LMU LUDWIG-MAXIMILIANS-UNIVERSITÄT MÜNCHEN

INSTITUT FÜR INFORMATIK
LEHR- UND FORSCHUNGSEINHEIT FÜR
PROGRAMMIER- UND MODELLIERUNGSSPRACHEN

Institut für Informatik

BIBPAD

DIGITAL ANNOTATION OF PHYSICAL MEDIA

SEBASTIAN MADER

Bachelorarbeit

Betreuer	Prof. Dr. François Bry
	Christoph Wieser
Beginn am	23.05.2013
Abgabe am	23.08.2013

DECLARATION

Hiermit versichere ich, dass ich diese Bachelorarbeit selbständig verfasst habe. Ich habe dazu keine anderen als die angegebenen Quellen und Hilfsmittel verwendet.

München, den 23. August 2013

Sebastian Mader

ABSTRACT

The Bayerische Staatsbibliothek possesses a large inventory of media about World War I. At this point these media are only searchable by metadata which makes it hard to formulate selective queries. Goal of the Bayerische Staatsbibliothek is to develop a semantic search engine for these media. The problem that is to solve is the acquisition of data for fueling this search engine.

This thesis tries to solve that problem by proposing a Web platform that allows users to create annotations about said media. These annotations contain information about the annotated media and can consequently be used to fuel the semantic search engine.

Before this is possible two problems have to be solved: How to represent the different kinds of physical annotations digitally in the best possible way? How to get the user to create the formal annotations needed to fuel the semantic search engine instead of annotations in natural language?

In this thesis approaches for both problems are developed and based on this approaches the concept of the Web annotation platform BibPad is outlined.

ZUSAMMENFASSUNG

Die Bayerische Staatsbibliothek verfügt über einen großen Bestand an Medien, die den ersten Weltkrieg thematisieren. Zum jetzigen Zeitpunkt sind diese Medien nur anhand ihrer Metadaten durchsuchbar, was es schwer macht gezielte Suchanfragen zu stellen. Ziel der Bayrischen Staatsbibliothek ist es deshalb eine semantische Suchmaschine für diese Bücher zu entwickeln. Das Problem, das hier noch besteht, ist die Beschaffung der Daten, welche die Grundlage für die Suchmaschine bieten sollen.

Die vorliegende Arbeit versucht dieses Problem zu lösen, indem sie eine Webplattform vorschlägt, die es erlaubt den Benutzern Annotationen zu Medien zu erstellen. Diese Annotationen enthalten Information über das Medium und sollen als Grundlage für die semantische Suchmaschine genutzt werden.

Bevor das allerdings möglich ist, müssen einige Probleme gelöst werden: Wie repräsentiert man die verschiedenen Arten von physischer Annotation bestmöglichst digital? Wie bringt man den Benutzer dazu, anstatt natürlichsprachenlichen Annotationen auch formale Annotationen, die nötig sind, um die Suchmaschine anzutreiben, anzulegen?

Diese Arbeit entwickelt Lösungsansätze für diese beiden Probleme vor und schlägt, aufbauend auf diese, eine Implementierung für eine Webplattform zum Erstellen von Annotationen, genannt BibPad, vor.

ACKNOWLEDGMENTS

First and foremost, I want to thank my advisor Christoph Wieser. Without his constant support and advice the thesis would not be on the current level.

Also, I want to thank Prof. Dr. François Bry for giving me the chance of working on such an interesting topic and offering me a workplace at the research unit.

I also want to express my gratitude to all members of the Teaching and Research Unit for Programming and Modelling Languages for their warm welcome.

Thanks also to my family for always supporting me in my studies.

Lastly, I want to express my sincere gratitude to everyone who was involved in this thesis even if not mentioned here.

CONTENTS

1 INTRODUCTION

Everybody uses annotations. They are an important part when working with scientific literature. Shabajee and Reynolds propose the following definition for annotation:

> "A note added to anything written, by way of explanation or comment."
>
> — Shabajee and Reynolds [17]

In other words: An annotation is a remark on a document that either explains or comments on the surrounding passages. This definition is rather restrictive: It limits the target of an annotation to written text, while the target also could be an image or a map. A note is usually written text, so the definition limits annotations to textual remarks, while there are other possible forms such as a graphical mark. And lastly, the function is limited to explanation and comment, while there are many other functions such as highlighting or summarizing.

This restrictive definition is not really fitting for the needs of this thesis, therefore, the following new definition is used for the remainder of the thesis.

> An annotation is a mark added to a medium or parts of it that is free in form and function.

Annotating directly on the medium is not an issue as long as the annotator owns the medium – it becomes an issue if the medium is not the annotator's property, such as books from a library. In this case, the annotator has to separate the annotation from the medium and needs paper, pen, and a system for managing his annotations. It is no longer possible to place the annotation exactly where the annotator wants it to be; the position gets coarse. It is no longer possible to mark words, sentences, or paragraphs. For a more accurate digital representation of physical annotation a digital version of the medium would have to be available – and in a perfect world such a version would exist.

But – this world is not perfect. Even though digitalization is an ongoing process, certain circumstances such as copyright and the sheer amount of publishing houses keep libraries from going all digital.

While there is ongoing research in the area of digital annotation of digital media, the research in the area of digital annotation of physical media is nearly non-existent. Existing research in this area focuses on approaches that use external hardware, such as the Annoto Pen – a pen equipped with a camera.

This thesis proposes an approach for the digital annotation of physical media that uses external hardware most of the annotators already own: smartphones, tablets, and laptops. The idea is to develop a Web platform that allows the creation of annotations for physical media and also adds social features, such as sharing media and collaborating on annotations.

Two main motivations exist for research in this area: The first is to develop something that does not exist in the proposed form, and the second is the prospect of scientific utilization of the gathered annotations.

First things first: At the moment there is no ongoing research in the area of digital annotation of physical media without the need of external hardware. Strictly speaking, smartphones, tablets, and laptops are external hardware too, but given that most users own at least one of those devices, this fact is negligible.

The gathered annotations are a valuable source of information about the annotated medium. The annotation is available for digital processing - so why not take advantage of this and use the information of the annotation?

The Bayerische Staatsbibliothek owns a large collection of media about World War I, and they want an ontology-based search engine that makes said media easily search- and filterable. The idea is to use the information in the annotations about these media to fuel the semantic search engine. This approach also eliminates the need to pay experts for creating data for the search engine.

TOPIC OF THIS THESIS

This thesis develops a concept for a Web platform that allows the digital annotation of physical media – BibPad. BibPad is a Web platform which allows users to create, share, and collaborate on annotations. The collected annotations are used to create the knowledge base for the semantic search engine.

For this to work, this thesis has to solve two problems: The first is to identify the most-used forms of physical annotations and find an appropriate projection of these forms into the digital world. This also includes the problem of the position of the annotation - a perfect representation is not possible, thus, a compromise has to be reached.

The second problem is knowledge gathering: User-created annotations are inherently informal, so ways and approaches for gathering formal – machine-usable – knowledge from users have to be developed. The developed approaches should yield advantages for the user, so that he actually uses them. If there is no advantage for the user, he won't use them.

CONTENT OVERVIEW

The first part discusses related work - it introduces Catherine Marshall's pioneering work in the field of digital annotation and two projects in the area of digital annotation of physical media using external hardware.

In order to develop a system that allows the physical annotation of digital media physical, annotations themselves have to be studied. That is the content of Chapter 3. It introduces various distinctions for physical annotations and proposes projections for them into the digital world.

Building on the results of Chapter 3, the next Chapter introduces various ways to gather knowledge in the different kinds of annotations. But also other ways for knowledge gathering beyond annotations, such as concept maps, are considered. The representation of the gathered knowledge is also part of this chapter.

Chapter 5 combines the theoretical concepts introduced in the two previous chapters and discusses the actual implementation of BibPad, and the technology used in the implementation.

The last chapter summarizes the findings of the thesis, highlights topics that could not be covered in the thesis, and takes a look ahead into the future.

2 RELATED WORK

Annotations and their digital representation are a broad field of research, but as already mentioned, the focus of this thesis is on digital annotation of digital media. Research in the the area of digital annotation of physical media is scarce, and the introduced approaches are mostly reliant on external hardware, such as a special desk with beamer and camera.

This chapter presents research in both areas starting with Catherine Marshall's pioneering work in the field of digital annotations.

FIRST STEPS – CATHERINE MARSHALL

Marshall [12] started research in the area of digital annotation of digital media in 1997. In her study she examined annotations students leave in their textbooks. In the USA students are usually required to buy their textbooks – the bookstores offer to buy back books after the semester even when they contain annotations. Marshall took annotated books from different subjects and examined the annotations found in these books.

Her study identified various characteristics of annotations. The annotations found in the examined books were either telegraphic, like underlined sentences or brackets on the margins of the page, or explicit, such as brief notes between lines or in the margins.

Marshall concluded that it is nearly impossible to represent the complete range of physicial annotations with digital annotations and that it is very difficult to match the seamlessness of physical annotating in a digital annotation system.

DIGITAL ANNOTATION OF DIGITAL MEDIA

MADCOW [7] is an acronym for Multimedia Annotation of Digital Content over the Web. This system allows the user to annotate objects on a website, like text, hyperlinks, images, videos, or audio. Annotations in MADCOW are classified by function, like comment, explanation, or summary. An annotation is either public or private; public means that the annotation is stored on an annotation server, while a private annotation is stored locally on the users' computer.

A client-server-architecture with independent servers is utilised by MADCOW, that is a user can subscribe to the servers which store the kind of annotations he wants to be displayed on websites. The client is implemented as a plugin for Microsoft Internet Explorer which

displays the public annotations on websites and gives the user the option to create his own annotations.

DIGITAL ANNOTATION OF PHYSICAL MEDIA

MADCOW required one thing: digital availability of the medium. If the digital availability of the medium can be guaranteed, this approach is preferable – but for a number of reasons such as copyright the digital availablity can not be guaranteed in a library.

Decurtins et al. [10] are proposing the usage of external hardware such as the Anoto Live digital pen. This pen is a ballpoint pen enhanced with a digital camera, an image microprocessor, and means for wireless communication. The documents have to be printed on paper with a special pattern, so that the pen is able to pinpoint its exact position on the paper [3].

Usually annotations belong to parts of the document – their context. Decurtins et al. chose x- & y-coordinates provided by the Anoto pen as context. If the user moves the pen into a part of the document that contains an annotation, the annotation is displayed on a display. The pen enables the perfect positioning of digital annotations but has severe drawbacks: The Anoto technology captures the written annotation not through movements of the pen rather than by capturing the actual writings of the user. Therefore, Anoto can only be used when one is the owner of the annotated media and, thus, impractical for usage in libraries.

Wu et al.'s [27] approach WikiTUI requires a special workplace for annotating physicial media. A camera and a projector are fixed above the workplace. The projector projects the annotations directly onto the book while the camera captures user interaction, like the positioning of annotations. For adding annotations another device, such as a tablet or a laptop, is needed. The user creates the annotation – the annotation is shown somewhere on the workplace – and drags it with his fingers to the desired position.

WikiTUI has drawbacks: It needs even more external hardware than the approach suggested by Decurtins et al., and the hardware is bound to a desk – the user can not work where he wants to, and the desks are expensive, and thus, their availability is limited.

WRAPPING UP RELATED WORK

Previous research in the field of digitial annotation of physical media introduced approaches with external hardware, but there is no ongoing research for annotating without external hardware. This thesis proposes a Web annotation platform accessible from a wide range of devices for annotating physical media digitally. Strictly speaking, smartphones, tablets, and laptops are also external hardware. How-

ever, because the majority of potential users own at least one of the devices – unlike an Annoto pen or a whole workplace with camera and beamer – this thesis does not count those devices as external hardware.

The next chapter discusses the different classes of physical annotations and their digital representations.

3 ANNOTATIONS

Annotations are an important part of working with media with studies suggesting that up to 50% of the time working with media is writing, like taking notes [1].

Goal of this first part of the thesis is to find a solution to the first problem of this thesis: finding a digital representation for physical annotations. This representation should be as close as possible to its physical counterpart. In order to achieve this a thorough examination of different kinds of annotations is required. The results of this examination are presented in the first part of this chapter: It introduces different kinds of annotations used for the remainder of the thesis.

Building on the results of the previous part, the next section proposes digital representations for physical annotations. The advantages and drawbacks of this representation over physical annotations are discussed in the last part of this chapter.

3.1 PHYSICAL ANNOTATIONS

This part of the thesis examines the physical annotation itself. The platform should account for most kinds of annotations that user usually use when annotating media. In order to achieve this, first the different classes of physical annotations are identified.

The classification presented in this part provides the foundation for the remainder of the thesis.

3.1.1 *Context of an Annotation*

Before talking about classifications for physical annotations, one important term has to be introduced: Context. Every annotation has a context – the context of an annotation is the part of the medium the annotation refers to, such as a passage of a book.

With physical annotation the context can be as fine as the annotator wishes it to be: the whole book, a single chapter, a passage, or even a word in the document. The context is often indicated by the position of the annotation – an annotation referring to a certain passage is usually placed in the margin next to the passage; an annotation about a chapter before or after the chapter. It can also assume other forms like an asterisk linking a word and an annotation or an arrow between a passage and an annotation.

3.1.2 *Classification of Annotations*

Physical annotations are a vast area of research: Studies examine how note-taking in lectures boosts exam performance [6], if there is a difference between notes from annotators who read a formal and annotators who read an informal version of a document [9], and lastly many studies suggesting different classifications for annotations.

In most cases, these classifications are either by form or by function. In this thesis classification by form was chosen. Why so? All kinds of annotations put together form one set, the different kinds of classification divide this set into smaller subsets. Those subsets are not necessarily disjoint; overlapping is possible.

Goal of this thesis is to find a digital representation for the different classes of annotations, but different persons do not necessarily use the same form for the same function – so it could happen that a person is forced to use a form he does not associate with this function.

With classification by form that is not an issue: For a certain function the user chooses the form with which he wants to express this certain function.

Formal versus Informal

The first distinction is between formal and informal annotations. This distinction is introduced by Catherine Marshall [13].

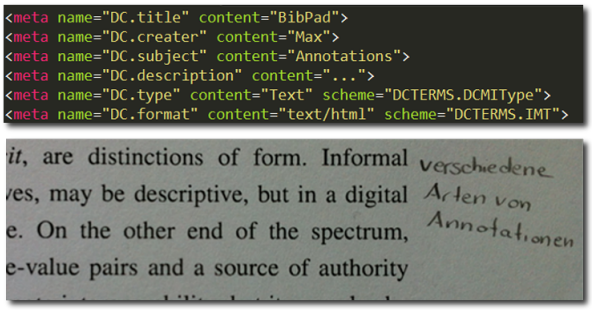

Figure 1: Examples for formal and informal annotations.

A formal annotation is an annotation which follows a formal structure, such as meta tags in an HTML document. Formal annotations are easily machine-readable and can be made machine-understandable if a vocabulary is provided. Vocabularies are discussed in detail later in Chapter 4.

The image on the left side in Figure 1 shows an example for a formal annotation – meta tags in an HTML document. The vocabulary are the values of the *name* attribute, for example *DC.title* or *DC.creator*. Together with the value of the attribute *content* a machine is able to

determine that this document is titled *BibPad* and created by *Max*. The underlying vocabulary is Dublin Core[1] as hinted by the prefix *DC*.

