Mag. phil. Matthias Rella, BSc

Bits and Pieces: A Generic Widget Framework for Sensemaking on the Web

Master's Thesis

Graz University of Technology

Knowledge Technologies Institute
Head: Univ.-Prof. Dipl.-Inf. Dr. Stefanie Lindstaedt

Supervisor: Univ.-Prof. Dipl.-Inf. Dr. Stefanie Lindstaedt

Graz, March 2015

This document is set in Palatino, compiled with pdfLaTeX2e and Biber.

The LaTeX template from Karl Voit is based on KOMA script and can be found online: https://github.com/novoid/LaTeX-KOMA-template

Statutory Declaration

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

Graz, _____ _____

Date Signature

Eidesstattliche Erklärung[1]

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

Graz, am _____ _____

Datum Unterschrift

[1] Beschluss der Curricula-Kommission für Bachelor-, Master- und Diplomstudien vom 10.11.2008; Genehmigung des Senates am 1.12.2008

Abstract

The term “sensemaking” refers to a universal concept being investigated in various sciences specifically or interdisciplinary. Briefly spoken from the perspective of the information sciences, sensemaking occurs when a person has to deal with a huge, perhaps overwhelming, and heterogeneous amount of information and make sense out of it. This process, which is probable to happen in everyday life, and the sense being made as a product of this process are subject to constant research, especially under the nowadays threat of the information deluge. The World Wide Web is the media for today’s huge and heterogeneous amount of information which pervades our everyday’s life. Whether we need to do a deep search for scientific literature, figure out which hotel to book when travelling or simply need to keep track of our Web surfing, we engage in a kind of sensemaking on the Web. However, common-purpose user interfaces capable of the dynamic and heterogeneous nature of the information on the Web are missing. This thesis enlightens the term sensemaking from various theoretical perspectives and reviews existing user interface approaches. Then, it develops a novel theoretical and technical framework approach for building user interfaces for sensemaking on the Web, which is finally evaluated in a user study and in expert interviews.

Kurzfassung

Der Begriff “Sensemaking” (zu Deutsch: aus etwas Sinn gewinnen) bezieht sich auf ein universelles Konzept, welches in verschiedenen, auch interdisziplinären Forschungsrichtungen untersucht wird. Aus Sicht der Informatik handelt der Begriff vom Umgang mit einer großen, nahezu überwältigenden und heterogenen Menge an Information durch eine einzelne Person, die daraus Sinn gewinnen will. Dieser Vorgang, der auch im täglichen Leben stattfinden kann, und der gewonnene “Sinn” als sein Produkt sind Gegenstand der aktuellen Forschung, insbesondere in Hinblick auf die immer größer werdende Informationsflut der heutigen Zeit. Das World Wide Web (WWW) ist das Medium dieser riesigen und heterogenen Informationsmengen, das unseren Alltag immer stärker durchdringt. Ob wir nach wissenschaftlicher Literatur suchen, ob wir ein passendes Hotel für die Reise finden wollen, oder ob wir einfach den Überblick über unser Surfverhalten behalten möchten, betreiben wir eine Art von “Sensemaking” im WWW. Benutzeroberflächen, die gängige Zwecke dieser Art erfüllen und dabei der dynamischen und heterogenen Natur der Informationen im WWW gerecht werden, gibt es jedoch nicht. Diese Arbeit erhellt den Begriff “Sensemaking” aus Sicht verschiedener Theorien und bespricht existierende Ansätze für entsprechende Benutzeroberflächen. Anschließend wird ein neuartiger Ansatz vorgestellt, welcher in Form eines theoretischen wie technischen Rahmenwerks die Implementierung von Benutzeroberflächen für das “Sensemaking” im WWW anleiten und ermöglichen soll. Schlussendlich wird der Ansatz in einer Benutzerstudie und in Experteninterviews evaluiert.

Acknowledgements

This master's thesis is not the outcome of my only efforts. As it was written in the course of my enployment at the Social Computing Group at the Know-Center, Graz, I want to say "Thank You" to all my colleagues there: Dieter Theiler, Dominik Kowald and Peter Kraker were always on my side. Especially Sebastian Dennerlein and Emanuel Lacić supported me generously throughout the evaluation of the thesis. Last but not least, I want thank Christoph Trattner who gave me great advice in structuring and writing this work.

It was also a pleasure to work with Mohammed Al-Smadi, Vladimir Tomberg and Pjotr Savitski from Tallinn University, who provided me with essential hints to the whole topic of "Sensemaking".

Finally, I want to thank my family and my wife Conny for always supporting me during my studies.

Contents

List of Figures

1. Introduction

The evolution of this master thesis is a case of sensemaking itself. As every eager student who is interested in a topic I started by searching for existing literature on this topic. Luckily most recent publications can be found on the Web in so-called online catalogues of well-known publishers like ACM, Springer or IEEE or via search engines like Google Scholar or CiteSeer. Not like two decades back when one had to spend days in libraries, dive through rows of catalogues, pull journals and books from shelves and make copies of articles, I had almost all the necessary information a few mouse click away. Thanks to dedicated software tools like Zotero or Mendeley collecting these resources has never been easier.

However, this technological support was both a blessing and a curse. Having not much sense for *sensemaking* I grabbed every little piece of information I could find, starting from the simple but broad question "What is sensemaking?". Within no time I retrieved a whole bunch of papers more or less related to the topic, the one treating it in the domain of information visualization and interaction, the other stressing a sociological approach. I even came across sensemaking in the course of decision making in combat situations. Briefly spoken, the amount of information was vast and diverse.

Luckily there were two external conditions which helped me prioritize the resources a priori. On the one hand my interest was directed towards the field of Human-Computer-Interaction (HCI) and on the other hand I already was able make a raw guess to what extend I could cover the topic at all in the frame of a master thesis. As some articles were little related to HCI and/or too specific to their scientific domain I could filter those out immediately. The rest I started to narrow down into groups. So step by step I distilled some kind of structure from the given sea of information. Interestingly, the

two groups I have had come up with at this point ("user interfaces" and "theory") helped me target my further research towards papers fitting these groups.

However, I could be running into a complete wrong direction if I were not keeping my eyes open for though ill-fitting but yet interesting information. If these *residues* would grow into a considerable portion in my collection, they would make me rethink my structure eventually. And in fact, this kind of *representational shift* happened several times while I was making sense of sensemaking. For instance there are approaches which are similar to sensemaking but still different in certain respects, eg. the SER model (Fischer et al., 1994) or Knowledge Discovery in Databases (Fayyad, 1996). As these were not fitting into the "theory" group although having a theoretical relation to sensemaking I created another group which I named "related". So I was not only building up structure but also refining it constantly.

Of course this sketchy picture of sensemaking is only the tip of it. There were more groups, categories and other kinds of structure involved which I did not create a digital representation for but merely kept in mind more or less consciously. This is due to the volatile nature of these representations. It is easier and less obtrusive to handle the whole process in short-term memory: building a representation on top of given data, finding more data on top of representations and eventually shifting them. However, there are two flaws with this part of human memory which finally render it incapable of making sense of vast and heterogeneous amounts of information. On the one hand its capacity will reach its limits very soon when it comes to complex and big data structures. On the other hand it is short-lived whereas sensemaking is a long-term process in most cases. Therefore, inevitably humans have to externalize representations, eg. take notes, during sensemaking. And that's what I did too apart from putting resources into groups. I just produced a couple of text files on my computer where I recorded additional thoughts, sketched outlines and collected quotes.

However, plain text is far from being the optimal medium for building representations although culture has brought up a couple of common representational codes. For instance, a hierarchical structure might be represented by indentation or sectioning, equivalent pieces of information can be put

into a bullet point list and cohesion is indicated by sentences and paragraphs. But these codes are still subject to the one-dimensional space of text and lack the possibility to represent complex information structures such as graphs, tables and other kinds of drawings. Moreover it is hard to reference and reuse individual textual representations for further sensemaking efforts unless they are written to individual files. Finally plain text notes are most likely not machine readable and hence make any further computer support impossible. In fact they do nothing more than bringing the centuries-old practice of note taking to digital text files with the usual advantages of instantaneous editing, copying and sharing. But in the nowadays digitized world, was that all then?

A new approach is needed, freed from any traditional burdens of the pre-digital era but still grounded in universal qualities of sensemaking, embracing technological advances of the last years and especially catching up with the upcoming Web 3.0. In a lightweight and unobtrusive manner, it has to support the aforementioned sensemaking process of finding representations on top of data, finding data on top of representations and shifting them, making the externalization of representations a matter of only a few clicks while still being machine readable in order to unleash the power of Web 3.0 APIs for semantic content management and recommender systems. So the research question raised by these considerations is:

How can software support sensemaking on the World Wide Web?

1.1. Contribution

In order to answer this question, this thesis will contribute the following:

- A theoretical framework for sensemaking in the context of HCI will be developed.
- Upon this, the design of a generic widget framework for Web-based sensemaking applications will be derived and implemented.

The software framework as well as the results of a user study have been published in the proceedings of the European Conference on Technology-Enhanced Learning (EC-TEL), which took place in Graz, September 2014:

S. Dennerlein, M. Rella, V. Tomberg, D. Theiler, T. Treasure-Jones, M. Kerr, T. Ley, M. Al-Smadi and C. Trattner (2014). **Making Sense of Bits and Pieces: A Sensemaking Tool for Informal Workplace Learning.** In: Open Learning and Teaching in Educational Communities. Ed. by Christoph Rensing et al. Vol. 8719. Lecture Notes in Computer Science. Springer International Publishing, pp. 391–397.

1.2. Outline

This thesis is structured as follows: The first part deals with theoretical approaches to sensemaking from the perspectives of the communication sciences and HCI and distills qualities of sensemaking. In the second chapter, several existing user interfaces for sensemaking on the Web are evaluated with respect to these qualities. Given the theoretical considerations and the evaluation of the related work, "Bits and Pieces", a novel framework approach for Web-based sensemaking applications and its implementation is presented. The last chapter evaluates the framework from a user perspective on the level of the user interface as well as from the viewpoint of experts on the level of implementation, both with respect to the sensemaking qualities.

2. Sensemaking Theory

Theoretical approaches to sensemaking can be found in various sciences, ranging from HCI to military command. Additionally theories exist which have many aspects in common with sensemaking without refering to sensemaking as such. For instance, the SER model (Seeding - Evolutionary Growth - Reseeding) by Fischer et al. (1994) resembles the reciprocal interchange of bottom-up and top-down processes, described later in this chapter. Whereas Knowledge Discovery in Databases (Fayyad, 1996) is more about narrowing down huge amounts of data and finding structures in them, which is similar to the Notional Model of Pirolli and Card (2005) (2.2).

