3)
4.1.5 Achievements of the Usability Testing Sessions

Mixed Quotations  The below quotations show needs and feelings of the test persons. The needs point out usability issues as well as future features for *Curator*.

- ‘Was difficult to identify the incorrect attachment [in Task 3]. Had to scroll along the entire file name before I saw “The Rocks”.’ (#1)
- ‘Drag and Drop is an easy way to move a spot.’ (#2)
- ‘How do I specify areas on the map?’ (#3)
- ‘Can I group add spots?’ (#3)
- ‘When I rename a spot again, why is its name not in the text field?’ (#4)

4.1.5. Achievements of the Usability Testing Sessions

The usability testing session to evaluate *Curator* showed that every user was able to create data-sets for *Focus* by himself without major problems. This study helped to answer four questions:

*Is drag and drop easy enough to use for the target group of Curator users?* - Yes, all users did manage to use drag and drop for both creating/moving spots as well as attaching files and liked to use it. However, some test persons hesitated and would prefer to use a button or a menu entry instead of dragging files from the operating system’s file system. The icons of spots could be emphasised so that they visualise drag and drop functionality. These two aspects would also help people not familiar with drag and drop to use the program.

*Is the layout of Curator easily comprehensible to the target group of Curator users?* - Yes, all users agreed that the layout of *Curator*’s main window is good and they did find everything they needed. With renaming the tab ‘Linkages’ to ‘Attachments’ in the English properties files and renaming the menu entry ‘Load image...’ to e.g. ‘New Data-Set...’ would reduce the confusions described above.

*Are the dialogues in Curator precise so that the user understands their consequences?* - Yes, all users were immediately able to change the default target folder of *Curator*. However, the first row of the Options dialogue should be altered so that it helps the user to identify the name of this data-set instead of confusing him. Also, the program should not ask the user to save his data before quitting *Curator* if he saved it just before.

*Is the user happy with the workflow of Curator?* - Yes, all test persons liked the workflow of *Curator* and most of them appreciated the guidance system. However, some users renamed...
spots before attaching files. If the guidance system is enabled, it should react on both events. Therewith, all icons of the guidance system with their labels should be revised again to make them better visible and even clearer. Alternatively, a wizard could replace the existing guidance system.

Other improvements as a result of the study would be:

- Duplicating spots (see Task 4: Create one kidney and copy it to another place)
- When renaming a button again: Fill the text field with the former name
- Offer a ‘save as...’ functionality so that the user can choose a folder and then save his work there.
- Provide a help file that also explains further techniques like attaching multiple files to one or more spots.

Scenario 2 showed that the usability testing participants learned in Scenario 1 how to use the application and then were able to create a data-set without tutorial. With a higher budget and more time to perform the studies, it would have been possible to show the users Focus running on a tabletop and give them the opportunity to explore it themselves which might improve them to understand the meaning of Curator. Moreover, it is likely that the end-user will be introduced into the program by another person.

Figure 4.5 shows a paper prototype of the improved interface of Curator as a result of the usability study.

Figure 4.5.: Improved Curator paper prototype on the basis of the usability testing sessions
4.2. Time Evaluation

Apart from the goal to make it easier to curate data-sets for Focus, Curator’s task is to save time when curating data-sets. This section describes a short comparison when creating data-sets for the application developed in this thesis.

4.2.1. Study description

During Task 2 in all the usability testing sessions described in Section 4.1, time to perform this task was taken without the knowledge of the test persons. This test was done with two IT students as test persons. Both test persons are familiar with the image processing software Gimp and know how to find out pixel co-ordinates in an image. Moreover, they both are used to the concept of XML files.

The shortest time to perform this task using Curator was one minute and ten seconds, the longest three minutes and thirty-five seconds.

Then, two users performed the same task again - but this time they created the data-set manually without Curator. This includes extracting pixel co-ordinates of a spot, manually creating an XML file with all required data in it as well as copying the background image and linkage files.

As a first task, users had to load the ‘Main.jpg’ image into the image processing software Gimp. This free software is able to tell the user pixel co-ordinates on a picture. The test person was asked to note the pixel co-ordinates of a Circular Quay position.