On the contrary, there are informal annotations: This kind of annotation is completely free in form and style and mostly used by humans when annotating media. An example for an informal annotation is the image on the right side in Figure 1. In this case, it is a textual remark in the margins of the document but could also be something like an arrow or an exclamation mark.

If an informal annotation is textual, then it is in most cases written in natural language. While a machine can read such an annotation, it is nearly impossible for a machine to understand the content of the annotation.

For fueling a semantic search engine formal annotations are required, but the users of the platform create informal annotations. Solutions for this problem are discussed in detail in Chapter 4.

Implicit versus Explicit

Another distinction proposed by Catherine Marshall [13] is by comprehensibility – a distinction between implicit and explicit annotations.

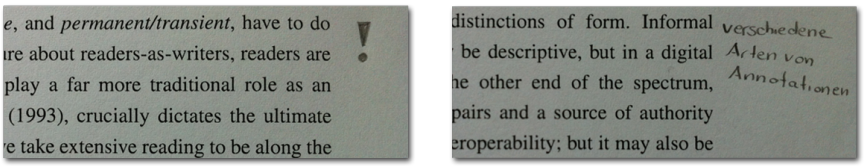

Figure 2: Examples for explicit and implicit annotations.

An implicit annotation could be ambiguous or even unintelligible for somebody who is not the author of the annotation. The meaning depends on the thoughts and the situation of the annotator. The image on the right side of Figure 2 shows an implicit annotation. One who is not the author of the annotation can only guess what the exclamation mark expresses: An important passage? A mistake in this passage? The need to read this passage again?

The opposite of an implicit annotation is an explicit annotation. This kind of annotation is completely understandable for non-authors without having knowledge about situation and thoughts the annotator had when creating the annotation. On the left side of Figure 2 an example for such an annotation is given: It is pretty clear that the passage next to the annotation is about classes of annotations.

Users create both kind of annotations, but our main interest are explicit annotations. While not part of this thesis, there is ongoing research in the field of information extraction, that is creating formal

1 http://dublincore.org/

annotations from informal annotations. For this to work the annotations have to be explicit.

Textual versus Graphic versus Referencing

Another distinction is proposed by Agosti et al. [2] – a distinction by the actual sign, the physical manifestation on paper: textual, graphical, and referencing annotations.

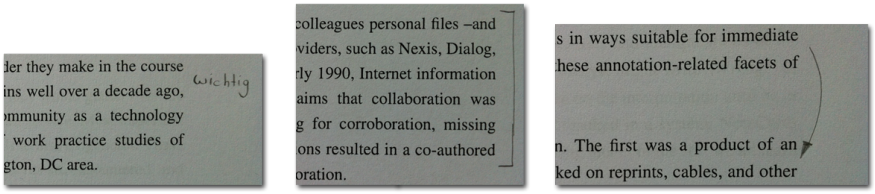

Figure 3: Examples for textual, graphical, and referencing annotations.

The easiest form of annotation is a textual annotation: It is an annotation that consists of "text that is added to a document or parts of it" [2].

A graphical annotation is a "graphic mark added to the document or parts of it" [2], like an exclamation mark, brackets around parts of the text, or passages highlighted with colour.

With a referencing annotation the annotator expresses some kind of relation between two passages, documents, or chapters. A referencing annotation can take many forms ranging from arrows connecting two passages to marking two passages with an asterisk indicating their connection.

One could argue that a referencing annotation is not a classification by form rather than a classification by function, and a referencing annotation is simply a graphical or even a textual annotation. The difference is the context: While graphical and textual annotations only have a single context, a referencing annotation connects two contexts, that is a referencing annotation has two contexts. This difference is major and should be accounted for, hence, the distinction.

Figure 3 shows examples for the three signs of annotations. From left to right: The textual annotation is a small remark in the margins of the document, in this case, indicating that something in the passage next to it is important. The bracket in the middle picture is a graphical annotation indicating a connection between the enclosed lines. The last image shows a referencing annotation: The arrow between the two passages implies some kind of relation between them.

An annotation does not strictly belong in one of the three categories: One could write a text next to a bracket enclosing some lines, one could label the arrow connecting two passages, or highlight words of a textual annotation creating some kind of meta annotation.

One thing these three annotations have in common is that they are in the majority of cases informal. They can be either implicit or explicit with graphical annotations being mostly implicit, because a certain symbol is not connected with a certain meaning.

These three signs of annotations are sufficient to express the majority of the functions a person wants to express.

3.1.3 *Wrapping Up Physical Annotations*

In this chapter different classifications for physical annotations were introduced: formal and informal annotations, implicit, and explicit and, lastly, a classification by the physical manifestation of the annotation – textual, graphical and referencing.

The next part of this chapter discusses digital representations for textual, graphical, and referencing annotations.

3.2 FINDING THE RIGHT PROJECTION

The previous part concluded that textual, referencing, and graphical annotations are sufficient for the majority of functions. Based on this rationale, the goal of this part is to find an appropriate digital representation for each of the three signs.

The first big challenge is the representation of the context.

3.2.1 *Digital Context*

In the physical world context is not an issue. The annotator places the annotation exactly on the position he wants it to be. When there is a digital version of the media, it is similar: The annotator can place the annotation directly on a digital version of the media. In both cases the annotation and the annotated medium are conjoined.

The real problem arises when annotation and annotated medium are physically apart – in this case, it is impossible to place an annotation directly and creating and relating an annotation to the medium gets more demanding.

The context has to somehow be defined before creating an annotation: The idea is to present the user fields for inputting the needed information about the context. The challenge here is to choose the context fine enough, so that the user can easily make the connection between the digital annotation and the medium, but also rough enough, so that the user does not have too input to much data before he can start annotating.

There is also a wide range of media that the user can annotate: books, magazines, images, maps, and tables. Images and maps can be either be standalone or contained in a book or magazine. The context has to account for all of these cases.

	Context	**Abstract Context**
I	Chapter of written text	(Media, Chapter, Section)
II	Page(s) of written text	(Media, Startpage, Endpage)
III	Table/Image in written text	(Media, Type, Page, Number)
IV	Image/Map	(Media)

Table 1: Specification of digital context.

Table 1 shows the abstract definitions of digital context represented as tuples. The context of a document can either be a chapter or a range of pages in the document. When annotating an image or a table inside a book, a component for the *type* and one component *number*, in case there is more than one image or table on a single page, is added to the tuple. Table 2 shows examples for the different abstract contexts.

	Example	**Explanation**
I	(A Semantic Web Primer, 2, 3)	Annotation refers to Chapter 2.3 of the book "A Semantic Web Primer"
II	(Parallel Coordinates, 102, 105)	Annotation refers to the pages between page 102 and page 105 of the book "Parallel Coordinates"
III	(Debug It!, DataSet, 45, 1)	Annotation refers to the first table on page 45 in "Debug It!"
III.2	(Semantic Web, StillImage, 13, 2)	Annotation refers to the second image on page 13 in the book "Semantic Web"
IV	(Germany 1918)	Annotation refers to the map "Germany 1918"

Table 2: Examples for digital context.

Now that the context is defined the last step is to find representations for each of the three signs of annotations.

3.2.2 *Proposed Projections*

Based on the definition for digital context, the next step is to find the actual digital representation for textual, referencing, and graphical annotations. For textual and referencing annotations the solution is easy, for graphical annotations rather difficult.

Textual Annotations

The creation of a textual annotation is straightforward: Choose the medium, the context, and enter the content of the annotation in a text area.

Referencing Annotations

With referencing annotations it is similar to textual annotations: The user defines two contexts and that is it. More is not needed to express the relationship between two contexts.

Graphical Annotations

Graphical annotations pose an issue: As already stated, graphical annotations are mostly implicit. One can not say for sure what an exclamation mark or a highlighted passage means. The proposed solution is to identify the widely used functions of graphical annotations and implement them directly.

The creation of a graphical annotation could look like that: A user defines the context and selects the function for this annotation, like "*important*"or "*read again*". With the function hard coded graphical annotations become explicit.

3.2.3 *Mixed Annotations*

Annotations do not strictly belong in one of the three groups, a fact the platform should account for. For example, a user could annotate a chapter and highlight individual words in the annotation or place an asterisk next to a word to reference something. Strictly speaking, such annotations are meta annotations, that is annotations about annotations.

Users take advantage of mixed annotations so the platform should offer the possibility to create them. The solution is the replacement of the simple text field when creating a textual annotation with a Microsoft Word-esque text editor.

Such an editor should provide means for setting text to bold and italic, colour words and passages in the annotation, and creating references from single words or passages.

3.2.4 *Another Approach: Annotating Images*

Before concluding this chapter, one alternative approach should be mentioned: There is a possibility for digital annotation of physical media which opens the possibility for a finer context.

The annotation platform is compatible with a wide range of devices including smartphones and tablets. The idea is that the user takes a photo of the page of the document or image he intends to annotate with the device's camera. After that, the image can be shown on screen which allows for direct placement of annotations.

Graphical and textual annotations can easily be represented: For a graphical annotation provide something like the pen tool in image editing programs, such as Microsoft Paint, and allow the user to draw onto the taken picture. For highlighting the possibility to adjust the opacity of the stroke should be available.

For the creation of a textual annotation the user selects the position on the image he wants the annotation to be placed at. After that, he enters the content of the annotation in a text area and is done. Annotations should not be shown directly on the document rather than being represented through a small icon at the position of the annotation. To read the annotation the user taps on the icon.

Referencing annotations between passages in the same image are no problem: The user can use a graphical annotation and draw something like an arrow. For references between documents or pages, matters become more complicated.

That is the problem with this approach: The context of the created annotations is relative to the taken image with nothing known about the absolute context, that is the medium from which the picture was taken and the page. The context is extremely important for the semantic search engine.

A solution for this would be to combine this approach with the definition of digital context: Before taking a picture, the user defines the document and the page he is going to take a picture from. When creating an annotation, the user has to define what he is annotating: just text, a table, or an image. With this changes the gathered data would be similar in both approaches.

While this approach is definitely pursuable, this thesis focuses on the digital annotation of physical media without use of the camera.

3.2.5 *Wrapping Up Finding the Projection*

Based on the results of the first part of this chapter, this part found an appropriate digital representation for each kind of annotation and presented an alternative approach using the device's camera.

3.3 EVALUATION OF THE PROJECTIONS

This last part of the chapter evaluates the proposed representation of physical annotations.

The biggest problem was the context: When annotating physically, the user can choose to position of the annotation exactly where he

wants it to be. When annotation and medium are physically apart, this is not possible; the context has to be fixed to a certain specification.

But that is not a problem as the task never was to replace the physical annotation; the task was to find a way to allow users the annotation of media when they are not able to write into the medium. Up to this point the annotator needed paper, a system, and his own defined context for this task. And this user-defined context is, with a very high chance, similar to the digital context defined above.

The digital annotations themselves should be as close as possible to the physical annotations, therefore, the different forms of physical annotations were examined. Textual and referencing annotations translate seamlessly in the digital world, while graphical annotations needed a bit of work. Graphical annotations are mostly implicit, their function is for non-authors not always obvious. The proposed solution for this problem is to hard code a fixed set of functions for graphical annotations.

The digitalisation of annotations opens the way for a myriad of features: Forming groups with other users, sharing annotations with users and groups, collaborating on annotations, working on multiple devices, and recommend interesting passages to users.

The next chapter develops ways to gather knowledge in form of formal annotations from the user based on the three kinds of annotations found in this chapter. These formal annotations are going to be used to fuel the semantic search engine.

4

KNOWLEDGE GATHERING AND REPRESENTATION

In the previous chapter the different kinds of physical annotations were explored and ways for the digital representation of those annotations were proposed. Based on these results, this chapter proposes different ways for gathering knowledge in each of the three kinds of annotations. Beyond that, other ways for knowledge gathering are explored. Knowledge gathering is the gathering of formal annotations without resorting to information extraction.

This thesis instead focuses on approaches that make the user create near-formal annotations without him actually noticing doing so. That means the developed approaches can not require any familiarity with the creation of formal annotations, and their use also should yield some kind of advantage for the user: If something is only obstructing one's work, chances are that he will not actually be using it.

This chapter is structured as follows: The first part explains basic terms that are important for the understanding of the remainder of the chapter, in the second part different ways for knowledge gathering are presented, and the last part focuses on the representation of the gathered knowledge.

4.1 BASIC TERMS

The World Wide Web Consortium, abbreviated W3C, proposes technology for structuring, representing, querying, and interchanging knowledge as part of their Semantic Web technology stack.

For knowledge representation and interchange the Resource Description Framework is proposed, for structuring the Web Ontology Language OWL, and for querying the SPARQL Protocol And RDF Query Language. The following chapter introduces these terms.

4.1.1 *Ontology*

> "An ontology defines a set of representational primitives with which to model a domain of knowledge or discourse. The representational primitives are typically classes [...], attributes [...] , and relationships [...]." [11]

In other words: An ontology models the structure of knowledge about an area, for example animals, plants or – in BibPad's case – annotations. The classes are the building blocks of an ontology; the

basic elements that are related to each other and have attributes, for example *Animal*. Attributes model the properties of a class, for example User called *Name*. The relationships model the relations between the classes and classes or classes and attributes, for example User *annotates* Media or User *called* Name. This chapter follows [11].

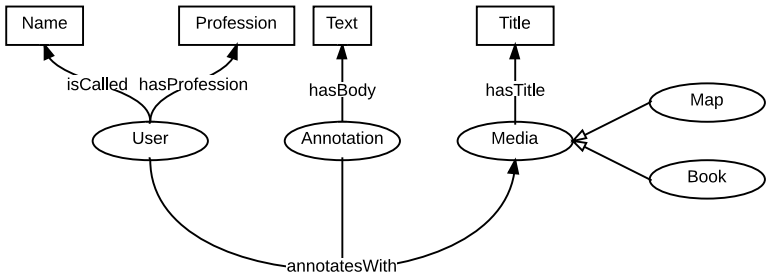

Figure 4: Example for an ontology about annotations.

Figure 4 shows an example for an ontology about annotations represented as a graph: The classes are depicted as ellipses, the attributes as rectangles, and the relationships are the labels of the edges. A user has a name and a profession. He annotates a book with an annotation. The textual representation of an annotation is its body. A book is identified by a title. Map and Book are a special case of classes: They are subclasses of Media that means they inherit all relationships from Media.

An ontology does not model the actual knowledge about a domain rather than being a blueprint for the knowledge about a certain domain. The representation of actual knowledge is called an instance of an ontology.

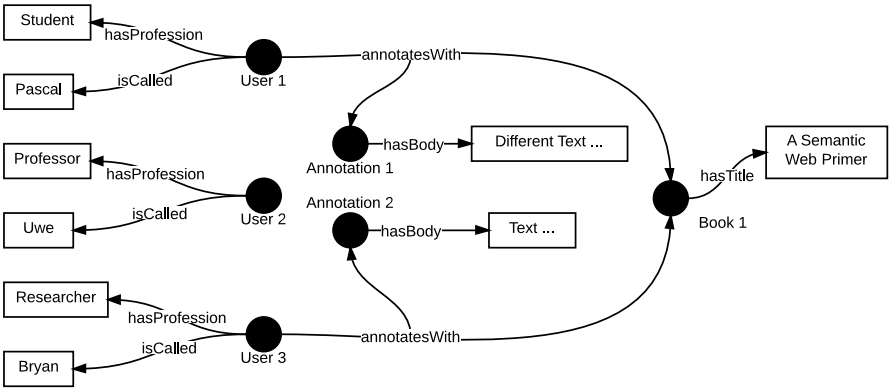

Figure 5: Example for an instance of the ontology about annotations.