However, the two main approaches commonly cited in the HCI literature are the Sensemaking Metaphor by Brenda Dervin from the communication sciences and the Learning Loop Complex resp. the Notional Model by Russell et al. (1993) and Pirolli and Card (2005) from HCI. These two are not specific to any domain. Sensemaking in Organizations by Weick (1995) and the "data/frame"-theory by Klein et al. (2007) have also been considered for deeper inspection for this thesis. However, Weick (1995) is going too far off the HCI field, and Klein et al. (2007) already take up many ideas from Russell et al. (1993). Nevertheless, a short review of the latter two approaches is given in the end of this chapter.

2.1. Communication Sciences

Brenda Dervin has been doing research on the topic of sensemaking for over three decades. Coming from the Communication Sciences her work has outspread to Information Sciences in general dealing with the question

of how people work with information. Due to its more philosophical implications and due to the influence it had on the other sciences Dervin's "Sensemaking Methodology" shall be discussed first. Moreover Dervin's "Sensemaking Metaphor" provides an easy understanding of the matter for the readers new to the topic.

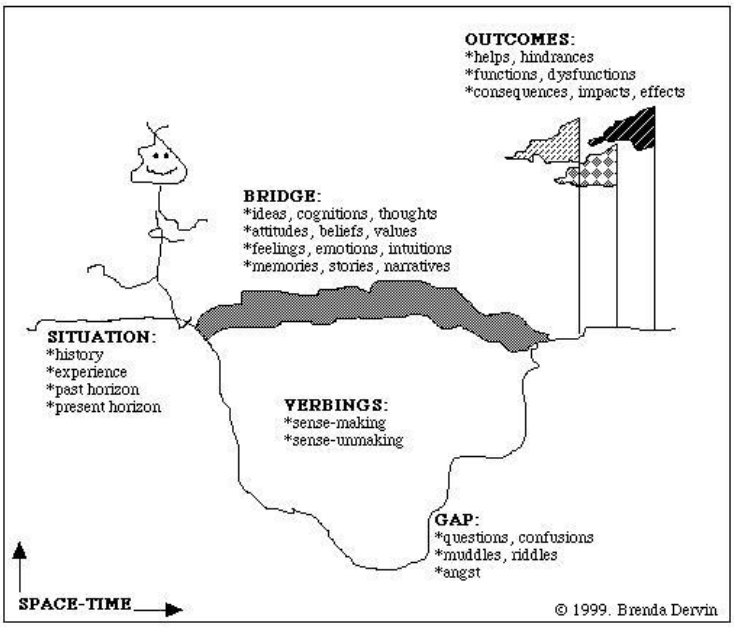


Figure 2.1.: The Sensemaking Metaphor: An individual makes sense of an ordered and chaotic world by moving through time and space and bridging gaps it encounters.

The Sensemaking Metaphor (Dervin (1998), see fig. 2.1) enlightens the term the following way: An individual moves through time and space, taking one step after the other, and needs to bridge gaps it encounters on its strive for a certain goal. The movement is specific to the individual's situation, its history, experience and horizon approaching the goal step by step through continuous exploration. A gap corresponds to questions or confusions which need to be answered by new ideas or thoughts. The outcome finally are certain consequences, helps or functions.

While this metaphor intuitively makes sense its very foundations lie in Dervin's notion of information which has to be stressed for the right under-

standing of her approach. Dervin (1999) proposes to see information as a tool of sensemaking rather than an entity of its own. The latter perspective views information as something which describes an ordered reality and thereby disregards humans totally. The problem with this view lies in the question what that "something" is. Dervin (1999) outlines the history of struggling with that question culminating in the solipsistic conclusion that information "describes an ordered reality that varies from person to person" (ibid. p. 37), which is tautologically based on the fact that "people see the world differently because they differ" (ibid. p. 40). Further approaches to resolve this philosophical dead end finally led to the view that information does not describe an ordered reality but rather creates the order in a chaotic reality. This view avoids the solipsism of different people's different reality by putting information as the means of order. However, it still cannot answer what information is and again disregards humans.

Dervin's notion goes one step behind this question and takes a constructivist communication oriented perspective, "that information is made and unmade in communication intrapersonal, interpersonl, social, organizational and global" (ibid. p. 42f). Her focus lies on the creation of information in the course of an individual's sensemaking of the world: "Information is a tool designed by human beings to make sense of a reality assumed to be both chaotic and orderly" (ibid. p. 39). Hence, in her view, information itself is not the thing that matters, it is the making of it.

That is how the Sensemaking Metaphor comes about: "[H]umans who are themselves ordered and chaotic moving through a reality that is ordered and chaotic. Humans make sense individually and collectively as they move: from order to disorder, from disorder to order" (ibid. p. 41). The result as well as the means of sensemaking is information: It underpins and guides the movement of the individual as well as it lets the individual form bridges over phenomenological gaps in reality. But information always keeps the role of a means to an end, not the end itself. The end, the goal, is what the individual wants to achieve in his or her daily life and work.

In this view, sensemaking is highly contextual, dynamic and grounded in action and must not be reduced to mere solution finding. "The Sense-Making metaphor must be understood as a highly abstract framework" (ibid. p. 46) and can be applied to any kind of work of a certain degree of

complexity, be it in physical reality as well as in mental or virtual worlds, ie. any world in which humans can move. Having this potential levels of abstraction in mind one also has to be aware that movement is not merely meant as a simple linear kind of getting forward. Movement ranges from uncertain padding in the dark to running fast as an arrow. Similarly gaps can be like small ditches or huge clefts, essentially inducing more or less work to bridge them. Finally an outcome might appear as a neat solution or as a subtle hint.

The Sensemaking Methodology is a systematic elaboration of this metaphorical view and has been applied in various real-world information tasks. For instance based on the methodology librarians were introduced to a new set of questions to ask customers in order to help them in their sensemaking of the right book to borrow: "What led you to ask this question? How do you hope to be helped? If you could get the best possible answer, what would it be like? What are you trying to do?" (ibid. p. 49). Questions like these shall support the identification of the customer's "movement", "gaps"/"bridges" and prior "outcomes". Hence, the methodology could also be seen as an action-oriented communication approach for knowledge exchange.

In summary, knowledge and sensemaking are related in such that "knowledge is the sense made at a particular point in time-space by someone" (Dervin, 1998, p. 36), ie. knowledge grounded in action. From this she concludes that one has to be doing something in order to come up with knowledge. However, again, "information and knowledge are rarely ends in themselves; they are means to ends" (Dervin, 1998, p. 40). For information systems this means to let the user do the action and instead of imposing structure a priori: "By freeing our interface with the user from the system's obsession with information and knowledge, we leave users free to define what is informing on their own terms." (ibid.) This finally also means to respect users' individuality in sensemaking: "Collaborative work is necessary to knowledge management, but it is not sufficient" (ibid. p. 41).

Based on this action-oriented, communicative approach to sensemaking the next section narrows the topic down to the interaction between humans and computers and takes it to a more information-technical level.

2.2. Human Computer Interaction

The following approach coming from the field of HCI looks at the sensemaking process from a bottom-up perspective by investigating its cost structure in terms of time/quality tradeoffs. The basic assumption is that the identification of the costly points in sensemaking leads the way to best-practices for HCI design. The two proponents of this approach are Russell et al. (1993) and Pirolli and Card (2005).

Russell et al. (1993) centered their investigation of the cost structure around a case study which dealt with the development of a training course on printing and scanning devices. On top of the technical descriptions of several devices from various companies the goal was to separate common terms and functionality from specific ones in order to unify information and hence making the course structure more concise. As a means of development a hypermedia knowledge structuring tool was used.

Especially as each device was documented over several thousands of pages using non-uniform terminology this case is interesting for the study of sensemaking and its relation to information retrieval tasks. Russell et al. (1993) define this relation as being "... best understood in their embedding in a larger overall task structure. The larger task often involves sensemaking, the process of encoding retrieved information to answer task-specific questions" (Russell et al., 1993, p. 269).

The authors argue that this process is guided by the tradeoff between the time a certain operation consumes (the cost) and the expected gain in quality of information, by which they mean quality with regard to the reduction of costs to approach the goal of the overall task. Furthermore, the resulting cost structure of sensemaking leads the way to identify operations or parts of operations which may profit from automation in the human-computer interaction.

Learning Loop Complex

According to Russell et al. (1993) the operations involved in sensemaking can be depicted by the so-called "Learning Loop Complex" (LLC, Fig. 2.2).

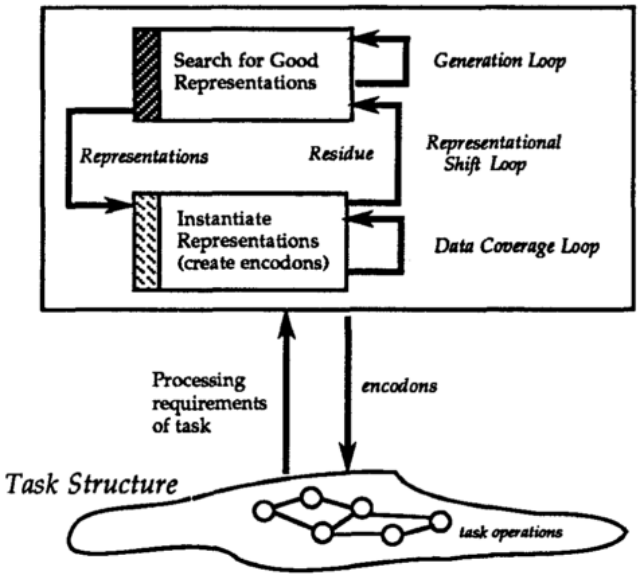


Figure 2.2.: The Learning Loop Complex (Russell et al., 1993, p. 271)

As defined above, the goal of sensemaking is to find appropriate encodings (or "encodons" as they call it) of information which "answer task-specific questions" and hence reduce the cost to achieve a certain task.

There are three core operations: generate, instantiate and shift representations. A representation in the sense of Russell et al. (1993) is a kind of pre-stage of an encodon, a volatile structure imposed on the information at hand which may be kept in the sensemaker's memory or in some external medium. The use of the latter grows with the size and complexity of the task in order to relieve cognitive resources.

The Generation Loop involves the search for "good representations" by qualifying how well (or better) the data fits with regard to the task's goal and with regard to existing instantiations of representations. After having tried to cover more and more data (Data Coverage Loop) by instantiating the found representation this loop may yield residues, ie. features of the data which could not be covered well, which in turn drive the need to shift the representation by generating new or adapted versions (Shift Loop).

Finally, when the appropriate representation, the encodon, is ready, it can be integrated in the task structure to ease the further strive for the goal.

The LLC embraces a core characteristic of sensemaking: the iterative interchange between bottom-up and top-down processes. Data drive the bottom-up search for representations in the Generation Loop, representations drive the top-down search for data in the Data Coverage Loop. The Shift Loop is the crucial part in the model of Russell et al. (1993): it connects the two other operations and hence corresponds with the "iterative interchange", which is the anchor point in the general notion of sensemaking.