Now, the users created all needed folders and moved the background image as well as the attachments. After this, he had to write the XML file with no spelling mistakes and in the same format as the generated ones. To simplify this process, the test persons were given a printed out sample XML file from a different data-set. Since both parts of this test, creating a data-set using Curator as well as the manual curation, simulate a first time experience, no digital version of the sample XML file was provided. It is crucial that there are no spelling mistakes in this file, so that Focus as well as Curator are able to read this manually created data-set.

4.2.2. Results

The first user finished this task in about ten minutes and ten seconds. However, his XML file was not able to be read by the systems because of a spelling mistake in the XML header. Finding this mistake (a ‘=’ instead of ‘-‘) took additional three minutes. The second participant needed about thirteen minutes; his file was able to be read.

Comparing the times, Curator enables the user to do his work in 11.4 per cent of the time it takes doing it manually. Therewith, Curator is approximately nine times faster.
This time study was based of only one map with one spot. More time per data-set can be saved if there are more spots to create, especially if spots have the same linkages and user attach multiple linkages at the same time to multiple spots.

This study showed without doubt that Curator fulfils the goal of hasten the creation of data-sets.
5. Conclusion

5.1. Summary & Future Work

This thesis set out to achieve two main goals:

- Development of a design environment for curating tabletop museum experiences
- Simplify and hasten the creation of data-sets for the Cruiser/Focus tabletop system.

To achieve these goals, Chapter 2 introduces to existing tabletops in museums and illustrates their functionality. In addition to this, this chapter explains Cruiser, developed by Trent Apter, as well as Focus, a plugin for Cruiser developed by Anthony Collins, both of the University of Sydney. Cruiser is an extensible framework for the rapid and flexible development of immersive tabletop applications. Focus on the other hand, developed as a plugin for Cruiser, is a novel tabletop interface for collaboratively accessing multiple remote file systems. With this file system, it is possible to present tabletop museum exhibitions.

However, it is tedious to create exhibitions with different data or themes, especially for people with basic computer skills. This is why Curator was developed in this thesis. To prepare the reader for Chapter 3 which explains the application, Section 2.4 shows the advantages of the ‘Swing Application Framework’, the framework used to develop the program.

Chapter 3 illustrates the analysis phase of Curator (see Section 3.1) and explains the development (Section 3.2). The Analysis Section forms the basis for the Curator development by identifying the requirements.

After the shape of the application to develop is outlined, Section 3.2 explains the implementation and interface of Curator with regard to the analysis phase.

The application is implemented in Java, supported by the ‘Swing Application Framework’. Java with its Virtual Machine offers implementations for many operating systems and thus fulfils the requirements of portability and flexibility. The Swing Application Framework adds Life-Cycle Management, Resource Management, Tasks and other important advantages. Under the maintainability aspect, it also helps to achieve usability needs by providing the possibility of, for instance, Modeless Feedback.

As described in detail in Section 3.2.2, The Workflow of Curator, Curator is designed with a special regard to a quick and simple workflow of creating data-sets for Focus. The user starts to create a data-set by loading a desired background image, for instance a map. He can then drag info icons, so called spots, onto it. By dragging files into Curator’s sidebar,
they will be attached to this spot. When a museum visitor clicks on this spot on Focus, these attachments will be presented on the tabletop. Curator users can also rename their spots which might help to identify them.

To validate this workflow, Chapter 4, Evaluation of Curator, examined and reviewed test persons of the working student target group using the program. This examination was separated into a usability testing study and an experiment which compares the amount of time it takes when manually curating data-sets and when using Curator.

The usability testing study showed that people appreciate the workflow and layout of the application. Even users who liked a different order of the workflow’s steps, for instance first renaming and then attaching, are free to do so. To introduce first time users to Curator’s workflow, a user guidance system is provided. These icons should enable them to understand the drag and drop functionality. The usability testing study showed that this system is appreciated, but could be made clearer in future work.

The study also proved that drag and drop is a convenient method to position spots as well as to attach files. Nevertheless, to enable users not familiar with drag and drop to work with Curator, implementing menu items or buttons to attach files or links may help to increase usability.

The usability testing study proved that Curator provides an interface which is quick and easy to use, for curating experiences for the Focus system. The question whether the program developed in this thesis also decreases the amount of time required to create data-sets is examined in Section 4.2, Time Evaluation. Compared with manually creating data-sets, which includes finding the co-ordinates of a position, copying attachment files and writing an XML file, Curator is approximately nine times faster.