An example for an instance of the annotation ontology is shown in Figure 5. There are three users: Pascal, Uwe, and Bryan. Pascal and Bryan each created an annotation about the book 'A Semantic Web Primer', while Uwe has yet to create an annotation.

Another part of an ontology are rules for inferencing and reasoning, that is rules for creating new knowledge out of the existing knowl-

$$\begin{aligned} recommend\_annotation(User1, Annotation) \Leftrightarrow \\ hasProfession(User1, Profession) \wedge \\ hasProfession(User2, Profession) \wedge \\ annotatesWith(User2, Annotation, \_) \wedge \\ \neg(User1 = User2) \end{aligned}$$

Figure 6: Example for a rule.

edge. With BibPad it would be imaginable to recommend one user another user's annotations if they are on a similar or equal level of education.

Figure 6 shows the rule that creates such recommendations. The predicate *recommend_annotation(User1, Annotation)* holds when there is another user with the same profession who has an annotation *Annotation* and is not the same as *User1*. A software that uses such rules to create new knowledge is called a reasoner. Rules can be represented in the Rule Interchange Format short RIF. The rule above in represented in RIF can be found in the Appendix.

4.1.2 *Resource Description Framework*

RDF is an acronym for Resource Description Framework and "is a language for representing information about resources in the World Wide Web" [21]. It is used to represent and interchange information in a standardized way. This chapter follows [4] and [21].

The fundamental concepts of RDF are resources, properties, and statements: The resource being the topic one wants to talk about, for example the annotation. The properties are used to express the relationship between different resources. Resources and properties are identified by an URI. A statement is a subject-predicate-object tuple with the subject being a resource, the predicate a property and the object either another resource or a literal. A literal can be seen as an atomic value represented as a string.

An RDF document can be used to represent an instance of an ontology: In this case, the resources are instances of classes, the properties relations, and attributes literals. The structure of an RDF statement allows only binary relations, therefore, the relation *annotatesWith* in the example ontology is, in its current form, not realizable. The W3C proposes ways to convert n-ary relations, that is relations between n objects, into binary relations in [22]. For the example ontology that means the tertiary relation *annotatesWith* is decomposed into two binary relations: One between *User* and *Annotation* called *creates* and another one between *Annotation* and *Media* called *belongsTo*.

The statements of an RDF document form a graph called a semantic net. In the case of the example ontology, it would be for the most part the same graph as in Figure 5 with URIs and *creates* and *belongsTo* instead of *annotatesWith*. The usage of URIs allows the graph to span several documents because the connection can be established via the unique URI.

There are different ways to serialize a semantic web, among others: The Terse RDF Triple Language, RDF/JSON or RDF/XML. The W3C encourages the usage of RDF/XML.

```
<rdf:RDF
  xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
  xmlns:xsd="http://www.w3.org/2001/XMLSchema#"
  xmlns:bp="http://example.org/bibpad#">

<rdf:Description rdf:about="User1">
<rdf:type rdf:resource="http://example.org/bibpad#User"/>
  <bp:isCalled>Pascal</bp:isCalled>
  <bp:hasProfession>Student</bp:hasProfession>
  <bp:creates rdf:resource="#Annotation1"/>
</rdf:Description>

<rdf:Description rdf:about="Book1">
<rdf:type rdf:resource="http://example.org/bibpad#Book"/>
  <bp:hasTitle>A Semantic Web Primer</bp:hasTitle>
</rdf:Description>

<rdf:Description rdf:about="Annotation1">
<rdf:type rdf:resource="http://example.org/bibpad#Annotation"/>
  <bp:hasBody>Different Text ...</bp:hasBody>
  <bp:belongsTo rdf:resource="#Book1"/>
</rdf:Description>

</rdf:RDF>
```

Listing 1: RDF/XML serialization of the instance of the annotation ontology (snippet).

Listing 1 shows a snippet of the RDF/XML serialization of the ontology instance shown in Figure 5. Every resource is represented with an *rdf:Description* tag. Every *rdf:Description* tag has an attribute *rdf:about* which defines the like of an ID for this resource allowing for referencing. The different properties of the resource are modelled inside the *rdf:Description* tag. The class of the resource is defined via the *rdf:resource* attribute of the *rdf:type* tag. The properties *isCalled*, *hasTitle*, and *hasBody* do not have an object but a literal as object. Literals are modelled as a text node inside the property tag. The other properties use the *rdf:resource* attribute to reference the resource that represents the object of the statement.

At the moment there is no defined vocabulary and semantics for the classes and properties used. One possibility to define such a vo-

cabulary is RDF Schema, which is introduced in the next part of this chapter.

4.1.3 *RDF Schema*

The RDF document shown in Listing 1 can be used to represent and interchange information about annotations, but one thing that is missing is an ontology - or vocabulary - for defining the classes and properties used in the document. For defining a vocabulary W3C proposes the use of RDF Schema, the RDF Vocabulary Description Language. This chapter follows [25].

The term schema is poorly chosen because RDF Schema does not really define a strict schema the document has to follow rather than providing additional information about the used classes and properties.

RDF Schema is provided as a form of an RDF vocabulary and may consequently be serialized as an RDF/XML document.

```
<rdf:RDF
  xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
  xmlns:rdfs="http://www.w3.org/2000/01/rdf-schema#">
<rdfs:Class rdf:ID="User" />
<rdfs:Class rdf:ID="Annotation" />
<rdfs:Class rdf:ID="Media" />

<rdfs:Class rdf:ID="Book">
  <rdfs:subClassOf rdf:resource="#Media" />
</rdfs:Class>

<rdf:Property rdf:ID="isCalled">
  <rdfs:domain rdf:resource="#User" />
  <rdfs:range rdf:resource="http://www.w3.org/2001/XMLSchema#
      string" />
</rdf:Property>

<rdf:Property rdf:ID="belongsTo">
  <rdfs:domain rdf:resource="#Annotation" />
  <rdfs:range rdf:resource="#Media" />
</rdf:Property>

<rdf:Property rdf:ID="creates">
  <rdfs:domain rdf:resource="#User" />
  <rdfs:range rdf:resource="#Annotation" />
</rdf:Property>
</rdf:RDF>
```

Listing 2: RDF/XML serialization of vocabulary for annotation ontology (snippet).

Listing 2 shows some of the classes and properties of the annotation ontology in RDF Schema. The *rdfs:Class* and *rdfs:Property* tags are

used to define a class or property, the *rdf:id* attribute in these tags is used to define the name of the class or property. Properties have a domain and a range. Domain is the class of the subject, range the class of the object in an RDF triple. If the object is a literal, an appropriate data type from XML Schema is used, such as integer or string. A subclass relationship is defined inside the definition of the subclass with a *rdfs:subClassOf* tag as shown in line 9.

Domain and range of a property are not mandatory, that is they do not dictate what kind of resources are allowed for usage with this property. In fact, they allow a reasoner to infer that resources connected via this property are instances of the classes defined for subject and object.

RDF Schema allows for the defining of vocabulary about a domain, but its capabilities are rather basic. A more expressive language for ontology definition is the Web Ontology Language OWL 2.

4.1.4 *Web Ontology Language*

With RDF Schema only the most basic things about a domain of knowledge can be modelled: resources and the relations between the resources. Things as the characteristics of properties such as transitive or irreflexive, or equality of properties can not be expressed [20]. The W3C developed the Web Ontology Language OWL in 2004 followed by OWL 2 in 2009 as a language for defining ontologies more expressive than RDF Schema. The content and some of the examples in this chapter follow [23]. OWL introduces both language constructs for creating ontologies as well as representing instances of ontologies.

When modelling an ontology, OWL 2 offers much more expressivity than RDF Schema. OWL allows for classes to be declared as a conjunction, disjunction, or intersection of other classes, for example the class *Parent* is the conjunction of classes *Father* and *Mother*. With the possibility to define the characteristics of properties it is possible to declare classes such as *sameGradeAs* as reflexive, symmetric, and transitive.

None of the new language concepts introduced in OWL 2 are needed to express the structure of the example ontology, therefore, it is omitted at this point.

For modelling instances OWL introduces negated statements like Mary is not married to Bill without having to introduce a property *isNotMarriedTo*. With OWL it is also possible to define the cardinality of certain properties: Statements like Bill has a maximum of three kids or Bill has exactly one kid are possible.

OWL 2 offers many more features – describing them all would be beyond the scope of this thesis.

4.1.5 *SPARQL Protocol And RDF Query Language*

The previous part introduced various ways for modelling ontologies and their instances. One thing all those ways had in common was their syntax: They can be serizialized in RDF/XML. The last thing that is missing is a language that allows querying an RDF graph. For this task the W3C introduced the SPARQL protocol and RDF Query language abbreviated as SPARQL. This section follows [26].

The simplest SPARQL query is a SELECT query consisting of a projection and a triple in which the objects can be replaced with variables. The projection is one or more of the variables in the triple and defines the structure of the result. A SPARQL query is evaluated by matching the pattern defined in the tuple against the queried RDF graph. The result are the the values of the variables with which the tuple becomes a tuple in the RDF graph.

SPARQL queries can consist of multiple tuples connected with conjunction or disjunction and variables can span multiple tuples.

```
PREFIX: rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
PREFIX: bp:  <http://example.org/bibpad#>

SELECT ?y, ?x
WHERE
{
  ?y rdf:type bp:Annotation .
  ?y rdf:hasBody ?x .
}
```

Listing 3: Example for a SPARQL query.

Listing 3 shows an example for a query that consists of two tuples connected with a conjunction. It returns the URIs of all annotation resources and their textual representations. The first tuple matches every resource of type Annotation and substitutes y for the URI of the resource. The next tuple can be evaluated similarly now that only on variable is unassigned: x is substituted with the content of the annotation y. After that, the substitution is undone and the query looks for another part of the graph that matches the query. The result are the substitutions for x and y with which the tuples of the query become a valid pattern in the RDF graph.

SPARQL also offers features for constructing graph patterns (CONSTRUCT) and getting the whole subgraph for a URI (DESCRIBE), but those features are not important for the platform and thus omitted at this point.

4.2 KNOWLEDGE GATHERING IN ANNOTATIONS

This part of the chapter discusses the different ways for gathering knowledge: One way to gather knowledge is directly in textual, graphical, and referencing annotations, but also other possibilities such as concept maps, introduced later in this chapter, have to be considered.

As already mentioned in the introduction, the goal of this thesis is not to extract knowledge from the annotation itself rather than offering the user different ways for creating formal, or near-formal, annotations without him noticing. Reason for this is simple: In order to test systems for information extraction one needs test data – and at the moment there is no test data. But at a later point, when there is sufficient data to test information extraction systems, it should definitely be considered.

4.2.1 *Textual Annotations*

At the moment a textual annotation only consists of its context and its body. The body is the text the user entered for this annotation. This chapter proposes a way to gather near-formal annotations from users without using the body of the annotation.

The idea is to offer the user not only a simple text area when creating an textual annotation but also pairs of text fields for the creation of key-value pairs.

A key-value-pair is a pair where the first element is an attribute, such as *War* or *Year*, and the second element is a value for this attribute, such as *World War I* or *1918*. Key and value put together result in a tuple: *(War, World War I)* and *(Year, 1918)*.

Together with the knowledge about the context of the annotation these key-value pairs form a statement about the context of the annotation. Figure 7 shows the examples from the paragraph above formulated in natural language.

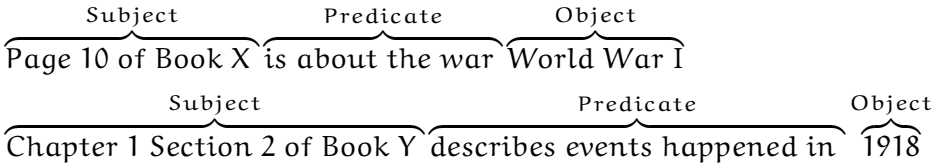

Figure 7: Structure of the statements.

Figure 7 also shows the structure of the statements: Each of the statements can be broken down into a subject, a predicate, and an object. The subject is always the context, the predicate the key, and the object the value. That means those statements can also be formulated as triples similar to RDF statements.

The only thing that is missing for a complete RDF triple is the property and URIs representing subject and object. This part only

discusses the property, the URIs for the resources are discussed later in this chapter. At the moment the property is a word in natural language, but RDF properties like *isAboutWar* or *happenedIn* are needed to describe the knowledge formally. The idea is that the user chooses the right property without him noticing by mapping the natural language keys to properties.

Key	**Property**
War	isAboutWar
Year	happenedIn
Battle	isAboutBattle

Table 3: Examples for keys and their mapped properties.

Table 3 shows examples for natural language keys and their mapped properties. One problem with this solution is that the mapping has to be created manually which means that there has to be one person proficient in the area of ontologies.

Another problem is that different users do not necessarily use the same terms for same meaning. There is a simple solution: Auto-completion that suggests the user keys and values other users have entered while he is typing. The user can either choose one of the suggested terms or continue typing his own term.

One possibility for auto-completion is to compare the current value of the text field with the database - terms that have this value as prefix are suggested. That solution does not work for terms like campaign and expedition. Both mean the same but do not even share the same first character. Therefore, the auto-completion should suggest terms other users used for this medium. If there are too many terms that could be suggested, only the most popular should be shown.

The last thing that has to be considered is that the key-value pairs are, strictly speaking, no key-value pairs rather than keys paired with value lists. Why so? A user may know more values to a certain key, for example battles in World War I had different names depending on the side. The solution is easy: Either the user uses the same key in different key-value fields, or he separates the different values for the same key with a separator, such as a comma.

The key-value pairs gathered here are the driving force behind the semantic search engine – they represent the actual knowledge base that is queried. If there is enough evidence that a key-value pair is true, it can be used to serve queries. If appropriate keys are used by an user and the mapping to RDF properties is established, queries such as *show me books about battles that happened between 1917 and 1918 in Japan* can be answered.

To push the user in the right direction, to give him a hint what kind of keys should be used, some keys could be predefined in a newly created annotation.

A big question is how to verify if a value is correct: A simple majority vote is not possible because, as already mentioned, a key can have multiple valid values. One approach would be not to say a value is true or false rather than a value was used by n users. If a value is used by more users, it is probably more relevant, but a value used by few users is not automatically wrong.

4.2.2 *Referencing Annotations*

When referencing annotations were introduced, one thing was already mentioned: A referencing annotation implies some kind of connection between two contexts. Figure 8 visualizes this fact: The two contexts are nodes, and the arrow is the edge between those two nodes.

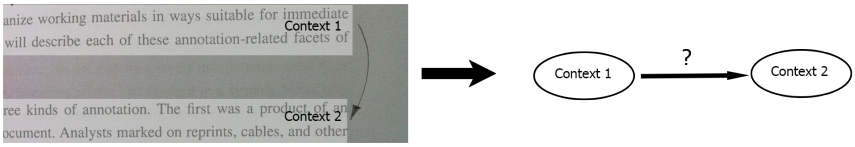

Figure 8: From referencing annotations to graphs.

As suggested by the question mark above the edge, with a normal referencing annotation the kind of relation between the two contexts is unknown, but the kind of relation is important – both for the semantic search engine and the user. The user gets more information about his own referencing annotations, and non-authors can make more sense out of referencing annotations of other users. The semantic search engine can use the referencing annotations to suggest similar media or additional reading about a certain topic.

The solution for this is easy: When creating a referencing annotation, the user has the possibility to define what kind of relationship the two contexts share, such as *see also* or *further reading*.