The identification of residues means to pin down the parts of a representation which need improvement. Thus, the Shift Loop is the operation which actually leads to gain in quality and hence to reduction of cost. Russell et al. (1993) illustrate this by a short example of another case study:

> "In one of our case studies, the sensemaker looks up data about laptop computers in a collection of magazines and product sheets. His goal is to make a purchasing recommendation meeting given constraints. The data representation created by sensemakers carrying out this task invariably includes tables giving properties of competing laptops. Representation shifts are changes to the table structure as the sensemaker decides which properties are most relevant and retrievable and ultimately are able to help solve the problem of determining the best choice." (Russell et al., 1993, p. 275)

As the Shift Loop is not the costly but the eventually cost-saving operation it are the Generation and the Data Coverage Loop which impose big load on the sensemaker. These involve information retrieval and data extraction, which "is often the most time-consuming task in sensemaking" (Russell et al., 1993, p. 273). For instance in the aforementioned printer case study, all textual description of the devices spanned several thousand pages which had to be read in several iterations in order to extract relevant data.

This is the phase where automation may reduce costs most effectively. On the one hand, clustering algorithms can help in finding similarities and in identifying broader concepts for instance. This can ease the pain of finding representations which is foundational to identifying residues. The

representation shift then may consist in adapting the algorithm's parameters by the sensemaker or even by switching the algorithm itself. On the other hand from a top-down perspective, the instantiated representations could be used to compute recommendations to guide further data exploration.

These automation tools can only calculate the best solution given the current data available at any time during the sensemaking process. They cannot provide the ultimate, or global, solution to the sensemaking task, because it is always up to the sensemaker to take sophisticated decisions when shifting representations. This is the so-called "anytime principle".

The Notional Model of the Analyst's Process

Pirolli and Card (2005) take up Russell et al. (1993)'s approach by investigating again the cost structure of sensemaking. In contrast to the former the authors separate the actual sensemaking process from the information retrieval on a meta-level. In their view, the overall development process from the raw data to the goal consists of the information foraging loop in the first place and the sensemaking loop in the second place. Consequently this distinction empowers them to devide the analysis of the leverage points, ie. the costly points of the process which could be eased.

The Notional Model of the analyst's process depicted in Fig. 2.2 can be seen as an derivative of Russell et al. (1993)'s LLC by giving names to the succeeding stages ("shoebox", "evidence file", ...). These notions, which have been identified in a preliminary study of the work of business intelligence analysts (Pirolli and Card, 1999), are framed by two scales: structure and effort, ie. through the analyst's effort data becomes structured. So the model explicits implicit assumptions of the LLC for the case of intelligence analytics. However, the identification of stages could be applied to other domains as well. The point here is that the LLC is a general model of sensemaking which can be taken down to concrete problem fields or even specific problems. This can also be seen by the fact that the most important property of the LLC is still prevalent in Pirolli and Card (2005)'s Notional Model: its symmetry between top-down and bottom-up processes. Again it is the reciprocal interchange of these processes which drives the overall process.

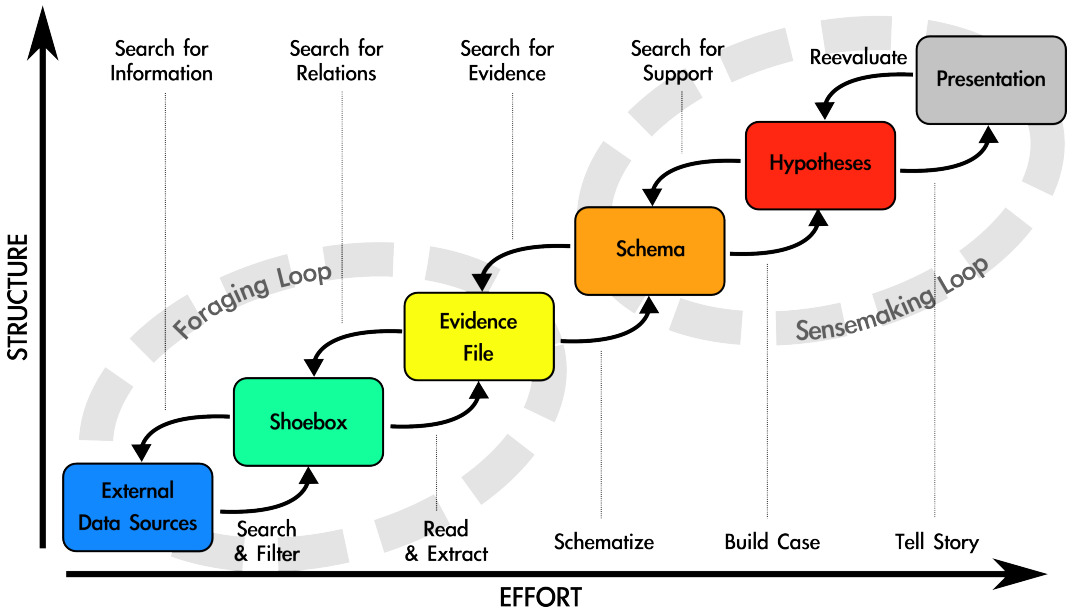


Figure 2.3.: The Notional Model of the analyst's process (Pirolli and Card, 2005)

It all starts with the raw data sources which are external in the sense that they are not part of the information created through the process. Analysts use "search and filter" technology to retrieve a "shoebox" of data relevant for their task. As the metaphor suggests this is a mere subset without deeper structuring. They only share a common measure of relevance which may be adapted by the top-down process of finding more or other information, ie. refining the relevance criteria.

The shoebox data serves as the foundation to "read and extract" the "evidence file" which is a collection of approaches to the goal. With the evidence at hand the shoebox can be searched for more or other relations in order to skim previous indications. The evidence file then gives rise to "schemata" which aid in representing complex structures. In turn, these schemata are refined by looking back at the evidence. This stage of the overall process is essential as it lies at the passage from the information foraging loop to the sensemaking loop. Pirolli and Card (2005) put the interchange of bottom-up and top-down processes at this stage by example: "Evidence may

be organized into small-scale stories about typical topics or in answer to typical questions (eg., who, what, when, where, why, how) that are used to organize raw evidence" (p. 3). The first example can be seen as a bottom-up process where evidence is stitched together whereas the second example imposes a question with a schematization in mind.

During the sensemaking loop "hypotheses" are "built" from schemata on the one hand. On the other hand, given an hypothesis "support needs to be searched" for it by looking into the schematic organization of the information again. Finally, on top of the hypotheses a "story" has to be "told" to others through a presentation or publication. In turn, these clients could pose remarks on the presented theory which would induce "re-evaluation" of the hypotheses.

Although this summary of the Notional Model might give the impression that these stages are discrete, they are not. For instance, "schema-based expert skill can be used at all points in the process [...], for example in rapidly skimming and rejecting information in the early stages" (Pirolli and Card, 2005, p. 3). Hence the borders between stages are blurred rendering the whole process a continuum rather than a stepwise way of structuring. However, it is still worthy to distinguish between the stages and the two loops because they help at naming the costly pain points which "structure user information behavior [...] and can often been altered (positively or negatively) by compute or methodological innovations." (ibid. p. 4). Therefore the task of user interfaces for sensemaking is to hook supportive visualization or tools into these leverage points where they are most effective.

2.3. Qualities of Sensemaking

There are many more theoretical approaches to sensemaking which cannot be tackled in detail in this thesis. For instance, Klein et al. (2007)'s data/frame-theory makes a similar point as Russell et al. (1993)'s Learning Loop Complex: Data "are explained when they are fitted into a structure" whereas a Frame denotes an "explanatory structure that defines entities" (Klein et al., 2007, p. 118). It also lends from Dervin's human-centered action-oriented approach, when stating that "[s]ensemaking begins when

someone experiences a surprise [a gap, *note from the author*] or perceives an inadequacy in the existing frame and the existing perception of relevant data." (ibid.)

Another perspective from the Organizational Sciences is also worth mentioning: Weick (1995) boils sensemaking down to seven properties: "1. Grounded in identity construction, 2. Retrospective, 3. Enactive of sensible environments, 4. Social, 5. Ongoing, 6. Focused on and by extracted cues, 7. Driven by plausibility rather than accuracy" (Weick, 1995, p. 17). Since the author puts focus on sensemaking in organizations he stresses different aspects than Russell et al. and Dervin. For instance, the Sensemaking Metaphor assumes "forward movement" rather than attention being "directed backward from a specific point in time" (Weick, 1995, p. 26). The social aspect is also covered just implicitly by Dervin when subsuming "culture and society" into the individual's situation. However, the other properties can be found with the approaches discussed in the latter two sections. For instance, the idea of "extracted cues" can be aligned with Pirolli and Card (2005)'s stepwise Notional Model.

Here Weick (1995)'s style of pinning sensemaking down to seven properties shall be taken up in order to streamline the theoretic rationales of this chapter. For the matter of distinction these pin-points are called "qualities of sensemaking". These qualities serve as a theory-grounded foundation for the generic widget-framework for sensemaking presented in chapter 4. These qualities of sensemaking are the following:

- **Individuality:** Sensemaking is an individual and indivisible process. According to the Sensemaking Metaphor it's an individual moving through time and space, coming from a personal horizon of experiences and always "self relating to self; self relating to another; self relating to a collectivity" (Dervin, 1999, p. 41). Hence sensemaking occurs if and only if there is one sensemaker.
- **Context-sensitivity:** How sensemaking works in reality highly depends on the context. The Sensemaking Metaphor makes that clear in saying that the individual's current situation (consisting of time and space, history, etc.) is the source for the next move. From an informational point of view it's the "data" in general that matters here. Furthermore, Russell et al. (1993) defines sensemaking to be specific

to task and question. Therefore, sensemaking is always sensitive to its context.