As for improvements or extensions, creating and managing projects might be extended. At the moment, projects are only logically connected datasets. If the user does not change the target directory in the Options dialogue, sub data-sets will be stored in the same folder as the ‘main’ data-set. To simplify the management of data-sets and projects, the user could be guided when he wants to create a new data-set. He may choose whether this new data-set is connected to another one, the main data-set, and choose this one instead. For instance, a map of Australia is the main data-set. It provides spots, each of them has a map of Australia’s states and territories attached. These states and territories would then be sub data-sets, providing data like images, documents or videos. Moreover, the user could be asked to enter a unique name for the data-set; this could help to keep track of the file and folder structure. This sub data-sets would then be stored in a subfolder of the main data-set.

Appendix C lists further improvements that might help to increase the usability of Curator.

5.2. Conclusion

By its ease of use as well as the short time it takes to curate new data-sets for Focus, Curator opens the door for a widespread use of tabletop systems running Cruiser with Focus.
Since Curator is a Java-based application, it can be run on popular operating systems such as Windows, Linux and Mac OS X. Thus, museums are not bound to a specific system or hardware configuration to create new tabletop experiences which enhance their exhibitions. Also, museums do not need to employ staff specialised in information technologies who are familiar with graphical software and XML files: because of the special regard to the usability, Curator is easy to understand and enables people with basic computer skills to handle work.

It is no more a challenge for exhibition designers, or their staff, to create a virtual exhibition for tabletop systems. Now Cruiser systems can be used in museums without a huge effort if the virtual exhibition on the tabletop must be changed. Exhibition designers are generally not programmers - with Curator they can focus on their exhibition again.

Apart from the museum context, it would be conceivable to provide tabletops in other scenarios. Photo albums of a recent holiday trip could be created with Curator, and relatives and friends could be invited to explore the different visited attractions. This photo album would be more exciting and interactive to the friends than a traditional album and could be enriched with videos and other documents.

Similar to the second scenario in the usability study, learning materials for students to explore could be provided. Teachers could set up contents for the tabletop which enriches their lessons. Companies could allow visitors on an open day to browse through their factory and special attractions they offer. Many other fields of application for Curator data-sets are possible.
A. Images

A.1. Background

Figure A.1.: Netbeans dialogue for a new Swing Application Framework project

A.2. Curator
Figure A.2.: Out-of-the-box SingleFrameApplication
Figure A.3.: Main workflow of Curator, without Options
Figure A.4.: Sequence diagram of a Spot which was clicked without any modifiers. The clicked Spot will be activated. It will also inform the SpotFactory class which forces all other Spots to deactivate and informs all SpotFactoryListeners that a Spot was selected.
B. User Studies

This appendix contains all user study related documents and tasks are presented.
Curator Usability Study - Pilot Tasks

Introduction
You work in a small advertising agency in Europe. Last month, you spent your holidays overseas in Sydney. Due to your long leave, your employer wants you to create a presentation in a creative way to show to your colleagues. You decide to use „Focus“ because it enables you to show pictures to others and also to offer them to have a closer look at the images they like. With Focus it is also possible to navigate through images by selecting spots on a map. Links between maps, maybe to have a closer look at a given area, are also possible. To prepare the presentation, you use the tool „Curator“ for an easy set-up of the presentation.

Task 1: Set-Up
Because it is your first use of the program, you should have a look at the Options first to see which settings are available. You are not pleased with the default directory, so you change the directory where to save the data-set to a folder named „presentation“ on the desktop. You want to save the changes and close the Options dialogue to return to the main window of Curator.

Task 2: Creating a Data-Set
You decide to create a data-set of inner Sydney with a zoomed-in map of Circular Quay. To start open the program Curator. You open the map of inner Sydney named „Main.jpg“ in the folder „data“ on the desktop.

Once the map is loaded, you create a new spot over the area of Circular Quay. Since you have taken a lot of pictures near this area, you want to offer a closer look to this area.

To do so, you want to attach the image „Circular Quay.jpg“ to your marked point. You want to tell the system that this image should be another map. To identify this spot, you give it the more significant name „Circular Quay“.