This again provides something like an RDF triple: the two contexts as subject and object, respectively and the relation as property. The property, again, is in natural language and an appropriate mapping to a formal property has to be established.

Key	**Property**
see also	seeAlso
further reading	furtherReading
same as	sameAs

Table 4: Examples for keys and their mapped properties.

While some of the properties from Table 3 may be applicable, the relations between the contexts are mostly meta relations, that is they relate two contexts and not the content of two contexts. As a result, another set of relations is needed. Table 4 shows examples for such natural language keys and their RDF property equivalents.

4.2.3 *Graphical Annotations*

Gathering knowledge from graphical annotations is not a trivial task: Although through hard coding the function in the creation process the annotations are no longer implicit, they have no real content.

Those annotations are not suited for gathering knowledge for a semantic search engine but for another kind of knowledge that can be used to fuel another feature: Passage recommendation, that is recommending the user passages the majority of other users marked with an annotation. This feature is discussed in detail in Chapter 5.

4.2.4 *Taken Images*

When annotating a taken image, most of the knowledge gathering methods described above can be reused: Fields for key-value pairs can be provided when creating a textual annotation, and the process of creating a referencing annotation is the same as for the digital annotation of physical media.

The only difference are graphical annotations. In this case, there are two kinds of graphical annotations: highlighting with a marker tool and drawing with a pencil tool. The idea is to consider the context associated with the taken image as graphically annotated if the user used either marker, pencil, or both for marking on the image. This allows for this annotations to be used for passage recommendation.

4.2.5 *Factboxes for Media*

The implementation of key-value pairs, the auto-completion and the mapping to RDF properties in annotations can be reused to gather knowledge at a different location: in media in general in the form of factboxes.

Factboxes are used to display tabular data and are found, among others, in many Wikipedia entries. Looking at the structure of a factbox, one thing that strikes is that each row of a factbox is basically a representation of a key-value pair.

The platform offers the user the possibility to create key-value pairs about the media itself, one could say to create a key-value pair with context being the whole media.

The creation of key-value pairs about media themselves offers advantages both for the user and the semantic search engine. The search

engine can use the key-value pairs in the same way as the key-value pairs created in textual annotations. On the user-side these keys and their values could be used to offer means for searching and filtering media, for example queries like *show all media where key Year has value 1918* can be served within the application.

4.2.6 *Maps*

Another possibility of knowledge gathering is to introduce another form of annotations to the users: Maps. This does not refer to a map in geographical terms rather than map as a tool for representing knowledge in form of a graph. This chapter follows [14].

One of the most prominent type of maps are concept maps. A concept map is a graph with its nodes being concepts and its edges being relationships between concepts. A concept is an umbrella term for a somehow related group of events or objects. Concept maps are hierarchical, that is concepts above another concept are always more general. The most general concept is always at the top of the map. Besides the more abstract concepts a concept map can also contain examples for the concepts, e.g. *JavaScript* as an example for concept *programming language*.

In their original form concept maps are not really suitable for gathering knowledge: They focus on concepts and their relationships not on actual examples. A concept map is reminiscent of a graphical representation of an ontology: no actual knowledge rather than concepts and relations. For the semantic search engine actual knowledge is important, so this part introduces a new type of maps based on the structure of concept maps.

This new type of map has the same relations for edges with the nodes being substituted for actual instances of the concepts. By substituting the abstract concepts with actual instances of these concepts, this kind of map can be used to visualize the content of a medium or parts of it.

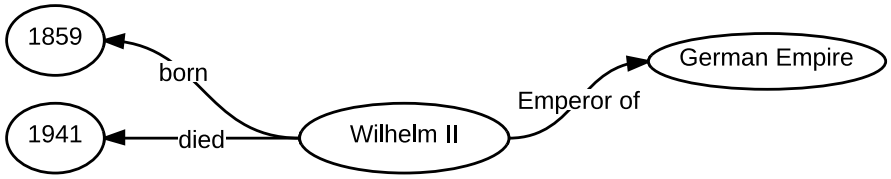

Figure 9: Example for a modified concept map.

Figure 9 shows an example for such a modified concept map: It states that Willhelm II, born in 1859 and died in 1941, was emperor of the German Empire. One thing that strikes is the similarity of a concept map and a semantic net: The nodes of the concept map being resources or literals and its edges being properties.

However, one thing that is missing are URIs identifying the resources and properties, but for properties this is a problem solved before. When creating an edge between two nodes, the user defines the relation between the nodes. Here, again, the auto-completion for previously entered terms and the mapping to RDF properties can be utilized. The URIs for the resources are discussed in the next section.

4.3 KNOWLEDGE REPRESENTATION

After finding means for gathering knowledge in and around annotations, the last step is to find an appropriate way for representing the gathered annotations. One approach for this task would be to define an own ontology, but this would require a great amount of work and thought, and the work would be redundant: An appropriate ontology for modelling annotations already exists!

The W3C proposes the Open Annotation Data Model [24] for this task. While this model is intended for the digital annotation of digital media, it is possible to refactor it with only small modifications for the digital annotation of physical media.

This following chapter is structured as follows: The first part introduces the Open Annotation Data Model, the next part discusses the implementation of the digital context for physical media, and the last part presents the structure for each of the three kinds of annotations, key-value pairs and concept maps.

The following namespaces are used in this chapter: *oa*[1], *cnt*[2], *dc*[3], *dctypes*[4], *rdf*[5], and *bp*[6].

4.3.1 *W3C Open Annotation Data Model*

The idea behind the Open Annotation Data Model "is to provide a standard description mechanism for sharing Annotations between systems" [24]. To achieve this, it specifies a vocabulary for expressing annotations, their content, and related metadata. The Open Annotation Data Model is primarily intended for expressing annotations about digital media, like a video or an image on a website, a PDF document, or a website or parts of it. This chapter follows [24].

Figure 10 shows the main idea of the model: An annotation consists of a *body* and a *target*. The *body* is the actual representation of the annotation, for example the textual content of the annotation. The *target* is the document or part of the document the annotation is referring to. The *type* properties of the *target* and *body* resources indicate what

1 http://www.w3.org/ns/oa#
2 http://www.w3.org/2011/content#
3 http://purl.org/dc/elements/1.1/
4 http://purl.org/dc/dcmitype/
5 http://www.w3.org/1999/02/22-rdf-syntax-ns#
6 http://example.org/bibpad#

kind of medium they are: In the example the *target* is an image that is annotated with a textual remark.

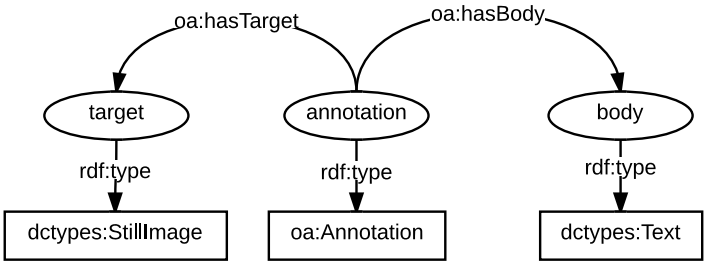

Figure 10: Body and target in the Open Annotation Data Model (adapted from [24]).

The properties of an annotation resource are shown in Figure 11. The objects of *annotatedAt* and *annotatedBy* represent the time when the annotation was created and the user that created the annotation, respectively. With the property *motivatedBy* the reason for creating the annotation is represented. In this case the motivation is *tagging*, that is the motivation to add a tag to a resource. Other motivations include highlighting, describing, or commenting.

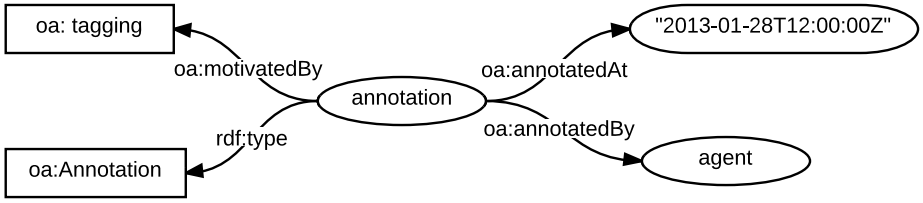

Figure 11: The annotation resource in the Open Annotation Data Model (adapted from [24]).

As already stated, an annotation is not always about the whole resource but most often about a part of it. To accommodate for this fact, the Open Annotation Data Model uses selectors. One of this selectors is a *Fragment Selector*. A fragment is the last part of an URL prefixed by #. The content of a fragment is called fragment identifier. With a fragment identifier a resource can be narrowed down to parts of it.

There are fragment specifications for a wide range of document types, such as PDF documents, plain text, or HTML documents. In HTML documents, for example the fragment identifier *annotation* refers to the element with the id *annotation*.

Figure 12 shows how selectors are used in the Open Annotation Data Model: Previously, the target of the annotation was the annotated resource. This is now replaced with a resource of type *Specific Resource* that represents the part of the resource the annotation is referring to. The original resource and the specific resource are connected through a *hasSource* property.

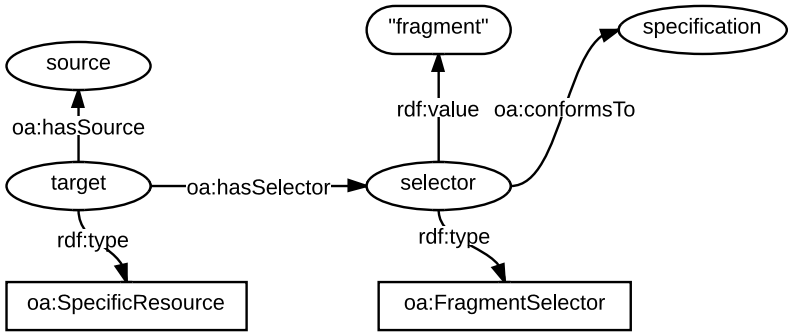

Figure 12: Usage of Fragment Selector (adapted from [24]).

The selector has three properties: *type*, *value* and *conformsTo*. With *type* being *Fragment Selector*, it is indicated that this selector uses a fragment identifier for selecting the desired parts of the document. The property *value* specifies the actual selector, such as *annotation*. The object of *conformsTo* is an URI linking to the specification for the used fragment. The actual URI of the resource is the concatenation of URI, # and the object of the *value* property.

It is very important to state the specification because fragment identifiers can be ambiguous. The fragment identifier *page=10* can be either seen as the selection of page 10 in a PDF document or the element named *page=10* on a website. Table 5 shows a few examples for fragment specifications and examples.

Type	**Example**	**Explanation**
XHTML, HTML	#element	Selects the resource named element
PDF	#page=10	Selects page 10 of a pdf document
W3C Media Fragments	#xywh=50,50,10,20	Selects a 10 pixel wide and 20 pixel high box at x=50 and y=50

Table 5: Example for fragment specifications (adapted from [24]).

Many parts of the Open Annotation Data Model are a perfect fit for the representation of the annotations gathered in the platform, but one thing is missing: a suitable representation of the digital context discussed in the next section.

4.3.2 *Digital Context*

Goal of this chapter is to find a way for representing the abstract context defined in Chapter 3 in the Open Annotation Data Model. The

most important requirement for this representation is upwards compatibility. The digital annotation of physical media offered by the annotation platform is an interim solution: Sooner or later the majority of the books will be available digitally. It should be easy to repurpose the context for digital annotation of digital media and reuse most of the created annotations.

The Open Annotation Data Model already defines a solution for context when annotating a digital document, so in the interest of upwards compatibility the solution for physical media should be as similar as possible to this solution. The solution is to use a *Fragment Selector* with a new fragment specification for physical media fragments. While a physical resource can have a URI, there is no specification for fragments of physical resources.

The specification for physical media fragments should satisfy two requirements: Accommodate all the cases of abstract context defined in Chapter 3, and offer an easy way to transform the fragment identifier into a fragment identifier for digital media.

Syntax	**Description**
chapter=*chapternumber*	Specifies the chapter the annotation is referring to
section=*sectionnumber*	Used to further specify the context of the annotation, to be used in conjunction with *chapter*
endingpage=*pagenumber*	Specifies the first page of the context
startingpage=*pagenumber*	Specifies the last page of the context, same as *startingpage* if the annotation only refers to one page
number=*itemnumber*	Specifies which item within the context is annotated

Table 6: Parameters for physical media fragments.

Table 6 shows the different parameters that are allowed in a physical fragment. A fragment identifier can use as many parameters as necessary separated by an ampersand; valid combinations being the combinations defined in Chapter 3. Table 7 shows how the examples from Chapter 3 are represented as fragment identifiers.

One thing that is missing is the type of the annotated content, such as image or dataset. This is not represented in the identifier but in the type of the specific resource. Figure 13 shows the structure of example III.2.

A problem is the encoding of the fragment identifier as a single string – this makes it impossible to formulate SPARQL queries such as *every annotation for certain media that annotates page 102*. A solution for this problem is to model every part of the fragment identifier as

	Explanation
I	URI#chapter=2&section=3
II	URI#startingpage=102&endingpage=105
III	URI#startingpage=45&endingpage=45&number=1
III.2	URI#startingpage=13&endingpage=13&number=2
IV	URI

Table 7: Examples for fragment identifiers for physical media.

object of a *value* property. To get the whole fragment identifier one simple concatenates all the objects of the *value* properties with an ampersand in between.

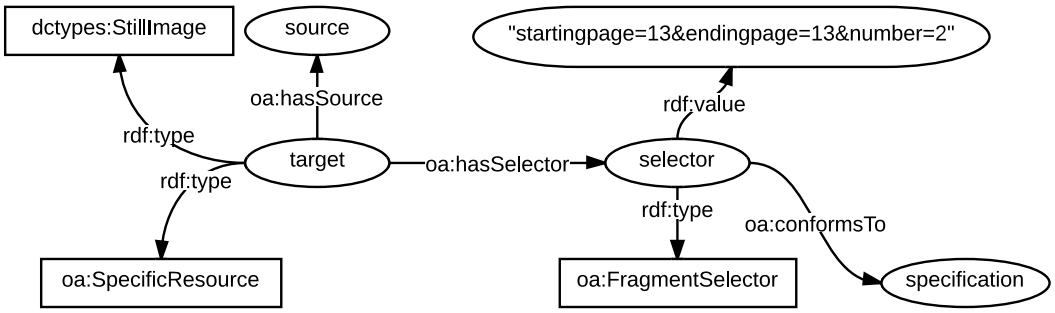

Figure 13: Example for the usage of a physical fragment identifier.

Another thing that is missing is the URI identifying the annotated resource. For this purpose the Bayerische Staatsbibliothek offers an API that provides RDF representations for most of its media. The URI where this representations are found is used as the representation of the annotated resource – the *source* of the *target* resource.

The second requirement for physical media fragments was the easy transformation into other fragments. This is important so that annotations can be reused even when the annotated resource changes from physical to digital. A perfect translation, one way or the other, is impossible, but Table 8 shows a translation that preserves the most important parts of the context.

	PDF Open Parameters	**Media Fragments**
I	URI#nameddest=2-3	-
II	URI#page=102	URI#xywh=10,10,50,50
III	URI#page=45	URI#xywh=10,10,50,50
III.2	URI#page=13	URI#xywh=10,10,50,50
IV	-	URI#xywh=10,10,50,50

Table 8: Translation for physical fragments into PDF Open Parameters[7] and Media Fragments[8].

The translation makes two things possible: The reuse of annotations when changing the resource from physical to digital and the interchange between annotations. An annotation of a physical resource could even show up on an image of the same page of another user.