- **Bipolarity:** On a meta-level sensemaking runs between two poles: information foraging and information organization. As Pirolli and Card (2005) state, in the beginning sensemaking is more concerned with finding the right data and narrowing it down to more handy representations. Later more and more organizational processes occur which structure the data in itself instead of bringing new data in. Although the other approaches do not come up with this point, it is especially important for the application of the sensemaking paradigm to HCI, ie. the design of computer-supported sensemaking systems. Hence, bipolarity needs to be taken into account as a quality of sensemaking.
- **Reciprocity:** At its core sensemaking is driven by interchanging bottom-up and top-down sub-processes, ie. finding representations on top of data vs. finding data in the view of representations. The LLC consists of these two processes and the "data/frame"-theory of Klein et al. (2007) also makes reciprocity to its defining property. Dervin (1998)'s Sensemaking Metaphor is also driven by the interchange of "movement" and "bridging of gaps". Movement works top-down, on top of reality and information, whereas the bridging works bottom-up by building a representation for the data in order to move on. Therefore sensemaking definitely consists of the reciprocal interchange of top-down and bottom-up processes.
- **Continuity:** Sensemaking is a continuous process, starting or stopping at any point. There is no need for the data to be totally raw as well as the result needs not reside on a certain level of abstraction. It depends on the "task-specific question" and the coming about of "representations" when sensemaking begins and ends (Russell et al., 1993). However, representations might serve as input for another sensemaking with no clear separation between the latter and the forth. The Learning Loop Complex obviously visualizes that fact by the circular dependency of its sub-processes. Hence, sensemaking has to be considered as "ongoing" (Weick, 1995).
- **Enactivity:** Borrowing the term from Weick (1995) it states that sensemaking is grounded in action, ie. "not in how humans [...] see their worlds but in how they 'make' their worlds" (Dervin, 1999, p. 40). It's a constructivist and enactive process, which means that sensemakers

"produce part of the environment they face" (Weick, 1995, p. 30). Additionally Russell et al. (1993)'s and Pirolli and Card (2005)'s dynamic, reciprocal approach of knowledge evolution suggests a big deal of interactivity with the data. Finally the term "sensemaking" itself is a verb, ie. something to be done. So enactivity plays an essential role in sensemaking.

- **Reification:** During sensemaking representations are constantly reified in order to serve as input for further sensemaking. Dervin (1999) says that information is a "tool" for sensemaking. Hence, information can be understood as representations which help in sensemaking in the sense of Russell et al. (1993). Further representations might reuse them as part of their structure and thereby reify them. This leads to the evolution of structures (meta-data) on several levels of abstraction (Pirolli and Card, 2005). Its the quality of reification (of any representation) which gives sensemakers the freedom "to define what is informing on their own terms" and hence plays a crucial part in it.

These seven qualities will serve as a measure of sensemaking for related work on existing user interface designs as well as a guideline for the development of this thesis' generic widget-framework for sensemaking.

3. Related Work

A couple of desktop applications for sensemaking on the Web exist. This section reviews the most elaborated ones which differ widely in their sensemaking support. Although more work in this field exists the selection was made according to the following features: First, the approach needs to be designed primarily for working with data from the Web. Second, the display dimensions of the target device should match those of desktop computers. Third, the work needs to address sensemaking explicitly instead of mere information collecting or visualizing. So the selection of approaches reviewed here is the following:

- ScratchPad (Gotz, 2007): a browser plugin which enables users to interactively collect Web resources (bookmarks, images, text fragments) and which provides tools for adding notes, creating folders and linking collected items individually.
- Coalesce (Ryder and Anderson, 2009): a Web application featuring interactive and fine-grained information foraging tools together with SenseMap for structuring items hierarchically.
- CoSense (Paul and Morris, 2009): a dedicated desktop application for sensemaking adding collaborative features supporting various interactive views and visualizations of the evolved information.
- Apolo (Horng et al., 2011): a dedicated desktop application which visualizes data of scientific publications through citation graphs fetched from Google Scholar and enables users to arrange nodes interactively to own needs. The tool provides mature filtering mechanisms and implements a recommender engine guiding the sensemaking.

Each of these are evaluated in comparison to the qualities of sensemaking. This systematic method of sensemaking assessment provides useful insights

into respective advantages and disadvantages and has not been applied by former approaches to sensemaking.

3.1. ScratchPad

The ScratchPad (Gotz, 2007) is designed as an extension to the sidebar of the browser window. On the one hand this pad serves as a collection area: While browsing the Web, users can create snapshots of the current Web site or drag and drop fragments of it (images and text snippets). Irrespective of the granularity, the collected items are displayed graphically in the ScratchPad. On the other hand there are various tools for manipulating the collection independent of the Web site's content: For instance folders can be created which help in organizing the collection, textual notes can be added "to express insight created by the users themselves as the sensemaking task evolves" (p. 1330) and related objects (website snapshots and/or fragments) can be joined interactively by drawing links between them. Additionally given these kinds of user-generated input the relevance of elements in the ScratchPad to the currently browsed Web site is calculated. The results are indicated either visually next to the elements or structurally in a table at the bottom of the pad (fig. 3.1).

Cheng and Gotz (2008) even improved this relevance detection algorithm as follow-up work to ScratchPad by incorporating context-based page unit recommendations with the context consisting of the user-created structure and notes. In addition to the relevance indication with the pad itself thus relevant portions of a Web page are highlighted also. As these recommendations are based on the user-collected elements in the ScratchPad they are likely to be specific to the user's sensemaking task. As the authors proof in a user study this extension enhances the user's information foraging significantly.

In the light of the qualities of sensemaking ScratchPad cuts a good figure. First, *individuality* is fulfilled as the approach enables the user to build up personal information collections and structures. Given that the built-in relevance detection algorithm additionally tries to give personalized recommendations potentially fitting the user's specific task. Furthermore,

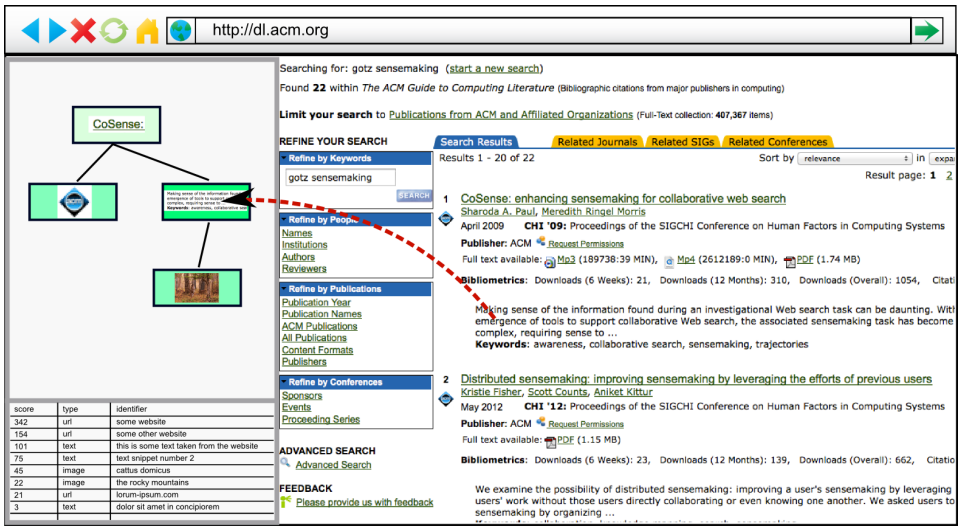


Figure 3.1.: Schema of the ScratchPad design according to Gotz (2007). By extending the browser window to the left it provides a collection/sensemaking area with a table of information relevance scores below.

enactivity is enabled through dragging/dropping gestures which let page units being added to the pad as well as relations between collected elements being created. As the approach is implemented as a browser plugin it is likely although not explicitly stated that the ScratchPad can be activated and deactivated at any time while browsing which fulfills the quality of *continuity*. As for the *reciprocity* of sensemaking ScratchPad essentially allows for building representations on top of collected data through its features for folder and link creation on the one hand. On the other hand it more data on top of the user-generated representations can be found by having the representation always visible in the sidebar. Additionally recommendations for page fragments also support the top-down aspect of reciprocity. The constant interchange of these two processes and their mutual stimulation come up to the reciprocal quality. Finally, ScratchPad self-evidently implements the *bipolarity* of sensemaking as it strictly divides the information foraging aspect from the organizing aspect of sensemaking by the sidebar-oriented design of the user interface. While the browser window itself remains unchanged in its usual functionality for information seeking, the sidebar pad allows for the information structuring and representation building.

However, *reification* is missing. Representation building is only possible on one level of abstraction, ie. folders and links. For instance, folders themselves cannot serve as objects of linking and links as such cannot be structured in folders. Furthermore, clusters of interlinked elements cannot be reused as a whole in further sensemaking tasks. Therefore, ScratchPad fails at reification of its outcomes. There could also be more about *context-sensitivity* as for instance exploiting features in the data such as location- and time-based properties. Additionally, it is questionable whether folders and links as the only forms of representation are sufficient for every kind of sensemaking task. Neither Gotz (2007) nor Cheng and Gotz (2008) allot features for extending the possibilities of building representations. Hence, there might be contexts were ScratchPad renders inadequate.

Dragging and dropping are a good start in providing *enactivity* but ScratchPad lacks other interactive features which might be expected from the interface. For example, the authors don't mention any way of viewing the contents of an element instantly although there is the capability to modify elements which is "roughly analogous to the functionality of traditional bookmark organization tools which allow you to modify titles and contents of bookmarked items" (p. 1330). Finally, it is unclear whether the state of sensemaking is preserved when the user closes the browser and comes back later for continuing the sensemaking process which would violate the quality of *continuity*.

3.2. Coalesce

Similiar to ScratchPad Coalesce (Ryder and Anderson, 2009) is implemented in a Web browser environment using the Google Web Toolkit. In contrast to the former approach, Coalesce comes as an integrated Web application without allotting space for native Web browsing. Instead data search and retrieval is achieved via numerous Web search APIs. By integrating these remote services Coalesce provides an HTML based presentation of the fetched results in a dedicated search tab. This tab can be activated instead of a snippet manipulation and another tab for viewing the full contents of SenseMap.

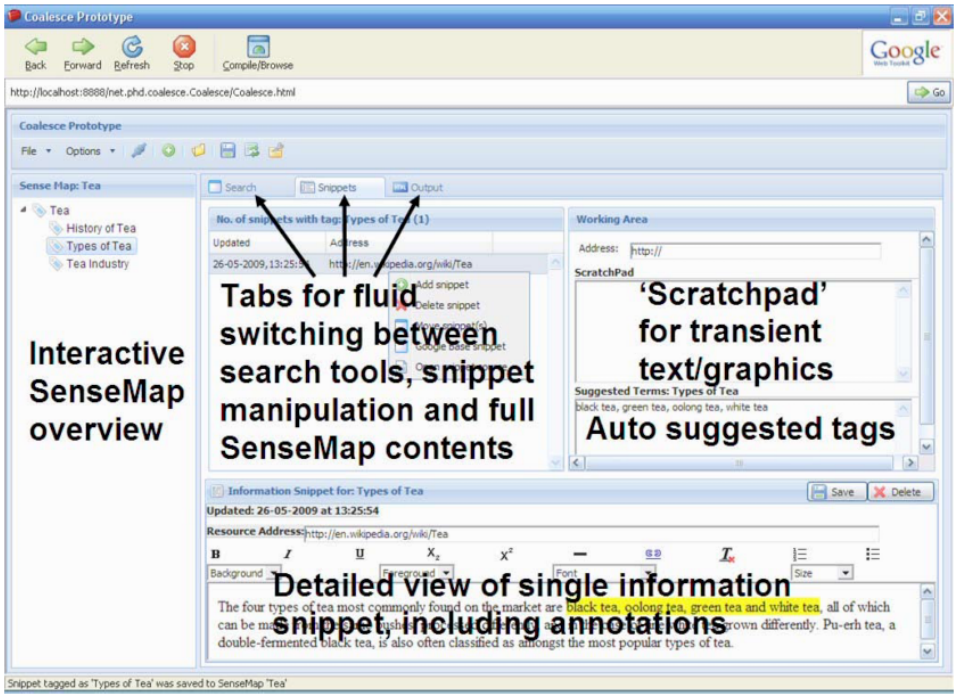


Figure 3.2.: Coalesce user interface (Ryder and Anderson, 2009) providing various views including the SenseMap, tabs for search, manipulation and overview, a working area and a detailed view.