Now you are finished with this data-set and save your work.

Task 3: Loading and Changing a Data-Set
You want to have a second look at the prepared data-set for „Circular Quay“

Therefore, you load the data-set saved in the„presentation“ folder on the desktop.

You are not satisfied with the position of the spot „Sydney Opera House“ so you want to move it closer to the actual location of the opera.

When you look at this spot’s attachments, you see that there is an undesired image of The Rocks attached to it. You remove this attachment and save your work.
PARTICIPANT INFORMATION STATEMENT
Research Project

Title: User control of tabletop file system and personal information management in pervasive computing environments.

(1) What is the study about?
Curator is a system which allows to compose data-sets for the Cruiser’s “Sidetracks” application. With Curator, it will be possible to link maps, images and more in no time without any programming skills. This user study is about discovering any usability problems which occur during the use of the problem.

(2) Who is carrying out the study?
The study is being conducted by Benjamin Sprengart under the supervision of Judy Kay with the School of Information Technologies.

(3) What does the study involve?
We would like to ask you to participate in a structured interview that is run as single session. The facilitator will take hand-written notes during the session, to make sure that no idea that came up during the session will be lost. At the end of the interview, you will be asked to fill in a questionnaire. There will be no audio or video recording during the session.
(4) How much time will the study take?

This study will take approximately 30 to 60 minutes. The interview is expected to last 20-50 minutes, filling in the questionnaire should not take longer than 10 minutes. The exact time can differ.

(5) Can I withdraw from the study?

Being in this study is completely voluntary - you are not under any obligation to consent and - if you do consent - you can withdraw at any time without affecting your relationship with the researchers or the University of Sydney.

(6) Will anyone else know the results?

All personally identifiable aspects of the study, including results, will be strictly confidential and only the researchers will have access to information on participants. A report of the study may be submitted for publication, but individual participants will not be identifiable in such a report.

(7) Will the study benefit me?

You are making a valuable contribution to our ongoing research.

(8) Can I tell other people about the study?

The study is not confidential and you are free to tell others.

(9) What if I require further information?

When you have read this information, Carolin will discuss it with you further and answer any questions you may have. If you would like to know more at any stage, please feel free to contact Benjamin Sprengart (e-mail: ben@it.usyd.edu.au), or Professor Judy Kay (e-mail: judy@it.usyd.edu.au, ph: 02 9351 4502).

(10) What if I have a complaint or concerns?

Any person with concerns or complaints about the conduct of a research study can contact the Manager, Ethics Administration, University of Sydney on (02) 8627 8175 (Telephone); (02) 8627 8180 (Facsimile) or g briody@usyd.edu.au (Email).

This information sheet is for you to keep
PARTICIPANT CONSENT FORM

I, ....................................................................................., give consent to my participation in the research project

Name (please print)

Curator: Design Environment for Curating Tabletop Museum Experiences

The following points have been explained to me:

1. The procedures required for the project and the time involved have been explained to me, and any questions I have about the project have been answered to my satisfaction.

2. I have read the Participant Information Sheet and have been given the opportunity to discuss the information and my involvement in the project with the researcher(s).

3. The experiment will consist of the following procedure: I will be asked to familiarise myself with a computer. Before the experiment starts, I will be asked to think aloud. I will then be asked to perform tasks for the study step by step, which includes handling of Curator as well as creating and loading data-sets with it.

4. The researchers do not foresee any risks to me as a result of participating in this study, nor do they expect that I will experience any discomfort or stress.

5. I understand that I can withdraw from the study at any time, without affecting my relationship with the researcher(s) now or in the future.

6. I understand that my involvement is strictly confidential and no information about me will be used in any way that reveals my identity.

Signed: ........................................................................................................................................

Name: ........................................................................................................................................

Date: ........................................................................................................................................
Curator Usability Study - Tasks

Scenario 1

Introduction
You work in a small advertising agency in Europe. Last month, you spent your holidays overseas in Sydney. Due to your long leave, your employer wants you to create a presentation in a creative way to show to your colleagues.

You decide to use „Focus“ because it enables you to show pictures to others and also to offer them to have a closer look at the images they like. With Focus it is also possible to navigate through images by selecting spots on a map. Links between maps, maybe to have a closer look at a given area, are also possible.