This leaves only one last task: The representation of the annotations themselves. The next chapter introduces representations for each of the three kinds of annotations, key-value pairs, and concept maps.

4.3.3 *Annotations Themselves*

This last part of this chapter introduces the structures for the annotations. The structure for textual and graphical annotations can be nearly completely be adopted from the Open Annotation Data Model. Referencing annotations, key-value pairs, and concept maps need more modification.

One could argue that it is unnecessary to model the whole metadata of annotations and that only the things needed for the semantic search engine should be modelled in RDF. The same applies to graphical annotations as they are only used for passage recommendation but are not required for serving queries to the search engine.

The reason for this is easy: to future-proof the application. At the moment neither metadata nor graphical annotations are used for serving queries, but this can change quickly. Another reason is that by doing so all the information is together instead of separated in two different kinds of databases. The third reason is modelling the annotations using the Open Annotation Data Model makes the textual and graphical annotations interoperable with other systems using the Open Annotation Data Model.

Textual, Graphical, and Referencing Annotations

The structure of a textual annotation is defined in the Open Annotation Data Model. Figure 14 shows this structure. The actual content of the annotation is the object of property *chars*. The other three properties are required to model the nature of the textual content: In this case it is plain text. An example for another kind of content would be if the annotation used HTML markup. This would be indicated by changing the object from property *format* to *text/html*. The actual selector is omitted in this figure but follows the structure defined in the previous section. For the motivation of textual annotations *describing* was chosen.

A graphical annotation is modelled as an annotation without a body as shown in figure 15. The motivation for graphical annotations

6 http://partners.adobe.com/public/developer/en/acrobat/PDFOpenParameters.pdf
7 http://www.w3.org/TR/media-frags/

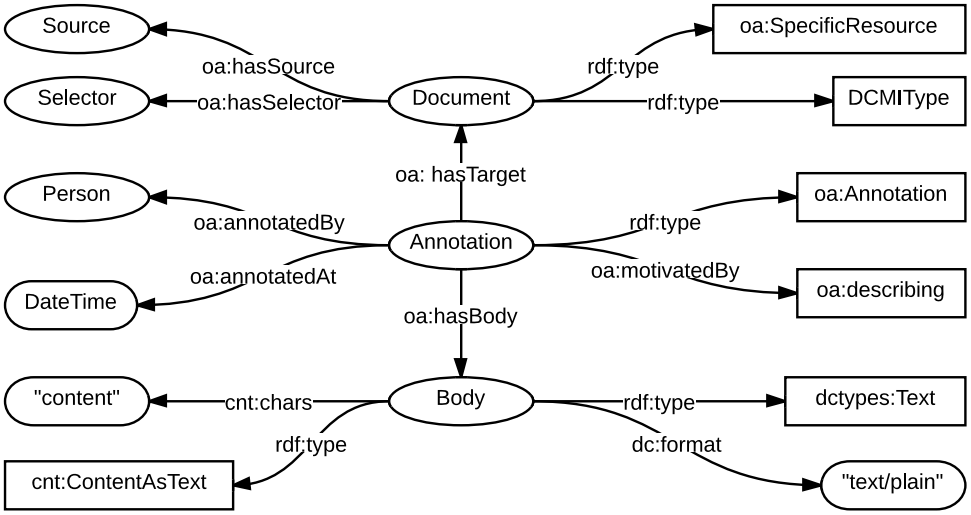

Figure 14: RDF structure for a textual annotation (adapted from [24]).

is either *highlighting* or *bookmarking* depending on the chosen function of the annotation

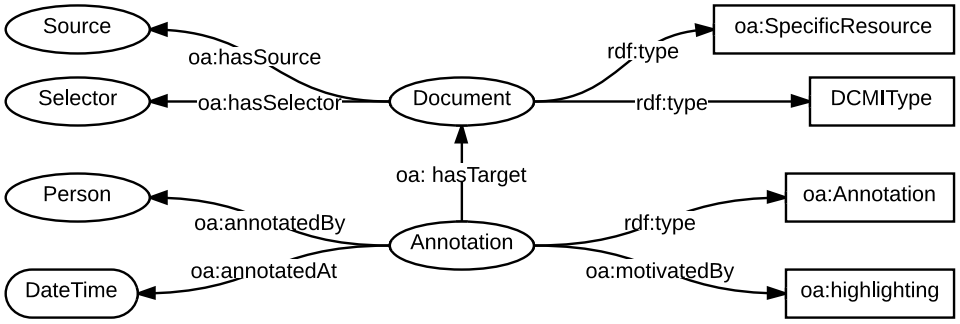

Figure 15: RDF structure for a graphical annotation (adapted from [24]).

The structure of a referencing annotation needs a little modification. First of all, the Open Annotation Data Model does not provide a motivation for referencing, only for linking. The difference between the two is that a link is untyped while a reference is a typed link. The platform requires users to define the kind of relationships between context, therefore, referencing annotations gathered are typed links between two contexts, thus, the first modification is the introduction of a motivation for typed links: *referencing*.

The next modification concerns the body of the annotation: A referencing annotation consists of two contexts – each modelled as a medium with context as shown in the previous chapter – connected through a property. One approach is to drop the annotation resource and connect the two contexts directly, but there are two drawbacks: Loss of metadata and the structure is no longer consistent with the other models. The solution is to treat a referencing annotation as an annotation without a body: The second context is added via the

mapped property – indicated with an asterisk in Figure 16 – to the annotation resource.

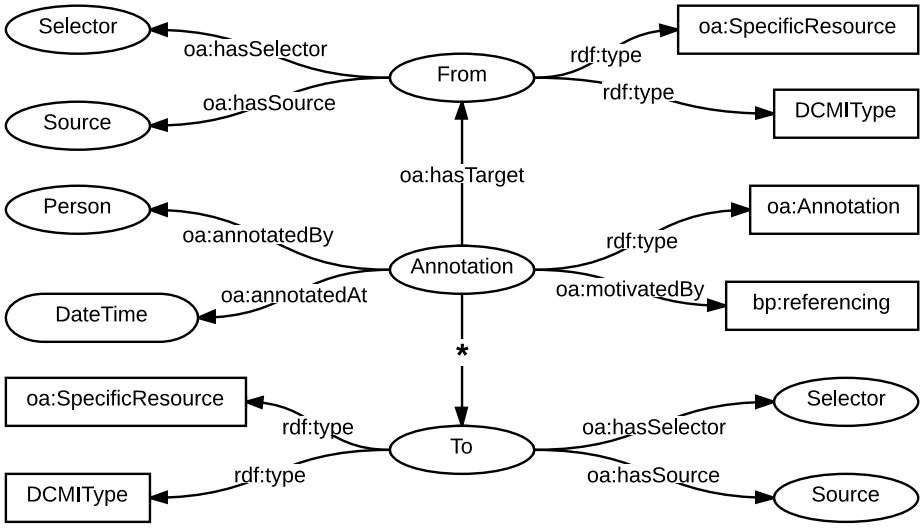

Figure 16: RDF structure for a referencing annotation.

Modified Concept Maps and Key-Value Pairs

The representation of a concept map looks nearly the same as the concept map itself – the only changes are the properties that model the type and the format of the nodes. The content of the node is modelled via property *value*. Figure 17 shows the structure for a small concept map consisting of two nodes and one edge. The property is represented by the asterisk between the two resources. The asterisk is to be replaced with the property mapped to the natural language key entered by the user.

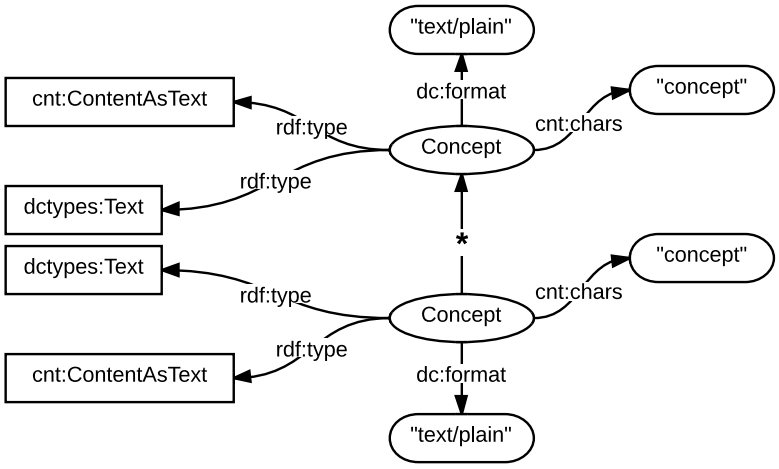

Figure 17: RDF structure for a modified concept map.

For a key-value pair this structure is reused: A key-value pair is seen as a concept map with only two nodes. The first node is used to model the key, the second one for the value. For the property between the two resources the mapped property is used. In case of more keys for the same value, a new value node for each value is created and connected via the mapped property to the key node.

For the sake of brevity, the structure of the annotation and target resource is omitted for key-value pairs and concept maps. The structure would be the same as for textual annotations. The only thing that is different are the motivations: A motivation for creating concept maps has to be created: *mapping*, for key-value pairs an appropriate motivation already exists: *tagging*.

4.3.4 *Ontology for the Mappings*

Referencing annotations, key-value pairs, and the modified concept maps all use some kind of mapping from natural language terms to RDF properties. A point that has yet to be addressed is the ontology where those properties come from. There are two possibilities: For each natural language term search for an appropriate property in the different ontologies found on the internet or define an own ontology and add properties when the need arises.

The first possibility is rather tedious: Looking for an ontology, checking if the intended meaning is the meaning one is looking for, and finally mapping it to the natural language term. The second possibility is much easier: Think of a property that represents the semantics of the natural language term and enter the property and a small description in which cases to use it. Because the platform creates its own ontology, the remaining task to create the OWL representation of the property can be done automatically.

One thing which so far remained unmentioned are classes for the keys, values, and the nodes in the concept maps. The problem is that those terms are in natural language and without test data it is impossible to make a general statement such as the majority of the subjects of the *happenendAt* property are countries. The classes are important but can not be decided at this point. But this is no problem: If, at some point in time, there is enough evidence that the majority of subjects of a property are of a certain class, this class can be added as range for the respective property. After that, the class can be inferred for all statements using this property.

4.4 WRAPPING UP KNOWLEDGE GATHERING

This chapter addressed two problems: How to gather knowledge from annotations and how to represent the gathered knowledge.

For knowledge gathering in annotations different ways, such as key-value pairs, were proposed. Most of those solutions are dependent on a mapping of natural language terms to RDF properties. This approach has one drawback: The mapping has to be established by a person.

Another possibility for annotating and, therefore, gathering knowledge was introduced: a modified version of concept maps which can reuse the mapping established for the annotations.

The representation of the gathered knowledge was examined in the next part of this chapter: For this task the Open Annotation Data Model was used. This part introduced physical media fragments as a mean for representing the digital context while preserving maximum upwards- and backwards compatibility.

The next chapter discusses the actual implementation of the annotation platform and the used technology.

5 BIBPAD

The previous chapters discussed the different kinds of annotations, their digital representation, and knowledge gathering in said annotations. The following chapter brings everything together – it discusses the actual implementation of those theoretical concepts in the annotation platform BibPad.

This chapter is structured as follows: The first section establishes a common vocabulary for recurring terms, like user or annotation, and explains the possible interactions between the actors. The next section discusses the different features an annotation platform should offer and their actual realization. In the last section the used technology is discussed.

5.1 BASIC TERMS

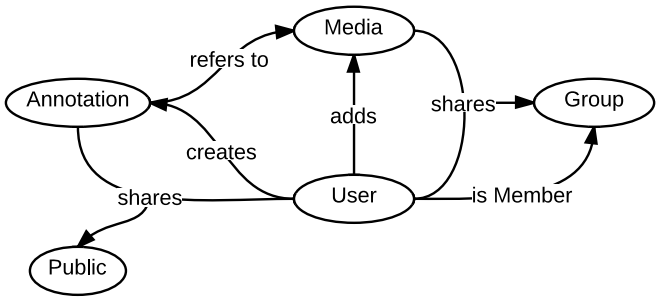

Figure 18: Actors and their interactions in BibPad.

Figure 18 shows the different actors and interactions between them in BibPad. The main actor is the user – every interaction is initiated by him. A user adds media to his personal library – after that, he can start creating annotations for this document or parts of it. Annotations created by the user can be flagged private or public. A public annotation is automatically shared with the public which consists of all users of the platform.

Groups are a major part of BibPad: They allow users to share media and the associated annotations between them. A group is led by a user and has other users as members. Not annotations, but media is shared within a group, that is after a user shared a medium with a group the annotations said user created for this medium are also available for the members of the group. Private annotations are only visible for the user that created the annotations.

5.2 FEATURES

The digital annotation of physical media is only an interim solution because digitalization of media is an ongoing process. As soon as a digital version for a medium is available, annotation directly on the digital version of the document is the preferred option. This should also be reflected in the architecture of the platform: Components that are independent from the form of annotating should be built in a way that makes them easily reusable.

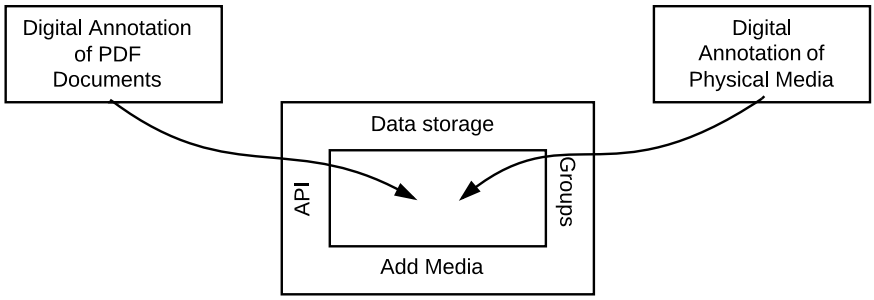

Figure 19: BibPad framework.

Figure 19 shows the structure of BibPad: Features like adding of media, groups, and especially data storage and API can be reused, with only the actual annotation component requiring replacement. There are also examples for possible annotation components given: digital annotation of physical media, which is the approach discussed in this thesis, and digital annotation of PDF documents, an approach that is imaginable in the future if a digital version of the medium is available.

5.2.1 *Framework Features*

The first set of features are framework features; features that are completely independent of the form of the annotated media. That includes, among others, features like login and registration, management of media in a user's library, and group management.

Registration and Login

An important feature is user management, that is a system for user registration and login. This is a very important feature because without this feature there would be no reliable way to keep track of a user's media, groups, and annotations.

The registration process in BibPad is kept as simple as possible requiring only the data that is needed for the smooth operation of the platform. Required data includes a unique username, an email address, and an opt-in for the usage of the annotation data for the

semantic search engine. Optionally, the user can provide his level of education and area of expertise.

Adding Media to the Library

Before one can start annotating media, there must be a possibility to add media: BibPad offers the users three different ways to add media – a barcode scan, the library's signature of the medium, or manually entering the media's metadata.

The easiest possibility is the barcode scan – scan the barcode of the medium with the device's camera, and the medium is added to the user's library. However, not every medium is a book or a magazine and not every medium has a barcode, so there must be other ways to add media. One of those ways would be adding by using the libraries signature of a piece of media. The signature of a medium is a unique, mostly alphanumeric, string that identifies media within a library.

If the two possibilities described above fail, another way is being provided: In this case, BibPad offers the possibility to manually add a medium by providing the user a couple of fields for entering the relevant metadata by himself.