The SenseMap area to the left enables users to structure information portions (text fragments from Web pages) in a tag-based hierarchy. This hierarchy can be edited in-place. Tags can be created by the user manually or recommended through kinds of Web services for content analysis in the so-called "scratchpad" (right-hand side of the interface). Recommendations are based on user-generated tags and the tagged information source itself. In the bottom of the interface a detailed view of a singe information snippet is provided (fig. 3.2).

Data items are stored as small XML documents in Google Base, a key/value based online storage. The type of the value can be defined either as one of the standard types provided by the storage engine or a custom one. For the representation of the application data "SenseMap" and "Information Snippet" have been invented as custom data types. The first stores hierarchical

data, the latter portions of content from Web pages.

As the former approach, Coalesce effectively supports the construction of individual information structure, thus accounting for the quality of *individuality*. As for *context-sensitivity*, offers a limited set of views on the data which enable users to adapt the application to own needs. *Enactivity* is served by "fluid switching" between tabs and in-place context-menus for manipulation operations. As users can "modify and save any aspect of the hierarchy continuously as they proceed" (p. 291) *continuity* of sensemaking is also satisfied. Having a set of views at hand users may easily proceed their search while having a certain custom SenseMap on the screen which enables one side of sensemaking *reciprocity*. On the other side Coalesce also enables the parallel building of a representation (the tag hierarchy) based on the retrieved data in view. As there is a search tab which needs to be activated on demand the prototype also reflects the quality of *bipolarity* to a certain extent.

However, as with ScratchPad, *reification* is not implemented in this prototype. Google Base having a typed key/value data structure as described by Ryder and Anderson (2009) cannot serve arbitrary reification of data to meta-data. Coalesce defines two custom and fixed data types which can only describe data on one single level of abstraction (information snippets with relation to SenseMaps). Time- and location-based features of the data could also be recognized with respect to *context-sensitivity*. Furthermore, Coalesce cannot be considered very *enactive* since all views are text-based with the main user input device being the keyboard. Concerning *continuity* the authors state that the SenseMap is saved continuously which allows for ceasing and resuming the sensemaking. However, it is unclear how the rest of the application is treated in this aspect since they also need to be seen as substantial parts of the sensemaking process. Finally, little is done for *bipolarity* as the search tab only plays a minor role in the whole user interface.

3.3. CoSense

Based on a formative user study on collaborative Web search Paul and Morris (2009) developed CoSense for collaborative sensemaking. Since sense-

making is concerned with a user's individual interaction with information collaboration only affects the sensemaking process with regard to this information. The process as such with regard to its qualities does not change in spite of the input of other people which merely adds to the data to make sense of. Paul and Morris (2009) put it like this: "Sensemaking is an integral part of the information seeking process; for groups, this sensemaking encompasses both the need to make sense of found information as well as the need to make sense of the collaboration process, such as group members' roles and task state." (Paul and Morris, 2009, p. 1779).

Indeed, findings from the study even confirm some of the theoretical viewpoints of sensemaking. First, the authors found out that users want a tool for collaborative Web search make them aware of the actions of other users as well as of the context of these actions. These requirements go in hand with the action-based perspective from Dervin (1998) which considers questions like "How did it evolve?", "What happened to it?" or "What were the gaps? What are the bridges?" central for sensemaking. The second finding highlights the importance of time-based cues for sensemaking as users want to get a grasp of the chronology of a certain piece of information. Time also plays a central role in the sensemaking metaphor of Dervin (1998) and is the most prevalent feature used for giving structure to data. Finally persistence of the sensemaking process and results has also been found essential by having explicit means for noting these down, eg. as meta-comments.

These findings partly reflect qualities of sensemaking: *Enactivity* seems to be encouraged by the need for action-awareness while context-awareness and time-based structuring account for the quality of *context-sensitivity*. As users found it important to note down outcomes of sensemaking instantly which keeps them available for later *continuity* can also be seen granted by the study of Paul and Morris (2009). Since the authors mention "meta-comments" as a means of persisting results or aforementioned meta-information like "group members' roles and task state" which evolves with collaborative sensemaking they also seem to indicate *reification* as a requirement.

CoSense is implemented as a dedicated desktop application particularly addressing these findings. It works in junction with SearchTogether (Morris, Lombardo, and Wigdor, 2010), a tool for collaborative Web search, where users can browse and search the Web and comment and rate pages. There

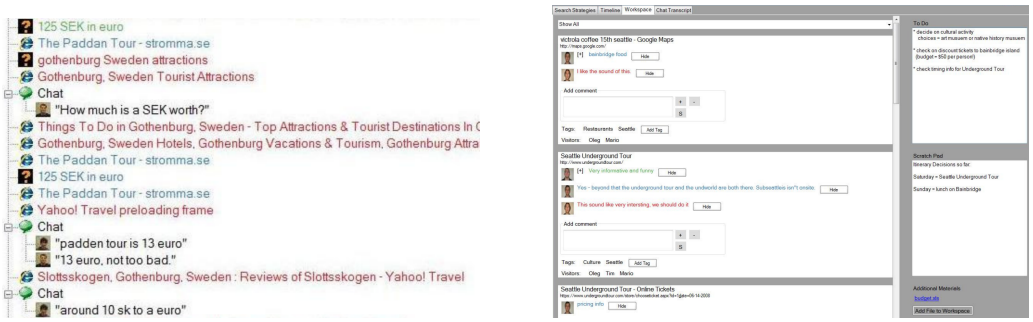
(a) Block diagram and tag clouds (b) Timeline of query history

Figure 3.3.: The search strategies view of CoSense (Paul and Morris, 2009) provides a query- or URL-based view of a block diagram of total amounts and respective tag clouds. Hovering/clicking on a keyword or URL shows the associated queries or Web pages. The timeline of query history of each user visualizes the evolution of the collaborative Web search.

are four views available: The search strategies view, timeline view, chat-centric view and workspace view. As the name suggests the search strategies view aims at supporting action-awareness. On the one hand it consists of a block diagram visualizing the total amount of search queries issued by each user of the group. On the other hand there are tag clouds of query keywords of each user and of the group as a whole. The diagram and the tag clouds can be switched to be based on visited website URLs. Additionally words in the tag clouds are directly associated to the respective search queries or Web pages which can be viewed instantly in a Web browser by clicking on them. The search strategies view also provides a timeline of queries in order to visualize the evolution of the group's collaborative Web search (fig. 3.3).

The second tab of the application contains the timeline view which lists all kinds of actions of the group chronologically, ie. search queries, Web page visits, chat messages and comments, or a subset of them by applying filters. Moreover Web page items can be clicked to bring up associated information, ie. the context of the item, to the right of the timeline. This includes chat messages in the sphere of the Web page visit, users who have visited the page, comments and a preview of the page. Apart from mere browsing of this contextual information users can also create and associate additional comments to the Web page (fig. 3.4).

Besides the chat-centric view which brings up Web pages associated to



(a) Integrated timeline (b) Workspace with "summary items"

Figure 3.4.: The timeline view integrates various kinds of actions in one place chronologically. The workspace allows for building tag-based structure and finalizing sensemaking outcomes as textual notes (Paul and Morris, 2009).

a certain chat-message (as the inverse of the timeline functionality), the workspace view aims at persisting sensemaking outcomes. Each Web page visited during the collaborative Web search is listed together with associated comments and the users who visited it. Web sites without comments are omitted. These "summary items" can be tagged and filtered by tags instantly. Additionally individual comments can be hidden. These "richer sensemaking features" and "means of organizing the workspace" allow for "imposing structure on collections of links and comments" (Paul and Morris, 2009, p. 1776). The workspace also facilitates creation of arbitrary to-do lists and textual notes in two dedicated text fields as well as the upload of external files from the local computer (fig. 3.4).

From the standpoint of the qualities of sensemaking CoSense in junction with SearchTogether hits a quite high score. First, *individuality* is ensured by the various possibilities to view and add individual information such as the user-based tag-cloud of search queries or Web page URLs as well as comments. Above all user's search data is constantly imported from SearchTogether. A big deal of CoSense features is explicitly pointed at providing "context-awareness" which accounts for the quality of *context-sensitivity*: on a low level by using time-based cues for presenting information such as in the query history view of the search strategies tab or in the integrated timeline view, on a high level by providing contexts of chat messages, comments

and visits by peer users when viewing the preview of a Web page listed in the timeline view. Since this view also shows other kinds of actions (commenting, querying, chatting etc.) in order to meet the "action-awareness" distilled by the preliminary study it also supports sensemaking's *enactivity* on a semantic level. On the level of the user interface this quality is achieved by rendering items interactive in the search, the timeline and the chat-centric view, ie. yielding instant access to associated information by clicking on them. Additionally, the timeline can be filtered by the type of items. Although stressing a collaborative approach CoSense also takes into account *continuity* by addressing "sensemaking handoff" between peer users. One user needs to continue sensemaking by taking up the outcomes made persistent in the workspace view by another user. This feature would also work in the individual case when a user has to continue the own sensemaking process. Furthermore, it seems that data generated throughout the sensemaking session (eg. comments or chat messages) is saved continuously in the background.

As for the *reciprocity* of sensemaking there are the four different views which can be switched easily (in analogy to "shifting representations"). Since CoSense is also made for making sense of other's actions the timeline and the chat-centric view provide means for data-driven representation finding while the tag-clouds are representations which enable focussing further data search, for instance for highly prominent queries of other users. Additionally the tag-clouds allow for refining queries for SearchTogether which in turn provides means of finding representation (comments, ratings) on top of data. Having this tool working seamlessly with CoSense while both still being separate also adheres to the quality of *bipolarity*. Finally, there also are rudiment features for *reification* since the workspace view provides taggable summary items which form a layer of abstraction on top of the comments and Web pages being treated in the other views. As tags are applied to these summary items they reify them and make the workspace organizable.