To prepare the presentation, you use the tool „Curator“ for an easy set-up of the presentation.

Task 1: Set-Up
You want to open the Curator application, it is located on the Desktop, named “Curator.jar“. Because it is your first use of the program, you should have a look at the Options first to see which settings are available. You are not pleased with the default directory, so you change the directory where to save the data-set to a folder named „presentation“ on the desktop.

You want to save the changes and close the Options dialogue to return to the main window of Curator.

Task 2: Creating a Data-Set
You decide to create a data-set of inner Sydney with a zoomed-in map of Circular Quay.

To start, open the program Curator if it is not open.

You open the map of inner Sydney named „Main.jpg“ in the folder „data“ on the desktop. Since you have taken a lot of pictures near this area, you want to offer a closer look to this area. That is why you want to create a new spot over the area of Circular Quay.

Now, you want to attach the image „Circular Quay.jpg“ to your marked point. You want to tell the system that this image should be another map.

To identify this spot, you give it the more significant name „Circular Quay“.

Now you are finished with this data-set and save your work.
**Task 3: Loading and Changing a Data-Set**

You want to have a second look at the prepared data-set for „Circular Quay“. Therefore, you load the data-set saved in the „presentation“ folder on the desktop. Data-set files end with „.xml“.

You are not satisfied with the position of the spot „Sydney Opera House“ so you want to move it closer to the actual location of the opera.

When you look at this spot’s attachments, you see that there is an undesired image of The Rocks attached to it. You remove this attachment and attach an image of the Sydney Opera House you found in the “data” folder instead.

When you are finished, you save your work and quit the program.

---

**Scenario 2**

**Introduction**

You are a working student in the Australian Museum. Your supervisor, an exhibition designer, wants you to enrich his latest exhibition “The Human Body” using the new tabletop display. To do so, he provides you with images and documents. He wants you to use the tool “Curator” to make the data ready for the tabletop. Because it’s been a while since your last biology lessons, he shows you the locations of the organs in the human body.

**Task 4: Creating a New Data-Set**

Create a new data-set for the “The Human Body” exhibition according to the image you have seen before. To do so, you want to store your new data-set in a folder on the Desktop named “BodyExhibition”.

You will find the data you need for your work in the folder “Human Body” on the Desktop. Choose “BodyOverview.jpg” as a background image for your data-set.

Now create a spot for each organ, distinguish between left and right kidney. Rename the spots so that they match the name of the organ. Attach all suiting files you can find in the folder your supervisor gave you to their spots.

When you are finished, you save your work.
# Curator – Questionnaire

1) I was able to perform the requested tasks without any problems.

<table>
<thead>
<tr>
<th>strongly disagree</th>
<th>disagree</th>
<th>somewhat disagree</th>
<th>neutral</th>
<th>somewhat agree</th>
<th>agree</th>
<th>strongly agree</th>
</tr>
</thead>
<tbody>
<tr>
<td>☐</td>
<td>☐</td>
<td>☐</td>
<td>☐</td>
<td>☐</td>
<td>☐</td>
<td>☐</td>
</tr>
</tbody>
</table>

Please explain your decision:

_____________________________________________________________________________
_____________________________________________________________________________
_____________________________________________________________________________

2) I liked the overall layout of Curator.

<table>
<thead>
<tr>
<th>strongly disagree</th>
<th>disagree</th>
<th>somewhat disagree</th>
<th>neutral</th>
<th>somewhat agree</th>
<th>agree</th>
<th>strongly agree</th>
</tr>
</thead>
<tbody>
<tr>
<td>☐</td>
<td>☐</td>
<td>☐</td>
<td>☐</td>
<td>☐</td>
<td>☐</td>
<td>☐</td>
</tr>
</tbody>
</table>

Please describe any issues you had in detail:

_____________________________________________________________________________
_____________________________________________________________________________
_____________________________________________________________________________

3) Task 1: The Options dialogue was easy to understand and I knew what to change:

<table>
<thead>
<tr>
<th>strongly disagree</th>
<th>disagree</th>
<th>somewhat disagree</th>
<th>neutral</th>
<th>somewhat agree</th>
<th>agree</th>
<th>strongly agree</th>
</tr>
</thead>
<tbody>
<tr>
<td>☐</td>
<td>☐</td>
<td>☐</td>
<td>☐</td>
<td>☐</td>
<td>☐</td>
<td>☐</td>
</tr>
</tbody>
</table>