In each of the three cases described above the user has the ability to categorize the added medium. Those categories could either be personal, such as 'Bachelor's Thesis', or more general, like 'Linear Algebra'.

Of the three ways described, the first two are the most desirable: They are enabling the creation of a link between physical resource, database record, and annotation. The third way bears a problem: How to determine which physical medium the user's manually created medium is referring to?

Sharing Media and Annotations

As already mentioned, one important aspect of BibPad is the social component. The user is not alone – he is part of a whole ecosystem of annotators. Annotations can be shared either with the whole ecosystem, the public, or only a subset of the ecosystem, a group.

To make this possible, the platform offers ways to create groups. BibPad's implementation of groups is simple: A user creates a group and adds other users to the group. The added user is now a member of the group and can share media and associated annotations with the group. An annotation can be set to private – by doing so a user can prevent the sharing of this annotation with a group even when he shares the medium associated with this annotation with the group. Conversely, an annotation can also be set to public – after that every user of the application can view the annotation.

Sharing is important, but what if two or more people want to work together on an annotation? There must be means for collaborating. Users can choose wether the annotations for a shared medium can be edited by other members of the group or not. This system gives the responsibility for the safety of the annotations the user. Collaboration is not a part of the current implementation of BibPad but certainly a desirable future extension.

Working in the Cloud

It is very likely that humans are more inclined to type longer texts and notes on a physical keyboard rather than on a digital keyboard found on tablets and smartphones. BibPad is available for every system that has an internet browser at its disposal – so why not take advantage of this?

After adding a medium on one of the user‘s devices, it is instantly available on the user‘s other devices. A user could for example use his smartphone for scanning the barcodes of a book – after that, the book is instantly available for annotation on the user‘s laptop. The reverse way is also imaginable: After a day of annotating books with a laptop, the user can review the annotations with his smartphone while commuting home without taking out his laptop.

This permanent synchronization is not restricted to annotations: creating a group, adding members, and sharing media – every interaction with the platform is immediately synchronized with all of the user‘s devices. This synchronization is not limited to the user‘s own actions: Actions of other users and their side effects are also immediately synchronized with every connected device. An example for this is being added to a group. After a user has been added to a group, the media shared within the group is being made visible to him. The synchronization of the other user‘s actions is not part of the current implementation.

Passage Recommendation

Another imaginable feature is passage recommendation, that is the recommendation of interesting passages in a medium based on the distribution of annotations in said medium. Passage recommendation is explored by Bradshaw and Light in [8] on which this chapter is based on.

Bradshaw and Light examined if and how the passages annotated by researchers in scientific papers overlap. For the study they collected 16 articles each annotated by four to seven participants and broke them down into passages of interest which they defined as “a continuous sequence of sentences in a single document for which at least one reader annotated every sentence “ [8].

Bradshaw and Light evaluated those passages of interest under various aspects:

- **Q1**: To what degree did readers overlap in the passages they highlighted, underlined, or in some other way annotated?
- **Q2**: On what fraction of the reading material did annotators focus the majority of their attention?
- **Q3**: What is the distribution of annotation throughout documents?

Reason for the first question is clear: If there is no consensus between the annotators, then a system for passage recommendation can not give reasonable recommendations. For the second question: If the annotated material spans a great fraction of the document, then such a system is also unreasonable as the goal is to reduce the amount of reading. On the third question: If the annotations are always concentrated on certain parts of the document, such as the discussion section, a system for passage recommendation would always recommend the same parts and would be useless.

For the first question the result was that the "majority of annotators agree on almost 60% of the annotated sentences found throughout the documents" [8]. Regarding Q2 the finding was that "nearly 80% of annotations are concentrated in passages accounting for only 1/3 of the total data set" [8] and for Q3 that "annotations are distributed fairly evenly throughout specific section types" [8].

These results demonstrate that a system for passage recommendation for scientific papers is definitely imaginable. In BibPad's case most of the media which are suitable for passage recommendation are books and magazines which tend to be longer. It has to be shown that a passage recommendation is also possible for books – the annotations gathered with the platform are an excellent starting point for doing so.

Other Features

There are also a number of smaller features which contribute to a positive user experience. These features are not a part of the annotation task itself rather than small features which add further incentive to usage of BibPad.

BibPad can also act as a bibliography management system as BibPad has every information at hand needed for outputting a bibliography. The user chooses a category and the platform exports the media in this category in a format readable for literature management systems, such as JabRef or EndNote.

5.2.2 *Annotation Components*

The main part of the platform is the annotation component. In the previous chapters the digital annotation of physical media was established, but also the annotation of taken photos was shortly outlined. This chapter discusses the actual realization for these two ways of creating annotations.

Digital Annotation of Physical Media

After adding a medium to their library, users can start creating annotations for it: The first step is always the definition of context and type. The next step varies between the different kinds of annotations: If creating a graphical annotation, the user chooses a function for this annotation. If creating a referencing annotation, the user defines the target context and kind of relation between them. The creation of a textual annotation prompts the user with a text area and text fields for entering key-value pairs.

Created annotations, annotations shared with the user by other users, and public annotations are shown in a list that can be filtered, for example to show only annotations the user created himself or annotations about a particular part of the medium. If the user taps on a textual annotation, the annotation itself and related annotations are shown. Related annotations are annotations that do not exactly have the same context but overlap with the context of said annotation, for example for an annotation about pages 100 to 104 the annotation about pages 102 and 106 is shown as a related annotation.

Digital Annotation of Photos

After the user defined the context – in this case, it is either the page and the medium or only the medium he is going to take a photo of – the user takes the photo. The taken photo is shown and the user can highlight things, draw graphical annotations, or place textual annotations on the image.

Figure 20 shows an screenshot of the prototype of the annotation tool: In the toolbar at the top of the page the user can choose to either draw or highlight things and the colour of the stroke. After selecting a tool, the user can draw on the image. Textual and referencing annotations are created by double-tapping on the desired position which opens a dialogue box where the user can choose between *textual* or *referencing*. After selecting *textual*, the dialogue shows a text area and text fields for entering key-value pairs. The existence of a textual annotation is indicated by a small icon on the image; a tap on the icon shows the whole body of the annotation. After selecting *referencing*, the user is shown a dialogue where he can define the target context

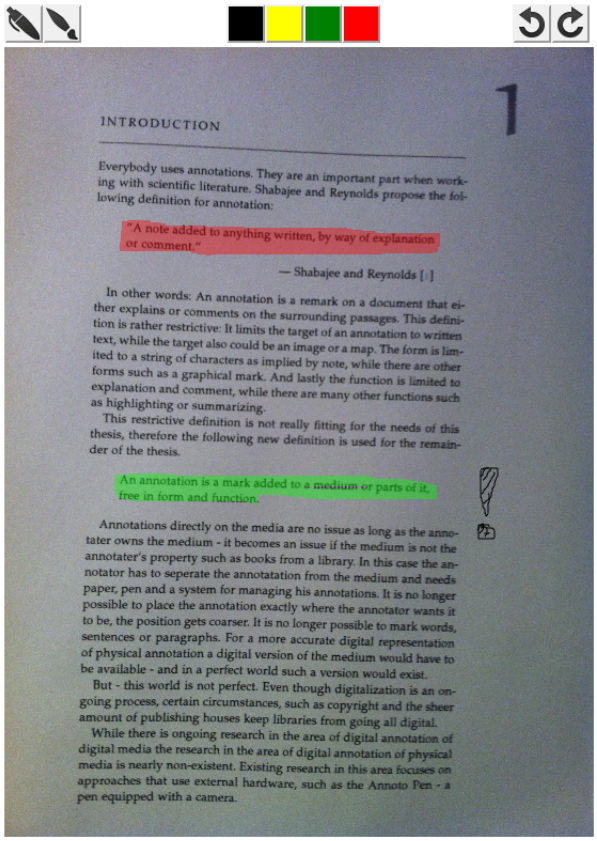

Figure 20: Screenshot of the annotation component for the annotation user-taken images.[1]

and the kind of relation. Here, too, the existence is indicated by an icon – different from the one indicating a textual annotation.

Sharing of textual and referencing annotations is not a trivial task: Users annotate on different versions of the same page, so that the positions of the annotations can not be applied directly to another picture. One solution would be to assume that the different pictures are similar enough that the connection between the original position and the new position can be easily established. At the moment it is not possible to predict the users' behaviour regarding taking pictures, so this is something that has to be monitored. Graphical annotations are much more position-dependent than referencing and textual annotations, so the sharing of graphical annotations over various versions of the same page is not supported by BibPad.

5.3 USED TECHNOLOGY

This section provides a brief overview of the technology used in the implementation of BibPad. As already mentioned, the main objective

1 Icons by http://pc.de/icons/

was to choose technology that allows for a fast deployment on as many platforms as possible.

BibPad uses Sencha Touch 2.2 as a foundation, Apache Cordova to gain access to the device's system functions, like camera or geolocation, and postgreSQL as database. The API for the communication between application and database is implemented in Java using the Java API for RESTful Web Services.

5.3.1 *Sencha Touch 2*

Sencha Touch, "a high-performance HTML5 mobile application framework" [16] is a model-view-controller framework written in HTML5, CSS3 and JavaScript. This section follows [16].

Sencha Touch supports a wide range of devices: every iOS device from iOS version 4.0 and up, Windows 8 Phones, Android starting with version 2.3, and the most recent BlackBerry devices. The framework provides native look and feel for most of these operating systems.

A big drawback of Sencha Touch is the varying performance – on older devices applications developed with Sencha Touch have nowhere near the performance of a native application. On newer devices the difference in performance is virtually non-existent. The advantages of developing with Sencha Touch greatly outweigh this drawback, thus, BibPad is implemented in Sencha Touch.

Applications written with Sencha Touch are basically a website. Therefore, the deployment is easy – just upload the framework's files to a web server and done. Another possibility for deploying an application is native packaging, that is packing and signing your application in a format supported by the respective store, like Apple's AppStore or Google's PlayStore. Drawback of native packaging is that every store needs its own package. A drawback of native packaging for iOS is the performance: It is possible that a native application runs slower than its browser equivalent. The reason for this is that Sencha Touch uses iOS' UIWebView class to display the website representing the application. UIWebView has no access to Apple's Nitro Engine which is responsible for interpreting JavaScript – that is why JavaScript on websites displayed via UIWebView is slower than in the native Browser, Safari.

Starting from version 2.0 Sencha Touch offers access of limited set of system functions via HTML 5 APIs, like taking pictures, getting the device's orientation, and creating native notifications. System functions not available include getting the device's exact geolocation and accessing the accelerometer data. In order to use this functions a wrapper API like Apache Cordova is needed.

5.3.2 *Apache Cordova*

Apache Cordova is an API which allows the access of native device functions through JavaScript. While the code which exposes the system functions in JavaScript is platform-specific, the JavaScript API for accessing these functions is consistent. That makes for very easy deployment across multiple platforms. Cordova supports nearly every existing mobile platform: iOS, Android, BlackBerry, Windows Phone, Palm WebOS, Bada, and Symbian. Cordova comes with plugin support so that users can write own APIs for own needs. This chapter follows [5].

BibPad requires Cordova for providing the possibility to do a barcode scan. There exists a Cordova plugin[2] that provides this functionality for iOS, Android, BlackBerry, and Windows Phone. The plugin allows for an easy way to read a book or magazine's barcode. The HTML 5 API for accessing the device's camera can not be used at this point: Currently, only Blackberry's browser supports the required API. Nevertheless, an implementation of a barcode scan using only JavaScript is discussed in the Appendix.

Cordova can be used in conjunction with Sencha Touch, however, the native code of Cordova prevents the deployment of the application as a webpage. When using Cordova, the only possibility for distributing the application are the respective stores. As a result, only the packed version has the means of adding media via a barcode scan.

5.3.3 *PostgreSQL*

BibPads database is powered by PostgreSQL, "a powerful, open source object-relational database system" [19]. A relational database allows for quick access to structured data. It offers operations for adding, manipulating and reading data through the Structured Query Language abbreviated as SQL.

ID	Username	Emailadress
1	Lilly Harmonie	lilly@harmon.ie
2	Phoenix Lauri	lauri@phoenix.org
3	Gabriel Jonas	gab@jo.com
4	Laureen Liona	lau@reen.com

Table 9: Example for a database table.

Relational means that all data is organized in relations. Table 9 shows an example for such a relation. The first row is not part of the relation – it only defines the schema of the table. A column in the

2 https://github.com/phonegap/phonegap-plugins/tree/master/Android/BarcodeScanner

first row is called an attribute. The other rows are manifestations of the schema defined in the first row – all manifestations together form the relation. The tuples have to adhere to the defined schema even tough one or more components of a tuple can be *null*.

There are other relational databases management systems like MySQL, Microsoft SQL Server, or SQLite. There are differences between the systems, but the choice is personal: One takes the system one is most familiar with. BibPad uses postgresSQL as database.

The next question is how to access the data stored in the database? The next chapter introduces a way to design a database API using REST principles.

5.3.4 *REST API*

REST is the abbreviation of Representational State Transfer which is "a set of architectural principles by which you can design Web services that focus on a system's resources" [15]. Examples for resources are annotations or users in the database. A web service with a REST API is called a RESTful web service. This chapter follows [15].

One of the basic principles of REST is that every resource in the database has its own URI. Listing 4 shows an example for such an URI – addressing the annotation with id 8372.

```
1 http://example.org/bibpad/api/annotation/8372
```

Listing 4: Example for a URI.

Another principle of REST is the mapping of create, read, update and delete operations (CRUD operations) onto the HTTP methods POST, GET, PUT, and DELETE.

```
1 GET     http://example.org/bibpad/api/annotation/8372
2 DELETE  http://example.org/bibpad/api/annotation/8372
3 POST    http://example.org/bibpad/api/annotation/
4 PUT     http://example.org/bibpad/api/annotation/8372
```

Listing 5: CRUD and HTTP methods.

Listing 5 shows examples for API calls. The first call yields the data for annotation 8372 as answer. The second call deletes the annotation with the specified id. The other two calls are different from the first two because they need additional payload, so that the server can process the request. Call number three orders the server to create a new annotation with the payload provided. The last call updates annotation with id 8372 as specified in the payload.

The mapping of CRUD operations onto HTTP methods is, therefore, as follows:

- **C**reate is POST

- **R**ead is GET
- **U**pdate is PUT
- **D**elete is DELETE

A REST web service is stateless, that is that a client must include everything needed to generate the response in its request. This requirement allows REST web services to be highly scalable – when the state is included in every request, it is irrelevant which server answers the request as long as all servers have access to the same database.

The last principle of a REST web service is the format of the payloads. There are two possibilities: JSON or XML. In case of BibPad JSON is the best solution: Like the name suggests, JSON is native to JavaScript, so parsing and creating it is very easy.

The data for the application has to be exposed via a REST API because the web site representing the application is not rendered on the server side, but on the client side. If the web site would be rendered on the server side, there would be direct access to the database, but the client only has JavaScript at its disposal which provides no means for accessing a remote SQL database. One thing JavaScript offers though is the possibility to send HTTP requests to a server and receive answers – exactly what is needed to use a REST API, and therefore, a REST API is a perfect fit for BibPad.

The last question is how to implement a RESTful web service? There are different possibilities: Java offers the Java API for RESTful Web Services (short: JAX-RS), C# has the Windows Communication Foundation, and for node.js exist modules like *restify* or *express*.