Although CoSense tackles qualities of sensemaking to a broad extent there still are some shortcomings. First, it is unclear whether the application can be extended to present other kinds of contextual cues which would be necessary for the possible complexity and specificity of a sensemaking task which cannot be known in advance. There might be more and different views

needed. Furthermore, the only means of interaction on the level of the user interface is clicking on items although dragging and dropping would also be a common way of interaction which is, however, not provisioned by CoSense. A small drawback for the *reciprocity* is that views or features for top-down and bottom-up processing cannot be visualized in parallel which hinders the immediate reciprocal interchange of them. However, views can be switched instantly by tabs in the top of the application. Concerning *reification* there are only limited and pre-defined means for it in the workspace (summary items and tags) although other reifications might be possible as for instance tagging/rating of comments/users or having creation of outcomes (textual notes) visualized in the timeline view.

3.4. Apolo

Apolo (Horng et al., 2011) is a highly interactive and visual Java-based desktop application specifically made for sensemaking of networked data, ie. scientific citation networks in its current implementation. The user interface of Apolo consists of tree areas: one for the workspace where the citation network is visualized as a graph, one for configuring this visualization and one for filtering and grouping of nodes. Each node represents a scientific publication with directed edges to other nodes depending on the direction of the citation relationship ("cites" vs. "cited-by") (fig. 3.4). The network data is loaded from Google Scholar[1] before runtime.

Users can arrange nodes spatially within the workspace while a force-directed algorithm avoids mutual occlusion of nodes. Moving the mouse over a node brings up a box with details about the publication such as the full title, authors, publication year and citation count. The box also contains optional operations: One allows for starring the publication, one for adding annotations, one for toggling the visibility, one for toggling membership to a group and the other for pinning and unpinning the node to the current position on the workspace (fig. 3.6(a)). Pinning nodes releases them from the custody of the force-directed layout algorithm and enables users to create visually fixed clusters. Additionally, selected nodes can be "ranked in place"

[1] http://scholar.google.com (Visited: 2014-10-25)

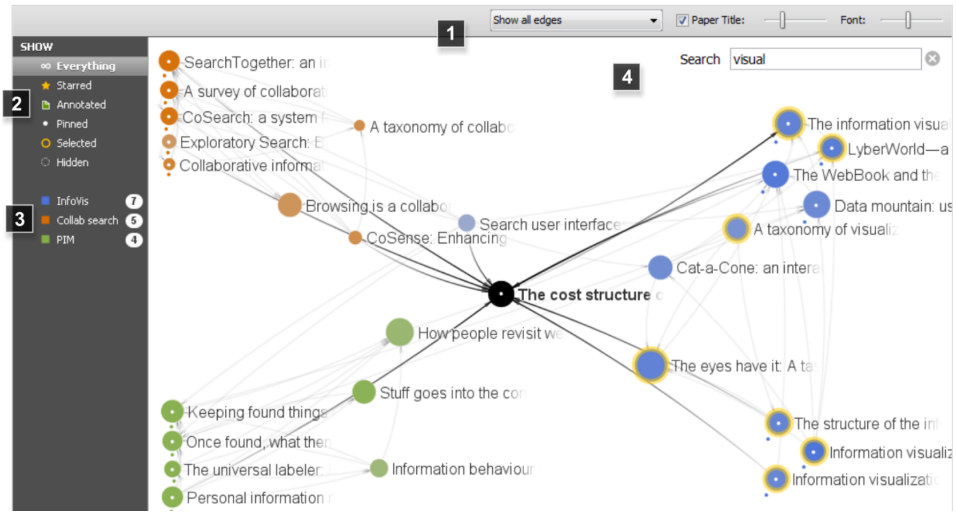
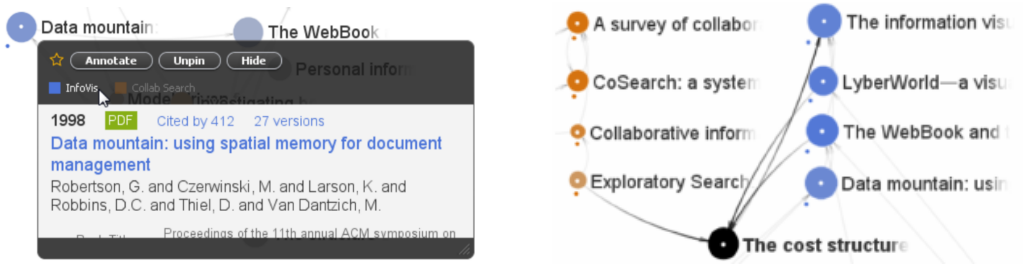


Figure 3.5.: The Apolo user interface with three areas: The configuration area (1) to adapt parameters for the visualization of the citation graph below (4) which shows publications as nodes and citations as directed edges. The filter (2) and grouping area (3) show options for filtering of nodes and for managing coloured groups of publications.

which arranges them vertically within the graph (fig. 3.6(b)). The workspace also contains a search field for highlighting nodes which match the input keyword.

The grouping area to the left of the workspace shows the user-generated group labels and colours. From within the detail box, a publication can be set to be a prototypical example of a group which makes the node appear in the corresponding colour. A similarity algorithm automatically calculates which of the other visible nodes are similar to the prototypical one and colours these according to the degree of similarity. By changing the membership in the detail box, the similarity is recalculated and the visualization re-rendered instantly. On demand, users can load more publications into the workspace which are similar (ie. relevant) to a certain group's prototypes. The grouping area also allows for filtering nodes by their membership to groups. However, filters can also be applied on an application level, eg., whether a node is pinned, annotated or starred.

Looking at Apolo with respect to the qualities of sensemaking the prototype



(a) Detail view of meta-data (b) Nodes ranked in place

Figure 3.6.: The detail view provides meta-data of a publication and several operations. A set of nodes can be ranked in place and arranged vertically.

yields several positive features. As for the *individuality* of sensemaking undoubtedly enables users to build up personal representations, "[...] that even if two users' landscapes included the same nodes, those landscapes could be very different based on their goals and prior experience" (Horng et al., 2011, p. 172). Concerning *enactivity*, "rich user interaction" plays a crucial role in Apolo. There is the visual workspace of the citation graph which can be edited intuitively by moving nodes around and pinning them on demand. Mousing over a node immediately brings up a detail view together with further possible actions. The filter options as well as the configuration toolbar allow for instant adaption of the visualization. Furthermore, "[a]s a node's group membership can be 'toggled' easily, the user can experiment with moving a node in and out of a group to see how the relevance of the other nodes change" (Horng et al., 2011, p, 172).

Continuity is ensured as the positions of pinned nodes can be preserved by saving the application and loaded again on resuming sensemaking. Apolo's fine-grained interaction features with the visualization also makes continuous sensemaking possible. Noteworthy, the authors have put special focus on "shifting representations" and thus account for *reciprocity* as a representation such as a group membership can be changed easily which leads to a different colouring of the nodes. It is also possible to retrieve more relevant data on demand given the current representation (top-down, by the similarity detection algorithm) as well as to build up representations given data by moving publication nodes around (bottom-up). On demand retrieval of data in contrast to organizing of data in the workspace can be

seen as an implementation of the *bipolar* quality of sensemaking.

However, although Apolo reaches high scores on the *individuality*, *enactivity* and *reciprocity* two important qualities are even missing. Since the prototype seems to be confined to citation network data a priori it yields no effort for supporting other kinds of data which might be specific to the user's context. Hence, Apolo lacks *context-sensitivity* completely. Consequently, due to the limitation in data types there aren't any means of creating meta- and meta^2-data, for instance, allowing for sensemaking upon groups as such. Thus, *reification* is also missing. As for *continuity* it seems that the application state is only preserved on explicitly saving it which might hinder starting and stopping sensemaking at *any* point. Apolo also hardly provisions any features for information foraging since data is selected and loaded by relevance through the similarity algorithm. However, Apolo puts comparably more focus on the organizing side of sensemaking's "bipolarity". There only remains the possibility of browsing nodes' meta-data by mousing over them, which could be taken as an implementation of its other side.

3.5. Summary

This section reviewed four approaches for user interfaces for sensemaking on the Web by depicting the functionality of each in detail and evaluating it in the light of the qualities of sensemaking. Table 3.5 summarizes the results on a scale of one to three.

CoSense (Paul and Morris, 2009) outperforms the other three user interfaces significantly by implementing features that match all qualities of sensemaking to a certain extent, even *reification* which could not be found in any of the other approaches. This due to their preliminary study which directed the implementation towards *context-sensitivity* and *enactivity*. By taking into account collaboration and sensemaking handoff CoSense puts particular effort into *continuity*. Multiple views together with easy switching between them mainly accounts for *reciprocity* and having SearchTogether as a separate but coordinated search tool clearly supports the *bipolarity* of sensemaking.

	ScratchPad	Coalesce	CoSense	Apolo
Individuality	+++	+++	+++	+++
Context-sensitivity		+	++	
Enactivity	+	+	++	+++
Continuity	+	++	+++	++
Reciprocity	+++	++	++	+++
Bipolarity	+++	+	+++	+
Reification			+	

Table 3.7.: Comparison of the reviewed user interfaces for sensemaking on the Web: CoSense and Apolo (Horng et al., 2011)

ScratchPad (Gotz, 2007) leverages a browser based approach by working in the sidebar of the window and hence scores highest in *reciprocity* and *bipolarity* but little in the realm of the other qualities due to its limited set of features. Coalesce (Ryder and Anderson, 2009) goes for a multi-view implementation which gives it one point in *context-sensitivity* but leads to a loss of *reciprocity* and *bipolarity*. It also saves the outcomes of sensemaking continuously. Apolo Horng et al., 2011 is highly interactive and visual and implements a similarity algorithm as a top-down processor of sensemaking with instant recalculation on group membership change which directly supports *reciprocity* but it confined to citation network data from Google Scholar and provides no means for information foraging.

All prototypes come up to *individuality* since this quality is foundational to sensemaking which is an individual process per se. *Reciprocity* is achieved to a high degree as well, most probably because this quality is also at the heart of sensemaking. Essentially, the Learning Loop Complex of Russell et al. (1993) consists of the reciprocal interchange of bottom-up and top-down processes. *Bipolarity* and *continuity* have been found as the third-most implemented qualities since it is obvious that the information to make sense of has to come from somewhere at first and since sensemaking is an activity itself with hard to define starts and ends. The question when the goal is reached lies in the eye of the sensemaker and cannot be designed a priori. So all approaches take *continuity* into account. *Enactivity* scores only one point less and hence shows that bringing action into sensemaking has also been considered an important feature by the reviewed prototypes.

However, *context-sensitivity* seems to be a complex feature hard to tackle. This might be due to the broad range of possibilities what is considered as context and how an application can be made sensitive for it. Apparently, a multi-view approach as implemented in Coalesce and CoSense goes into the right direction. The exact situation of the sensemaker, the types of data he might be confronted with, cannot be known in advance. So, provisioning various views on the data increases the probability that one serves the sensemaker's needs.