Please describe any issues you had in detail:

_____________________________________________________________________________
_____________________________________________________________________________
_____________________________________________________________________________
4) Task 2: I liked the workflow of creating a data-set based on only one map:

<table>
<thead>
<tr>
<th>Strongly Disagree</th>
<th>Disagree</th>
<th>Somewhat Disagree</th>
<th>Neutral</th>
<th>Somewhat Agree</th>
<th>Agree</th>
<th>Strongly Agree</th>
</tr>
</thead>
<tbody>
<tr>
<td>[ ]</td>
<td>[ ]</td>
<td>[ ]</td>
<td>[ ]</td>
<td>[ ]</td>
<td>[ ]</td>
<td>[ ]</td>
</tr>
</tbody>
</table>

Please describe any difficulties you may have had in detail:
_____________________________________________________________________________
_____________________________________________________________________________
_____________________________________________________________________________

5) It was easy and intuitive to me to use drag and drop while positioning spots.

<table>
<thead>
<tr>
<th>Strongly Disagree</th>
<th>Disagree</th>
<th>Somewhat Disagree</th>
<th>Neutral</th>
<th>Somewhat Agree</th>
<th>Agree</th>
<th>Strongly Agree</th>
</tr>
</thead>
<tbody>
<tr>
<td>[ ]</td>
<td>[ ]</td>
<td>[ ]</td>
<td>[ ]</td>
<td>[ ]</td>
<td>[ ]</td>
<td>[ ]</td>
</tr>
</tbody>
</table>

Please describe any difficulties you may have had in detail:
_____________________________________________________________________________
_____________________________________________________________________________
_____________________________________________________________________________

6) I had no trouble with Task 3: Loading and Changing a Data-Set.

<table>
<thead>
<tr>
<th>Strongly Disagree</th>
<th>Disagree</th>
<th>Somewhat Disagree</th>
<th>Neutral</th>
<th>Somewhat Agree</th>
<th>Agree</th>
<th>Strongly Agree</th>
</tr>
</thead>
<tbody>
<tr>
<td>[ ]</td>
<td>[ ]</td>
<td>[ ]</td>
<td>[ ]</td>
<td>[ ]</td>
<td>[ ]</td>
<td>[ ]</td>
</tr>
</tbody>
</table>

Please describe any difficulties you may have had in detail:
_____________________________________________________________________________
_____________________________________________________________________________
_____________________________________________________________________________
7) The guidance system of *Curator* helped me to perform the tasks.

<table>
<thead>
<tr>
<th>strongly disagree</th>
<th>disagree</th>
<th>somewhat disagree</th>
<th>neutral</th>
<th>somewhat agree</th>
<th>agree</th>
<th>strongly agree</th>
</tr>
</thead>
<tbody>
<tr>
<td>☐</td>
<td>☐</td>
<td>☐</td>
<td>☐</td>
<td>☐</td>
<td>☐</td>
<td>☐</td>
</tr>
</tbody>
</table>

Please describe any difficulties you may have had in detail:

_____________________________________________________________________________
_____________________________________________________________________________
_____________________________________________________________________________

8) The dialogues behaved like indicated and helped me to use the program?

<table>
<thead>
<tr>
<th>strongly disagree</th>
<th>disagree</th>
<th>somewhat disagree</th>
<th>neutral</th>
<th>somewhat agree</th>
<th>agree</th>
<th>strongly agree</th>
</tr>
</thead>
<tbody>
<tr>
<td>☐</td>
<td>☐</td>
<td>☐</td>
<td>☐</td>
<td>☐</td>
<td>☐</td>
<td>☐</td>
</tr>
</tbody>
</table>

Please describe any difficulties you may have had in detail:

_____________________________________________________________________________
_____________________________________________________________________________
_____________________________________________________________________________

9) What did you like most about Curator?

_____________________________________________________________________________
_____________________________________________________________________________
_____________________________________________________________________________

10) What did you not like about Curator?