BibPad's API is implemented in Java, because it is very easy to access the database and prototype the API in Java.

5.3.5 *Getting Data About the Media*

The last problem of this chapter is the retrieval of data about the media. Previously, three ways to add new media into the library were described – this chapter only discusses the first two: Adding via ISBN/ISSN and adding by using the library's signature of the media.

The ISBN/ISSN can be used to look up the media's metadata in the B3Kat directory. B3Kat is a joint directory of libraries in Bavaria, Berlin and Brandenburg. The directory comprises of 23 million records of books and magazines. The data is organized in form of RDF triples [18]. For querying the data B3Kat's LinkedOpenData-Server exposes a SPARQL endpoint.

```
1 SELECT * WHERE {?s bibo:isbn '9783897214903'} LIMIT 50
```

Listing 6: SPARQL query for medium with ISBN 9783897214903.

The query in Listing 6 returns the URI of the book with ISBN 9783897214903. This URI leads to a JSON document containing all the RDF triples associated with this book – every piece of information needed to create the book in our database.

As previously mentioned, not every media has a barcode or an ISBN/ISSN, so there must be a way to add media by the library's signature. At this point exists no API for that purpose. For a productive use of BibPad such an API has to be developed.

5.3.6 *Bringing Everything Together*

Now that all the technology used in the implementation of BibPad is discussed, it is time to bring everything together. Figure 21 shows the architecture of BibPad.

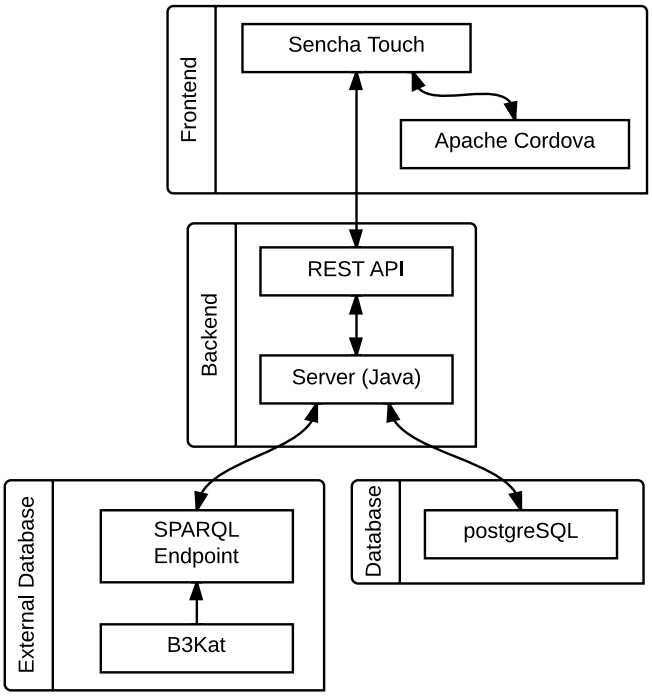

Figure 21: System architecture of BibPad.

On the bottommost level are the databases. For storage of application data postgreSQL, for getting information about media the SPARQL endpoint exposed by B3Kat is used.

All communication with databases is initiated by the server implemented in Java using JAX-RS to expose a REST API. This REST API is in turn used by Sencha Touch to get, create, and alter data in the database.

The application itself is implemented in Sencha Touch. In the packaged version Apache Cordova is used to gain access to system functions.

5.4 WRAPPING UP BIBPAD

This chapter discussed the architecture and implementation of BibPad. There are two kinds of features: Those who are independent from the form of annotation and those who are not. Independent are features like adding and management of media and social features, like groups and collaboration, while the not independent feature is the actual annotation component. Two different kinds of annotation components were introduced: the physical annotation of digital media and the annotation of user-taken images.

The other part of this chapter discussed the used technology: Sencha Touch allows for fast deployment on a wide range of devices, Apache Cordova for accessing system functions, and a REST API for querying the data via JavaScript.

Actual implementations of the features were not part of this chapter. A discussion on the implementation of some features can be found in the Appendix. Discussed features are the synchronization and the required HTTP push mechanism, the annotation of images, and an outline for a barcode scanner using only JavaScript.

6

SUMMARY AND OUTLOOK

This thesis had two goals: Find a solution for the digital annotation of physical media and find ways to gather knowledge from those annotations. Knowledge gathering was introduced as an umbrella term for approaches to gather formal or near-formal annotations from a user without him actually noticing.

The reason for research in this area is a project from the Bayerische Staatsbibliothek. The Bayerische Staatsbibliothek has a great amount of media about World War I and wants this stock to be searchable. The idea is to develop a platform that allows users to create annotations about media and use the knowledge gathered in the annotations to fuel a semantic search engine about this media.

The first goal was discussed in Chapter 3 which introduced different kinds of physical annotations and proposed a digital representation for context and annotations themselves. Textual, referencing, and graphical annotations turned out to be the most important kinds of annotations – they are sufficient to represent the majority of annotations users create. The context of an annotation could not be represented as fine as in physical annotation but fine enough that the relation between context and annotation can still be established. Another possibility for annotating was outlined: Annotating pictures taken with the device's camera. This approach offers the user the possibility to place the annotation exactly where he wants it to be but also has drawbacks, such as restricted sharing features.

Chapter 4 built on the findings of Chapter 3 and proposed various ways to gather knowledge from those kinds of annotations. Ways beyond those annotations to gather knowledge were discussed, too. When creating a textual annotation, the user is offered the possibility of creating key-value pairs which represent a statement about the context of the annotation. With the help of a mapping from natural language keys to formal RDF properties the statements are formalized and can be used to fuel the semantic search engine. For referencing annotations a similar approach was developed: By providing the user the possibility to define the kind of relationship between the two contexts, the semantic search engine gains the knowledge to make recommendations such as *see also* and *further reading*. Graphical annotations yield no real knowledge for the semantic search engine, but are used for another feature: Passage recommendation.

Besides knowledge gathering in the three kinds of annotations, two other proposals were made: factboxes for media and a modified form of concept maps. Factboxes reuse the idea of key-value pairs, but

instead of using them in a specific context they are used to create key-value pairs for the whole medium. Modified concept maps are an option to visualize knowledge – with the help of an appropriate mapping this visualization can be formalized and subsequently used the same way as the key-value pairs: fueling the semantic search engine.

Another big section of Chapter 4 was the representation of the gathered knowledge: The Open Annotation Data Model from W3C was used as a foundation and modified for the new task at hand. Modification was needed because the Open Annotation Data Model was originally conceived for representation of digital annotation of digital media and lacks support for key-value pairs, concept maps, and digital context.

In Chapter 5 the theoretical concepts discussed in the previous chapters were brought together to create a concept for the annotation platform BibPad. BibPad is built as a framework consisting of two parts: BibPad Core containing all the functionality that is independent from the form of annotating and BibPad Annotation which houses the actual annotation component. BibPad is written with the Sencha Touch framework which allows for fast deployment on a wide range of devices. Apache Cordova allows access of system functions and, therefore, provides the foundation for features like a barcode scan. A big focus of BibPad are the social features: User can share media in groups or with everyone using the application and collaborate on annotations.

FUTURE WORK

This thesis could not consider everything – either because some things would go beyond the scope of the thesis, or their consideration was just not possible at this point.

At the moment the actual content of a textual annotation is not considered for the semantic search engine – something that definitely should be done as soon as there is enough test data. Even if tests with systems for information extraction fail, other things should be considered, such as creating an inverted index from the content of the annotations.

The classes of values in key-value pairs and nodes in concept maps are not modelled at this point. While its probable that users use terms from the same type for the same concept, without enough test data safe prediction is not possible. However, that is not a real problem: If, at some point, enough evidence exists that a certain value or node is of a certain type, it can be added as range for the respective property. After that, the class can be inferred for all resources using this property.

A few features of BibPad were omitted in the implementation: Collaboration is not possible at the moment, and the tool for annotating

a user-taken photo only exists as a prototype and is not implemented in the BibPad ecosystem.

The annotation of images also opens the door for research in another area: crowdsourcing digitalization. It is to examine if users take pictures of the majority of pages of a book and if the pictures are good enough for OCR software to return reasonable results.

Lastly, the viability of passage recommendation for books has to be examined: At the moment BibPad only offers a perfect foundation for a study in this direction, but tools for the statistical evaluation of the annotations have to be developed.

OUTLOOK

The digital annotation of physical media is only an interim solution – the digitalization of media is an ongoing process and sooner or later the majority of media – at least the subset of media that is relevant for a semantic search engine about World War I – will be available digitally.

To accommodate for this changing landscape, BibPad utilizes a framework architecture: Many features can be reused with minor modifications, while only the annotation component itself has to be developed from ground up. The representation of the annotations is also built in a way that allows for usage with other forms of annotating and even the parallel usage of two different forms of annotating.

One way or the other the future has to show if BibPad has a place in libraries and if the features that the digitalization of annotation adds are compelling enough for users to break free from their existing annotation behaviour and embrace the digital annotation. Hopefully they are.

A

APPENDIX

A.1 IMPLEMENTATION OF HTTP PUSH

There are two kinds of events that have to be pushed to currently connected devices: internal and external events. An internal event is an event that only needs to be pushed to one user's devices, such as adding media to the library. An external event is an event that has to be pushed to a greater number of devices, such as adding a member to a group has to be pushed to every member of the group.

For this to work the server has to be able to push events to connected devices. The current implementation uses long polling; other options would be WebSockets[1] or Server-sent Events[2].

The idea behind long polling is easy: The client sends a request with a very long timeout, and the server only sends an answer if an event occurs. In case no event occurs, the request simply times out and the connection is re-established by the client. This allows the server to react immediately to occurring events and push them to connected devices. While there are more modern technologies, long polling was chosen because Sencha Touch offers native support for XMLHttpRequest. This section discusses the current implementation of long polling in BibPad.

```
1  @GET
2  @Path("events/{userId}/{deviceType}")
3  @Produces(MediaType.APPLICATION_JSON)
4  public static Response getEvents(
5      @PathParam("userId") String userId,
6      @PathParam("deviceType") String deviceType) {
7
8      Long timeStart = System.currentTimeMillis();
9
10     while((System.currentTimeMillis() - timeStart) < 40000) {
11         // check periodically for new events and if a event
               occurs send answer
12     }
13     return Response.status(Response.Status.NOT_FOUND).build();
14 }
```

Listing (Java[3]) 7: Long polling server-side (snippet).

1 https://developer.mozilla.org/en-US/docs/WebSockets
2 https://developer.mozilla.org/en-US/docs/Server-sent_events
3 http://jax-rs-spec.java.net/nonav/2.0/apidocs/index.html

Listing 7 shows the server-side implementation of long polling. The method *getEvents* is called if a GET request on path */events/{userId}-/{deviceType}* is retrieved – this is indicated by the first two lines of Listing 7. The third line indicates that the payload of the response is JSON. *userId* and *deviceType* are path parameters that are passed as arguments of the method.

The method checks periodically for the duration of forty seconds if an event occurs. If this is not the case, the server answers with *not found*. In case an event occurs, the method answers with a response containing the relevant event data as payload.

```
1  doEventRequest: function() {
2      var url = baseUrl + '/events/' + userId + '/' + deviceType;
3      Ext.Ajax.request({
4          url: url,
5          method: 'GET',
6          success: this.handleEvents,
7          failure: this.doEventRequest,
8          scope: this,
9          timeout: 60000
10     });
11 },
12
13 handleEvents: function(response) {
14     var responseObject = JSON.parse(response.responseText);
15
16     // handle events
17
18     this.doEventRequest();
19 }
```

Listing (JavaScript[4]) 8: Long polling client-side (snippet).

The client-side is shown in Listing 8. After calling *doEventRequest* for the first time, a HTTP request with timeout of 60 seconds is sent. In case of timeout or failure, the method calls itself again. In case of success, a second method *handleEvents* is called which handles the events and calls *doEventRequest* again.

The request timeout and the forced server-side timeout are different in duration. This is required as a kind of security padding. In the current implementation an event is immediately deleted after retrieving it from the database. In this scenario, it could happen that an event is retrieved from the database, but the request times out before the server can send an response with the event data as payload. The event is already deleted from the database, so even in case of a re-established connection the event is lost forever and never reaches the client. One solution to solve this problem would be to implement an

4 http://docs.sencha.com/touch/2.2.1/#!/api/Ext.Ajax

acknowledgement mechanism – an event is only deleted if the client acknowledges its retrieval.

The device type is included in the URL of the request. This is required to deliver the user the correct events: It is assumed that a user has at most one device of each kind, that is a smartphone, a tablet, and a laptop. An internal event is only created for the user's other devices – by doing so, it is prevented that the event is sent to the device that executed the action that spawned the event. An external event is created for every device of the affected users.

This implementation is pretty basic: It creates problems for users that use more than one device of the same type. For a more solid implementation another value should be used as an differentiator, such as an unique id assigned to every device when first connecting to the server.

Table 10 shows the different events that can occur and the event data that has to be sent. The first column describes the action the user executes; the second column describes what kind of information has to be included in the response's payload in order to handle the event and its side effects correctly.

Annotations are not synchronized. This is because annotations are the biggest part of the application, and loading every possible annotation at the start of the application would waste bandwidth. Annotations are loaded on demand and are refreshed if required. Events that require refreshing annotations are marked with an asterisk in Table 10.

Event	Created event data
User adds media	**IN**: Data about the added medium and the category
User creates group	**IN**: Data about the added group
User deletes media	**IN**: ID of the deleted medium **EN***: ID of the deleted medium to every member of the groups the user shared the medium with
User shares media	**IN**: ID of shared medium, ID of group **EN***: Data about the shared medium to every member of the group the medium was shared with
User unshares media	**IN**: ID of the unshared medium, ID of the group **EN***: ID of the unshared medium to all group members
User adds group member	**IN**: Data about added member, ID of group **EN**: Data about the added member to every group member **EN**: Data about group to the added member
User removes group member	**IN***: ID of removed member, ID of the media this user shared with group **EN***: Same as above to every group member **EN***: ID of group, IDs of media shared within group to removed member
User leaves group	**IN***: ID of left group, IDs of media shared within the group **EN***: Data about the left member, IDs of media this user shared with the group to every group member
User disbands group	**IN***: Data about the group, IDs of media shared within this group **EN***: Data about the group, IDs of media shared within this group to every group member

Table 10: Different events and created event data (IE for internal event, EE for external event).

A.2 IMPLEMENTATION OF IMAGE ANNOTATION

Image annotation can be divided into two parts: Creating graphical annotations and creating textual or referencing annotations. The actual realization was discussed in Chapter 5; this part discusses the implementation.

The current implementation uses HTML5's canvas element on which it is possible to draw images and simple shapes. The next two sections assume that a canvas exists and the picture to annotate is already drawn onto the canvas. The following example uses jQuery[5].

Graphical Annotation

As already mentioned, the graphical annotation is implemented as a simple pen tool – the user can choose between different colours and two tools: pen or highlighter. Pen draws a line, while highlighter draws a wider transparent line.

A line consists basically of three actions: Press the mouse button, move the mouse, and when finished, release the mouse button. The first action starts the line, the second action draws the actual line, and the last action closes the line. This behaviour can be modelled with the help of three event listeners: One for *mousedown*, one for *mousemove*, and a last one for *mouseup*. Listing 9 shows how to attach the event listeners to the canvas element. This snippet assumes that the canvas has the id *annotatecanvas*. Lines 8 and 9 attach event listeners for click and double-click used later for the creation of textual and referencing annotations.

```
1  function attachEventListener() {
2      var canvas = $('#annotatecanvas');
3
4      canvas.mousedown(beginLine);
5      canvas.mousemove(drawLine);
6      canvas.mouseup(endLine);
7
8      canvas.dblclick(showCreateAnnotationModal);
9      canvas.click(hideModals);
10 }
```

Listing (JavaScript) 9: Attaching event listeners.