Reification might be the most abstract quality of sensemaking which renders it hard for implementation. CoSense made a step in that direction by providing taggable summary items which allows for building of meta-structure upon the representations found during preceding sensemaking. The other approaches remain on one level of data, ie. confine themselves to a certain set of data types and/or representations. Hence, users are limited in their sensemaking by this set and can only evolve structure that is in the realm of possibilities of it. Therefore, omitting *reification* inevitably leads to implicit imposition of structure on the user's sensemaking. However, sensemaking is about the contrary. What is meant here by data and structure Dervin (1998) refers to as information and knowledge when saying that "[...] information and knowledge are rarely ends in themselves; they are rather means to ends. By freeing our interface with the user from the system's obsession with information and knowledge, we leave users free to define what is informing on their own terms" (Dervin, 1998, p. 40). As *reification* is about making an abstraction to a concrete thing it allows representations to become input to the sensemaking as if they were data. Thereby users are given the aforementioned freedom of definition.

The next section presents the design and implementation of the Bits and Pieces prototype which tackles these shortcomings by leveraging a framework approach founded on the qualities of sensemaking.

4. The Bits and Pieces Prototype

4.1. Approach

The sensemaking approach presented in this work is based on the user interface design idea B&P introduced in the strand of the EU project Learning Layers[1]. It has been developed and validated with healthcare professionals in several co-design sessions (e.g., via participatory observation, interviews, paper prototyping) and is particularly designed for sensemaking of one's own learning experiences (Tomberg et al., 2013). However, since the design is primarily made for working with people's everyday affairs, it relies upon several implicit assumptions on sensemaking. This section tries to make these assumptions explicit by applying the qualities of sensemaking distilled in chapter 2.3 and elaborating a framework for sensemaking around the design idea.

4.1.1. The B&P design idea

The user interface design idea is oriented towards the usual screen size of desktop computers. The screen area is split into two equally sized parts, the "browsing" and the "organizing" canvas, which are arranged vertically. The upper canvas is reserved for widgets which enable users to browse their digital artifacts along contextual cues (eg. time, location, topics) whereas the lower canvas gives space to widgets for organizing these items semantically. Since each canvas can only display one widget at once the user can switch widgets horizontally. Widgets shall be able to visualize entities, ie. learning "bits", in their own way and provide interaction upon them. It shall be

[1] http://learning-layers.eu/ (Visited: 2014-11-18)

possible to drag bits in the upper canvas and drop them in the lower one. Furthermore, entities need to be kept synchronized across widgets so that a change made to a bit in one widget is also reflected in the others (fig. 4.1.1). Finally, not visible in the wireframes, the state of the user's sensemaking is persisted in the background automatically without the need to save it explicitly.

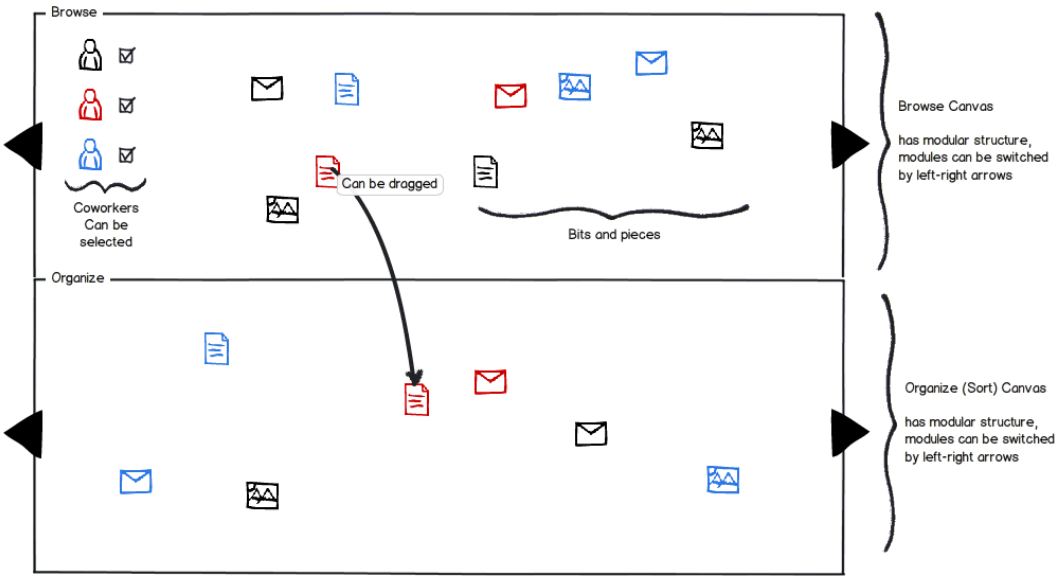
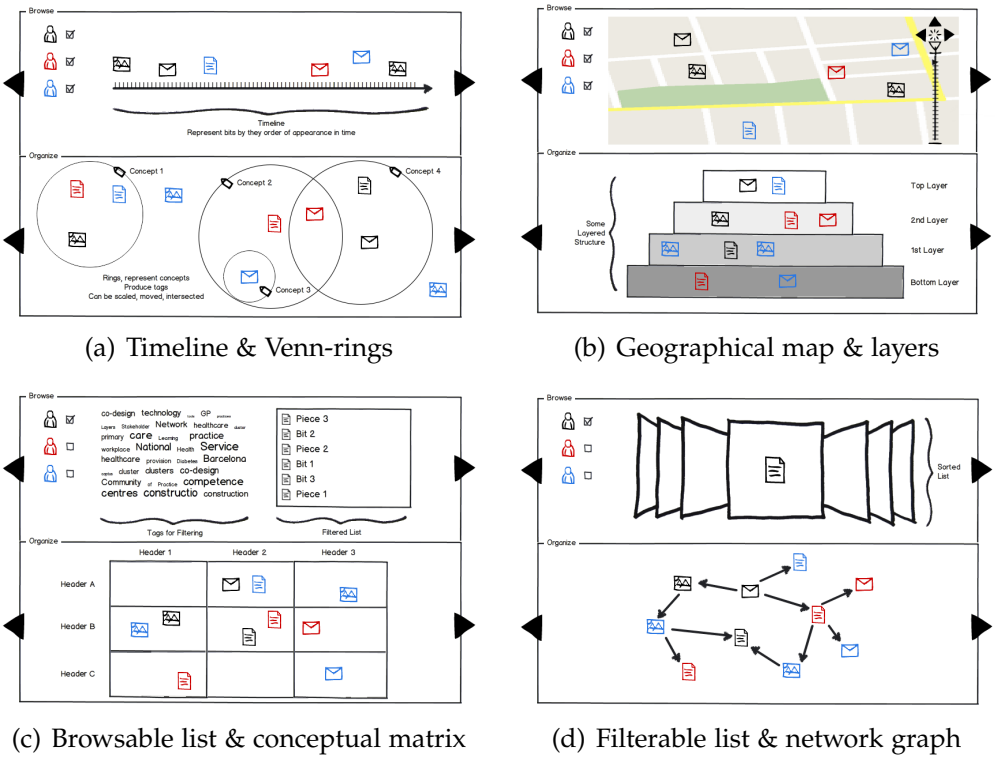


Figure 4.1.: The wireframe (by courtesy of Vladimir Tomberg) of the B&P user interface with two canvases, one for browsing data, one for organizing them. The current widget in either canvas can be switched by the arrow left and right.

The design does not confine the set of available widgets but comes with some initial suggestions. So as for the browsing canvas there could be a timeline, a geographical map and a browsable and filterable list, whereas for the organizing canvas a Venn-diagram, a layered structure, a conceptual matrix and a network graph are envisioned (fig. 4.2).

Finally, B&P should be capable of managing several "episodes" of sensemaking. An episode in the sense of B&P is sensemaking related to a specific topic, idea or question. Technically speaking, it is a means of giving a sensemaking session a name. An episode can have several versions, each



(a) Timeline & Venn-rings

(b) Geographical map & layers

(c) Browsable list & conceptual matrix

(d) Filterable list & network graph

Figure 4.2.: Example widgets of the B&P user interface (by courtesy of Vladimir Tomberg).

reflecting the state of sensemaking (ie. the arrangement of entities and the configuration of widgets) at a certain point in time. The user can switch between episodes and between versions of episodes instantly.

User Scenario

Stressing again the example from the beginning one might use the B&P user interface in the following way: First, collected data is loaded into the application via Web services and visualized by the current browsing widget, eg. the timeline, which would display the bits chronologically by the creation timestamp. However, the timeline might be unsuitable if the user already knows the label of a needed bit. Instead, by switching the upper canvas to

the browsable list, which displays bits in alphabetical order, finding a bit by name would be much easier.

Next, the user might find this bit related to another one which comes across while browsing the list further. In order to make this relation explicit the two bits are dragged and dropped to the lower canvas where the Venn diagram widget is active. The user can arrange the two bits spatially and draw a circle around them. The circle gets the label of the common category. Having found this representation, the user can continue the search for more bits which would fit into the same category.

However, if no more bits can be found, two things can be done: Either the representation needs to be changed (either by relabelling the circle or drawing new ones) or another widgets needs to be put in place, eg. the network graph. In that case there would already be three nodes visualized: the two bits and the category entity which was represented as a circle in the former Venn diagram widget. The edges would be drawn between the category entity and each bit - representing a "contains"-relationship. More bits can be dropped into the lower canvas and connected to the graph.

So having various widgets available can help in finding bits and building structure (representations) around them, ie. the B&P design idea seems to support the sensemaking process. In order to prove this theoretical assumption the next section reviews the design in the light of the qualities of sensemaking.

Proving Sensemaking Support

The qualities of sensemaking have been derived as a theoretical requirements for sensemaking applications in chapter 2.3. As they have been used in chapter 3 for evaluation of the related work B&P also needs to undergo the review of its design in the light of the qualities. This section will show that B&P fulfills these theoretical requirements.

First, the split screen design feature where the upper part is reserved for browsing and the lower part is reserved for organizing widgets apparently reflects the *bipolarity* of sensemaking. In the beginning, the process is more focussed on foraging for bits in the upper canvas, whereas towards the end

of sensemaking, it is mostly concerned with organizing them in the lower canvas. In between the process is continuously progressing its focus, ie. the user will more and more shift his attention from browsing to organizing. Since both poles are equally important for the process, splitting the screen into two equally sized parts is a feasible way of designing bipolarity.

Second, the division into two canvases also accounts for *reciprocity* of sensemaking. The organize canvas can be used to build up representations based on the data in the browse canvas whereas the browsing can be guided by representations present in the organize canvas. It is this reciprocity which drives the overall sensemaking process between the two poles of foraging and organizing. So bipolarity and reciprocity complete each other in the B&P design idea.