_____________________________________________________________________________
_____________________________________________________________________________
_____________________________________________________________________________
11) If you were a designer or programmer of Curator, what would you like to change?
_____________________________________________________________________________
_____________________________________________________________________________
_____________________________________________________________________________

12) Do you have any other comments?
_____________________________________________________________________________
_____________________________________________________________________________
_____________________________________________________________________________

*Thank you very much for having taken your time to participate this usability study!*
C. Possible Extensions

This section names and describes further improvements of Curator which came up during the development phase of the program.

**Switching between data-sets**  When switching data-sets in the current version of Curator, the active data-set is saved and unloaded and the desired data-set is loaded. The user has no influence on saving his old data-set. The Options dialogue might provide a setting which that asks the user what to do. Possible settings might be: always save, ask what to do, never save. It would be preferable to have a functionality which allows the user to undo his last steps.

**Tabs for each data-set**  Instead of switching between data-sets using a JComboBox, tabs could be used to display each data-set. The advantage is that multiple data-sets could be accessible at the same time in one instance of the program and switching could be done without saving the progress.

**Miscellaneous Improvements**  The following functions may help to improve the application:

- Context menu for attachments: creating a new data-set based on an attachment, instead of using the menu bar’s entry and the NewMapDialogue.

- Possibility to link attachments using the menu and a Open-File dialogue in addition to dragging and dropping files in the implemented way. See Section 4.1.4 for details.

- Recognise when the user tries to load an image which has already been used as basis for a data-set. Program could ask the user if he wants to load this data-set instead.

- Wizard to create a new data-set. This could help first time users to become familiar with the application.

- Help file which explains Curator’s functionality in detail.
List of Figures

2.1. Overhead projector tabletop construction ........................................ 5
2.2. Rear-projection tabletop construction .............................................. 7
2.3. Dangerous Australians tabletop ...................................................... 9
2.4. Dangerous Australians: Information on a Blue Ringed Octopus ............... 9
2.5. Hardware of Dangerous Australians .................................................. 10
2.6. Dialog Table .................................................................................. 11
2.7. Churchill Lifeline .......................................................................... 11
2.8. etx ........................................................................................... 12
2.9. floating.numbers .......................................................................... 13
2.10. Black Hole sucking in an image ..................................................... 14
2.11. Frame creating a screen shot .......................................................... 15
2.12. Storage Bin .................................................................................. 15
2.13. Copier ....................................................................................... 16
2.14. Focus with Sydney data-set ............................................................ 17
2.15. JSR-296 architecture (©java.net) ................................................... 18
2.16. Swing Application Framework life cycle ......................................... 20
2.17. Example: ResourceBundle .............................................................. 20
2.18. Example: Folder structure with ResourceBundle .............................. 21
2.19. Example: @Action ..................................................................... 23
2.20. Example: Task notification ............................................................. 25
2.21. Example: Persistant session state ................................................... 25