Listing 10 shows the implementation of the functions called for each of the events. On mousedown *beginLine* is called. It sets *mouseDown* to true indicating that the left mouse button is pressed. On *mouseup mouseDown* is set to false. After that, *beginLine* begins the path and sets the start coordinates of the line. On *mousemove* a line is drawn from the coordinates set in *beginLine* to the current coordinates

5 http://jquery.com/

of the cursor. The path is now stroked, that is it is now visible on the *canvas*. After that, the path is closed and a new path is started like in *beginLine*. On *mousedown* the path is only stroked and closed after that.

A simpler solution would be to only stroke the line on *mousemove*, but this solution does not allow transparency: On *stroke* the whole line so far is drawn, so repeated calling of *stroke* on a transparent line results in an opaque line. A drawback of the current solution is that the line appears aliased. The solution for this is the *redrawCanvas* function in combination with the *doneLines* array.

```
1
2  function beginLine(event) {
3      mouseDown = true;
4
5      context.beginPath();
6      context.moveTo(event.offsetX, event.offsetY);
7      doneLines.push({
8          startLine: true,
9          x: event.offsetX,
10         y: event.offsetY
11     });
12 }
13
14 function drawLine(event) {
15     if(mouseDown) {
16         context.lineTo(event.offsetX, event.offsetY);
17         context.stroke();
18         context.closePath();
19         context.beginPath();
20         context.moveTo(event.offsetX, event.offsetY);
21         doneLines.push({
22             x: event.offsetX,
23             y: event.offsetY
24         });
25     }
26 }
27
28 function endLine(event) {
29     context.stroke();
30     context.closePath();
31     doneLines.push({
32             stopLine: true
33     });
34     redrawCanvas();
35     mouseDown = false;
36 }
```

Listing (JavaScript[6]) 10: Functions for the eventlistener.

6 https://developer.mozilla.org/en-US/docs/HTML/Canvas

Every stroke is pushed to *doneLines* with start and end of a line having an additional flag that indicates the beginning and the end of a line, respectively. After each line *redrawCanvas* is called which uses the information saved in *doneLines* to redraw the lines in *doneLines* onto the canvas. Each line is drawn in a single stroke, so transparent lines are still transparent and not aliased.

Listing 11 shows the implementation of the *redrawCanvas* function. The first two lines clear the canvas and draw the annotated image again. The following for-loop has three cases: The first case begins a path and moves to the right position, the second case strokes the line and closes the path, while the third case moves the path to the saved coordinates.

```
1  function redrawCanvas() {
2       context.clearRect(0, 0, canvas.width, canvas.height);
3       renderImage();
4
5       for(var i = 0; i < doneLines.length; i++) {
6          if(typeof doneLines[i].startLine === 'boolean') {
7              context.beginPath();
8              context.moveTo(doneLines[i].x, doneLines[i].y);
9          } else if(typeof doneLines[i].stopLine === 'boolean') {
10             context.stroke();
11             context.closePath();
12         } else {
13             context.lineTo(doneLines[i].x, doneLines[i].y);
14         }
15     }
16 }
```

Listing (JavaScript) 11: Function for redrawing the content of the canvas.

The implementation of the different colours and tools is omitted at this point but is pretty straightforward: For different colours *context.strokeStyle* has to be set to the appropriate colour; for transparent colours the colour is set as an RGBA value, such as *rgba(255, 255, 0, 0.4)* for a transparent yellow line. In case of highlighter, *context.lineWidth* is set to 10, for pen to 1.

Referencing and Textual Annotations

Textual annotations are indicated by an icon placed on top of the canvas. A click on the box opens the edit dialogue for this annotation; a double-click on an empty position on the canvas creates a new annotation at the clicked position. This section only outlines how to place the annotations on the canvas element and how to show the dialogue for creating and viewing annotations.

The boxes indicating the existence of an annotation are *div* elements that are positioned on top of the canvas. Another solution would be

to draw the indicators directly onto the canvas, but as the API for click detection on paths inside a canvas is not yet implemented in most browsers, the current solution is preferable. The arrangement of DOM elements on top of each other is enabled via the *z-index* CSS property. Of two elements the element with the higher *z-index* is positioned on top of the element with the lower *z-index*. Listing 12 shows the CSS rules used to achieve this behaviour.

```
1  div#canvasholder {
2      position: relative;
3      z-index: 0;
4  }
5
6  canvas#annotatecanvas {
7      z-index: 0;
8  }
9
10 div.annotation, div#annotationdialog, div#annotationcontent {
11     z-index: 1;
12 }
```

Listing (CSS) 12: CSS rules for positioning of elements on top of a canvas element.

A *div* element with class *annotation* represents the indicator for an annotation; the element with the id *annotationdialog* is the dialog for creating the content of an annotation.

In Listing 9 event listeners for click and double-click where attached to the canvas element. Listing 13 shows the implementations of the functions called for these two events.

```
1  function showCreateAnnotationModal(event) {
2      var annotation = $('<div class="annotation" id=' + aId + '
           style="position: absolute; top:' + event.offsetY + 'px;
           left:' + event.offsetX + 'px;">');
3      annotation.appendTo('#canvasholder');
4      annotation.click(openAnnotation);
5      aId = aId + 1;
6
7      $('#annotationdialog').attr('style', 'display: block;
           position: absolute; top:' + event.offsetY + 'px; left:' +
            event.offsetX + 'px;');
8  }
9
10 function openAnnotation(event) {
11   var id = event.currenTarget.id,
12     x  = event.currentTarget.offsetLeft + 'px',
13     y  = event.currentTarget.offsetTop + 'px';
14
15   // set fields of annotationinfo to the contents of annotation
         id
16
```

```
17    $('#annotationcontent').attr('style', 'display: block; position
          : absolute; top:' + y + '; left:' + x + ';');
18  }
19
20  function hideModals(event) {
21      $('#annotationdialog').attr('style', 'display: none;');
22      $('#annotationcontent').attr('style', 'display: none;');
23  }
```

Listing (JavaScript) 13: Code for showing and hiding the annotation div.

When double-clicking the *canvas*, a new *div* element with class *annotation* is created and positioned where the double-click occurred. After that, the *div* element with the id *annotationdialog* is moved to this position and made visible – the user can now edit the contents of the annotation. If the user clicks somewhere else on the canvas, and the annotation dialogue is visible, it is hidden again.

An event listener is added to the created annotation indicator. A click onto the indicator calls the function *openAnnotation* that sets up the dialogue with the content of the annotation and shows the dialogue. To differentiate between the annotations, the *id* attribute of the *div* with class *annotation* is set to the *id* of the annotation. In this case, it is just a variable that counts up; in a more realistic implementation this should be substituted with the actual id of the annotation in the database.

The current implementation has no means for creating actual annotations, it only serves as a proof of concept that the technology is able to support the digital annotation of user-taken images.

A.3 BARCODE SCANNER IN JAVASCRIPT

The possibility to implement a barcode scan using JavaScript was briefly mentioned but waved aside as pointless because the needed APIs are not yet available for mobile browser. Nevertheless, an implementation of a barcode scan in JavaScript should be considered as soon as the required APIs are available for mobile browsers which would eliminate the need of a packaged version of BibPad.

Implementation of a barcode scan is using the *getUserMedia* function which allows the application to access the device's camera to get an continuous stream of video from the camera. Lines 6 to 10 in Listing 14 show how to call *getUserMedia*: The first argument is a JavaScript Object stating what should be streamed, the second and third argument are success- and failure callbacks. The implementation of the failure callback is omitted at this point.

The first three lines of code create references to the DOM elements needed in the next step: The first line gets a reference to the canvas element, the second line one to the canvas' context, and the last line

one to a *video* tag. The *video* tag has the CSS property *display: none* and the attribute *autoplay* set to true.

The success callback *success* sets the source of the *video* tag to the incoming videostream and, after that, initiates the barcode scan by calling the function *doScan*. This function takes the current image of the *video* tag and draws it on to the canvas element. After that, it gets the image data of the current content of the canvas and scans the image data for a barcode. If a barcode is found, it can now be used to look up the medium's metadata. Otherwise the function *doScan* is called again to get the current picture from the camera and, thus, a new chance to find the barcode.

```
var canvas = document.getElementById('scancanvas'),
    context = canvas.getContext("2d"),
    video = document.getElementById('webcamvideo');

function init() {
  navigator.getUserMedia(
    {video: true, audio: false},
    success,
    failure
  );
}

function success(mediaStream) {
  video.src = window.URL.createObjectURL(mediaStream);
  doScan();
}

function doScan() {
  context.drawImage(video, 0, 0);
  var imageData = canvas.getImageData(0, 0, canvas.width, canvas.
      height);

  // do a barcodescan on imageData

  if(barcodeFound) {
    // get medium's metadata
  } else {
    doScan();
  }

}
```

Listing (JavaScript[7]) 14: Barcode scan using HTML5's getUserMedia API (snippet).

7 https://developer.mozilla.org/en-US/docs/Web/API/Navigator.getUserMedia

One thing that is missing is an actual implementation of a barcode scanning algorithm in JavaScript. A solution for this would be to port the Zebra Crossing library[8] ("ZXing") from Java to JavaScript.

A.4 RULE INTERCHANGE FORMAT

```
Document(
  Prefix(rdfs <http://www.w3.org/2000/01/rdf-schema#>)
  Prefix(pred <http://www.w3.org/2007/rif-builtin-predicate#>)
  Prefix(bp <http://example.org/bibpad#)

  Group(
    Forall ?User1 ?User2, ?Profession, ?Media (
      bp:recommend(?User1 ?Annotation) :-
        And(bp:hasProfession(?User1, ?Profession)
            bp:hasProfession(?User2, ?Profession)
            bp:creates(?User2, ?Annotation)
            bp:isCalled(?User1, ?Name1)
            bp:isCalled(?User2, ?Name2)
            External(pred:literal-not-identical(?Name1 ?Name2))
        )
    )
  )
)
```

Listing (RIF[9]) 15: Rule for annotation recommendation.

8 http://code.google.com/p/zxing/

9 http://www.w3.org/TR/2013/NOTE-rif-overview-20130205/

BIBLIOGRAPHY

[1] Annette Adler, Anuj Gujar, Beverly L. Harrison, Kenton O'hara, and Abigail Sellen. A Diary Study of Work-Related Reading: Design Implications for Digital Reading Devices. In *Proceedings of the SIGCHI conference on Human factors in computing systems*, pages 241–248. ACM Press/Addison-Wesley Publishing Co., 1998.

[2] Maristella Agosti and Nicola Ferro. Annotations: Enriching a Digital Library. In *Research and Advanced Technology for Digital Libraries*, pages 88–100. Springer, 2003.

[3] Anoto Group AB. Anoto. http://www.anoto.com/lng/en/mode/sublist/documentId/1150/pid/480/, 2013. Accessed May 17, 2013.

[4] Grigoris Antoniou and Frank Van Harmelen. *Semantic Web Primer*. The MIT Press, 2004.

[5] Apache Software Foundation. Apache Cordova. http://cordova.apache.org/, 2013. Accessed May 07, 2013.

[6] Bonnie B. Armbruster. Taking Notes From Lectures. In *Handbook of College Reading and Study Strategy Research*, pages 175–199. Lawrence Erlbaum Associates Publishers, 2000.

[7] Paolo Bottoni, Roberta Civica, Stefano Levialdi, Laura Orso, Emanuele Panizzi, and Rosa Trinchese. MADCOW: a Multimedia Digital Annotation System. In *Proceedings of the working conference on Advanced visual interfaces*, pages 55–62. ACM, 2004.

[8] Shannon Bradshaw and Marc Light. Annotation Consensus: Implications for Passage Recommendation in Scientific Literature. In *Proceedings of the eighteenth conference on Hypertext and hypermedia*, pages 209–216. ACM, 2007.

[9] Burke H. Bretzing and Raymond W. Kulhavy. Note-taking and passage style. *Journal of Educational Psychology*, 73(2):242, 1981.

[10] Corsin Decurtins, Moira C. Norrie, and Beat Signer. Digital Annotation of Printed Documents. In *Proceedings of the twelfth international conference on Information and knowledge management*, pages 552–555. ACM, 2003.

[11] Tom Gruber. Ontology. www.tomgruber.org/writing/ontology-definition-2007.htm, 2007. Accessed May 08, 2013.

[12] Catherine C. Marshall. Annotation: from paper books to the digital library. In *Proceedings of the second ACM international conference on Digital libraries*, pages 131–140. ACM, 1997.

[13] Catherine C. Marshall. The Future of Annotation in a Digital (Paper) World. In *Proceedings of the 35th Annual Clinic on Library Applications of Data Processing: Successes and Failures of Digital Libraries*, pages 43–53, 2000.

[14] Joseph D. Novak and Alberto J. Cañas. The Theory Underlying Concept Maps and How to Construct Them. *Florida Institute for Human and Machine Cognition*, 1, 2006.

[15] Alex Rodriguez. RESTful Web services: The basics. http://www.ibm.com/developerworks/webservices/library/ws-restful/, November 06, 2008. Accessed May 13, 2013.

[16] Sencha Inc. Sencha Touch. http://www.sencha.com/products/touch, 2013. Accessed May 07, 2013.

[17] Paul Shabajee and Dave Reynolds. What is Annotation? - A Short Review of Annotation and Annotation Systems. Technical report, Graduate School of Education and Institute for Learning and Research Technology (ILRT), University of Bristol, Bristol, UK, 2003.

[18] Bayerische Staatsbibliothek. LinkedOpenData-Service des B3Kat: lod.b3kat.de. http://lod.b3kat.de/doc, 2013. Accessed May 14, 2013.

[19] The PostgreSQL Global Development Group. PostgreSQL: About. http://www.postgresql.org/about/, 2013. Accessed May 07, 2013.

[20] W3C. OWL Web Ontology Language Overview. http://www.w3.org/TR/2004/REC-owl-features-20040210, 2004. Accessed June 27, 2013.

[21] W3C. RDF Primer. http://www.w3.org/TR/rdf-primer/, 2004. Accessed June 24, 2013.

[22] W3C. Defining N-ary Relations on the Semantic Web. http://www.w3.org/TR/swbp-n-aryRelations/, 2006. Accessed June 25, 2013.

[23] W3C. OWL 2 Web Ontology Language Primer (Second Edition). http://www.w3.org/TR/2012/REC-owl2-primer-20121211/, 2012. Accessed June 26, 2013.

[24] W3C. Open Annotation Data Model. http://www.openannotation.org/spec/core/index.html, 2013. Accessed May 08, 2013.

[25] W3C. RDF Vocabulary Description Language 1.0: RDF Schema. http://www.w3.org/TR/rdf-schema/, 2013. Accessed May 08, 2013.

[26] W3C. SPARQL 1.1 Query Language. http://www.w3.org/TR/sparql11-query/, 2013. Accessed June 27, 2013.

[27] Chih-Sung Andy Wu, Susan J Robinson, and Ali Mazalek. Turning a page on the digital annotation of physical books. In *Proceedings of the 2nd international conference on Tangible and embedded interaction*, pages 109–116. ACM, 2008.