3.1. User story cards ............................................................................. 27
3.2. Primary persona of Curator ............................................................. 31
3.3. Secondary persona of Curator .......................................................... 31
3.4. First Curator paper prototype .......................................................... 32
3.5. Second Curator paper prototype ....................................................... 33
3.6. Third Curator paper prototype ........................................................ 33
3.7. Workflow of Curator .................................................................... 35
3.8. Use Cases of Curator .................................................................... 38
3.9. Structure and basic interaction of the bachelorarbeit.spot package .... 39
3.10. User interface related classes .......................................................... 46
3.11. The Curator application and its component parts ............................ 48
3.12. Sidebar of Curator ...................................................................... 48
3.13. Curator’s dialogue to create a map from an attachment .................... 52
3.14. The Options Frame ..................................................................... 53
3.15. Displayed information if multiple Spots are selected (left) and only one Spot is selected (right) .......................................................... 54
<table>
<thead>
<tr>
<th>Figure</th>
<th>Description</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>3.16</td>
<td>Dialoge to confirm deletion of <em>Spots</em></td>
<td>55</td>
</tr>
<tr>
<td>3.17</td>
<td>The <em>LinkagePanel</em></td>
<td>56</td>
</tr>
<tr>
<td>3.18</td>
<td>Multiple Selection of <em>Spots</em></td>
<td>57</td>
</tr>
<tr>
<td>3.19</td>
<td>Other Components of <em>Curator</em></td>
<td>58</td>
</tr>
<tr>
<td>3.20</td>
<td>Sample file structure of a <em>Curator</em> project with two data-sets</td>
<td>62</td>
</tr>
<tr>
<td>3.21</td>
<td>Example of key/value pairs for internationalisation</td>
<td>64</td>
</tr>
<tr>
<td>3.22</td>
<td>Label to indicate that a spot can be dragged and dropped</td>
<td>65</td>
</tr>
<tr>
<td>3.23</td>
<td>Label to show drag and drop functionality to attach files or URLs</td>
<td>66</td>
</tr>
<tr>
<td>3.24</td>
<td>Label to indicate how to rename a spot</td>
<td>66</td>
</tr>
<tr>
<td>3.25</td>
<td><em>Curator</em> with its sidebar, detached</td>
<td>67</td>
</tr>
<tr>
<td>3.26</td>
<td>The <em>ImagePreviewPanel</em> used in an open file dialog</td>
<td>68</td>
</tr>
<tr>
<td>4.1</td>
<td>DECIDE as framework for Usability Testing</td>
<td>69</td>
</tr>
<tr>
<td>4.2</td>
<td>Files for the first scenario. The desired output of this scenario is shown on the right-hand side. Maps: © Google</td>
<td>72</td>
</tr>
<tr>
<td>4.4</td>
<td>Performing the usability testing study</td>
<td>74</td>
</tr>
<tr>
<td>4.5</td>
<td>Improved <em>Curator</em> paper prototype on the basis of the usability testing sessions</td>
<td>79</td>
</tr>
<tr>
<td>4.6</td>
<td>Comparison of times building a data-set manually and using <em>Curator</em></td>
<td>81</td>
</tr>
<tr>
<td>A.1</td>
<td>Netbeans dialogue for a new Swing Application Framework project</td>
<td>87</td>
</tr>
<tr>
<td>A.2</td>
<td>Out-of-the-box <em>SingleFrameApplication</em></td>
<td>88</td>
</tr>
<tr>
<td>A.3</td>
<td>Main workflow of <em>Curator</em>, without Options</td>
<td>89</td>
</tr>
<tr>
<td>A.4</td>
<td>Sequence diagram: Click on a <em>Spot</em></td>
<td>90</td>
</tr>
</tbody>
</table>
List of Tables

2.1. Construction-specific advantages and disadvantages of overhead projector tablespops with vision technology .................................................. 6
2.2. Construction-Specific advantages and disadvantages of rear-projection tabletops 7
2.3. Relationships among class file, ResourceBundle name, and ResourceBundle file name ................................................................. 22

3.1. Functionality requirements .............................................................. 28
3.2. Quality requirements for Curator .................................................... 29
3.3. Content of bachelorarbeit.spot .......................................................... 40
3.4. The Enum Event ............................................................................ 41
3.5. Methods a SpotListener must provide ............................................ 44
3.6. Methods a SpotFactoryListener must provide ............................... 44
3.7. User interface related classes .......................................................... 47
3.8. Overview of Parts of Curator .......................................................... 49
3.9. Other Components of Curator ......................................................... 58
Listings

2.1. Manual access to a properties file using a \texttt{ResourceMap} \hspace{1cm} 21
2.2. Example of a Field Resource Injection \hspace{1cm} 22
2.3. Example: Using the \texttt{@Action} Annotation for Associating Actions \hspace{1cm} 23

3.1. XML File created by \textit{Curator} \hspace{1cm} 60
3.2. Example of accessing a language properties file \hspace{1cm} 64
3.3. Using Apple’s menu bar on Mac OS X \hspace{1cm} 67
Nomenclature

API Application Programming Interface, page 18
EDT Event Dispatch Thread, page 19
FTIR Frustrated Total Internal Reflection, page 7
GUI Graphical User Interface, page 18
GUI Graphical User Interface, page 71
IDE Integrated Development Environment, page 18
IT Information Technology, page 74
JSR Java Specification Request, page 18
JVM Java Virtual Machine, page 21
OS Operating System, page 33
POJO Plain Old Java Object, page 36
SAF Swing Application Framework, page 17
UI User Interface, page 19
URI Uniform Resource Identifier, page 41
URL Uniform Resource Locator, page 27
VM Virtual Machine, page 33
References