Third, as B&P envisions various widgets for various sensemaking situations and the ease of switching between them, it adheres to the quality of *context-sensitivity*. Whether a tool for sensemaking is appropriate mainly depends on the context, eg. the kind of data, the specific goal, the user's personal preferences, and so on. Furthermore, as the bits represent informal learning experiences they can not be defined by a certain type or structure. On the one hand they may vary in their intrinsic properties (textual notes, photos, audio records, etc.), on the other hand in their extrinsic ones (creation time, location, relation to other items etc.). By having the contents of both canvases to be switchable over an extensible set of widgets dynamically and independently the user can find the appropriate ones to visualize and interact with the data at hand.

Fourth, according to sensemaking's *enactivity*, the design idea enables users to interact with each visible element. Generally, items can be dragged and dropped from the upper to the lower canvas. Furthermore, the design suggests a couple of specific widgets for each canvas each being interactive respectively. On the one hand, for instance, the timeline widget for the upper canvas can be panned and zoomed, on the other hand the lower canvas may contain an organizing widget for drawing rings and arranging items like in a Venn diagram.

Fifth, due to sensemaking's quality of *continuity* users may cease sensemaking as soon as they see their task-specific goal achieved (Russell et al., 1993)

or may come back later. B&P achieves that by persisting the sensemaking state continuously in the background.

Sixth, *reification* is also reflected in the B&P design idea as illustrated in the user scenario. When switching widgets, representations built in the one widget might serve as entities (ie. data) in another. For instance, categories may not only function as means of structuring bits but may be related to each other on a higher level of abstraction. B&P achieves that by merely enabling reification of representations instead of defining a higher ordering structure explicitly.

Finally, the *individuality* of sensemaking is covered a priori as informal learning experiences are personal per se. Furthermore, B&P allows for building of own representations through widgets and even the individual extension of the set of widgets.

The following list sums up the mapping of B&P design features to qualities of sensemaking:

- two canvases (bipolarity, reciprocity)
- switching between widgets and extensibility of the set of widgets (context-sensitivity, individuality)
- visualization and interaction of bits within and between widgets (enactivity)
- continuous preservation of the sensemaking state (continuity)
- representations serving as entities in certain widgets (reification)

4.1.2. Functional Requirements

As B&P is targeted towards a generic framework there are only a few functional requirements to mention. Most sensemaking functionality comes from individual widgets which is out of scope of the framework. However, as for design features which are related to the visualization, interaction and continuous preservation it is necessary to list at least the following:

- Drag and drop: Every bit needs to be drag- and droppable in order to bring bits from the upper to the lower canvas. Hence, browsing

widgets need a common interface to pass out entities as do organizing widgets to receive entities.
- Bit visualization: Every bit needs to be visualized in a globally, across-widget consistent form, ie. bits need to look the same in different widgets for the sake of recognition.
- Continuous preservation: The framework needs to save the sensemaking state continuously so that the user can cease sensemaking at any point in time without worrying about its preservation. This includes the state of the widgets as well as which widgets are currently visible.
- Data structure for episodes, versions, users and entities: These are data types which are not specific to an application and hence need to be implemented on the level of the framework.

4.1.3. Non-Functional Requirements

The non-functional requirements are of more interest for the generic framework implementation because frameworks usually provide the means (the frame) to develop functionality rather than functionality as such. In the case of B&P it is a framework for sensemaking widgets which has to meet the following non-functional requirements. These also match widely with *The Reactive Manifesto* (2014) which states four properties reactive systems comply to: responsive, resilient, elastic and message-driven.

- On account of the extensibility widgets need to be *independent* modules each tackling a specific aspect of data or functionality. Independence maximizes the number of possible combinations of widgets and thereby the adaptability to the specific sensemaking objection. Furthermore independence of widgets makes switching between them flawless. Independence of modules also entails resilience because individual widget may fail while leaving the rest of the application run unaffected.
- Interactivity requires *responsiveness*, ie. the application has to respond to user input immediately without passing on time consuming loading and processing times to the user's experience. On the one hand this implies that every visible bit needs to respond to any user action which is executed upon it. On the other hand data loading and processing

logic needs to be decoupled from the visual frontend. This requirement implies client-side elasticity since "the system stays responsive under varying workload" (*The Reactive Manifesto* 2014).
- In order to support continuous preservation the application's state needs to be *serializable* at any point. This further implies that any operation triggered upon the data has to be atomic in order to make the application resilient on the level of data.
- Finally, if widgets have to be independent modules there needs to be a common means of message passing in order to enable communication with the framework and inter-widget coordination. However, the type and structure of data can not be defined a priori and thus a common data format either. Furthermore, any-time serializability needs a dynamic and extensible data structure. Both issues can be tackled by the use of a *graph data structure* as it does not impose any pre-defined structure apart from nodes and links. Furthermore, graphs can represent reification which is also a quality of sensemaking.

4.2. Framework

The implementation of the prototype is split into two parts. On account of the generic requirements described above a framework for use case agnostic features was developed. The framework is intended to ease the implementation of application specific widgets which can leverage the modularity, responsiveness and serializability provided. The main part of this section is concerned with the framework implementation. As an application two widgets of the Bits and Pieces design idea have been developed on top of this framework. The source code of the implementation can be found on Github[2].

[2] https://github.com/learning-layers/BitsAndPieces/archive/v2.0.0.zip (Visited: 2014-11-18)

4.2.1. Web Browser Runtime System

The framework relies on the runtime environment of Web browsers. The decision to use the Web browser as the application platform has several reasons. First, as the Bits and Pieces framework aims to facilitate sensemaking on the Web, the Web browser itself renders the runtime enviroment for this purpose. Second, HTML5[3] together with CSS3[4] provides a feature rich framework for structuring, integrating and presenting multimedia content. As a W3C Web standard it is implemented across various Web browsers. Third, Web applications running natively in the Web browser relieve users from awkward installation requirements. Furthermore as Web browsers are pre-installed on nowadays operating systems users can start working with the Web application right away.

JavaScript is the programming language commonly used for Web application programming. On top of the browser's runtime environment, it deals with the aforementioned requirements out of the box. As for the common interface for inter-module communication, the events API of the Document Object Model (DOM)[5] can be used to trigger and listen for application wide events which allow to transmit data objects between modules. The same mechanism enables interactivity as user initiated events (eg. a mouse click) can be captured and processed further. Asynchronous outbound communication is provided by the "Asynchronous JavaScript and XML (Ajax)"[6] technique. The DOM tree, ie. the hierarchical HTML5 structure of the user interface elements, offers fine-grained access to the presentation layer as visual elements can be addressed and updated directly. Most importantly JavaScript is a dynamic scripting language which needs no compilation of its source code. Hence objects can be extended by additional properties at runtime which is especially handy in the given case of unstructured data.

However, these are low-level features of the language which are delicate to use in a productive environment. For instance, the manipulation of the DOM tree, the handling of events and the invocation of Ajax calls requires

[3] http://www.w3.org/TR/html5/ (Visited: 2014-11-18)
[4] http://www.w3.org/Style/CSS/ (Visited: 2014-11-18)
[5] http://www.w3.org/DOM/ (Visited: 2014-11-18)
[6] http://www.w3.org/TR/XMLHttpRequest/ (Visited: 2014-11-18)

a considerable amount of code for relatively simple tasks. Due to slight differences in the various browser implementations specific knowledge and additional switches in the algorithms are necessary. Moreover, for JavaScript objects there is no communication channel like the DOM events and understandably the language does not ship with a graph database. For these reasons, a couple of open-source JavaScript frameworks have been used to ease the prototype development.

Backbone

Backbone[7] consists of three classes which are relevant for the prototype. Web developers can extend `Backbone.Model` which bundles logic for creation, access, alteration and deletion of objects. `Backbone.Collection` is needed for logic at the level of collections of models, for instance retrieval, addition and removal of objects. Collection contain Models in an 1:n relationship. Models can be bound to a `Backbone.View`, which cares for the presentation in HTML5. A view also keeps a reference to the DOM element where it puts model data to. All classes implement the Observer Pattern[8], which allows for event-driven communication between Backbone modules. For instance, a view can listen to changes to a model and trigger the appropriate functions to redraw the associated DOM element. A collection also fires events in the case of any operations invoked on it, eg. when adding a new object. Additionally `Backbone.Model` provides a generic API for synchronizing data with external RESTful server applications. Backbone does not impose any specific architecture but allows for implementation of a clean software structure.

JQuery

JQuery[9] provides a high-level API on top of JavaScript to simplify the traversal of the DOM tree, the retrieval of specific DOM elements and thus their manipulation. It also reduces the amount of code for the binding

[7] http://backbonejs.org/ (Visited: 2014-11-18)
[8] http://en.wikipedia.org/wiki/Observer_pattern (Visited: 2014-11-18)
[9] http://api.jquery.com/ (Visited: 2014-11-18)

and unbinding of event listeners to the DOM which is needed for inter-widget communication. Sending AJAX calls and receiving responses is also covered by the library by providing an exhaustive callback registration interface. Furthermore JQuery's modular structure allows for extending its functionality via plugins, eg. to get common user interface concepts like dialog boxes and drag and drop interactivity working with little effort.

VIE

VIE[10] is the so-called semantic interaction framework which brings together Backbone and the Semantic Web (Grünwald and Bergius, 2012) and enables developers to build interactive Web applications on top of semantic data. As the re-implementation of a graph database in JavaScript would neither be feasible nor high-performing VIE offers a subject-centric view of the semantic data based on JSON-LD (see 4.2.2). On top of Backbone it mimics a graph database by implementing `VIE.Entity` , which is a type of `Backbone.Model`, and `VIE.Collection` which is a type of `Backbone.Collection`for storing all entities in a central place. Every property of an entity is a URI and any entity can be retrieved from the collection by its identifying URI. Additionally, VIE provides utility functions for handling namespaces and the type hierarchy of its entities. For instance, an entity can be retrieved either by a URI or a (Safe) Compact URI (CURIE[11]). Moreover, given a type hierarchy VIE can localize an entity's type URI within that hierarchy. The type system can be parsed and imported from JSON files which use the same format[12] as Schema.org[13]. The library also supports interaction with RDFa-annotated HTML content. Synchronization with various external data Web services is enabled through service adapters. Adapters for Open Calais[14] or Apache Stanbol[15] (Westenthaler and Grisel, 2012) are already shipped with the official distribution of VIE.

[10] http://viejs.org/ (Visited: 2014-11-18)
[11] http://www.w3.org/TR/2009/CR-curie-20090116/ (Visited: 2014-11-18)
[12] http://schema.rdfs.org/all.json (Visited: 2014-11-18)
[13] http://schema.org/ (Visited: 2014-11-18)
[14] http://opencalais.com/ (Visited: 2014-11-18)
[15] https://stanbol.apache.org/ (Visited: 2014-11-18)

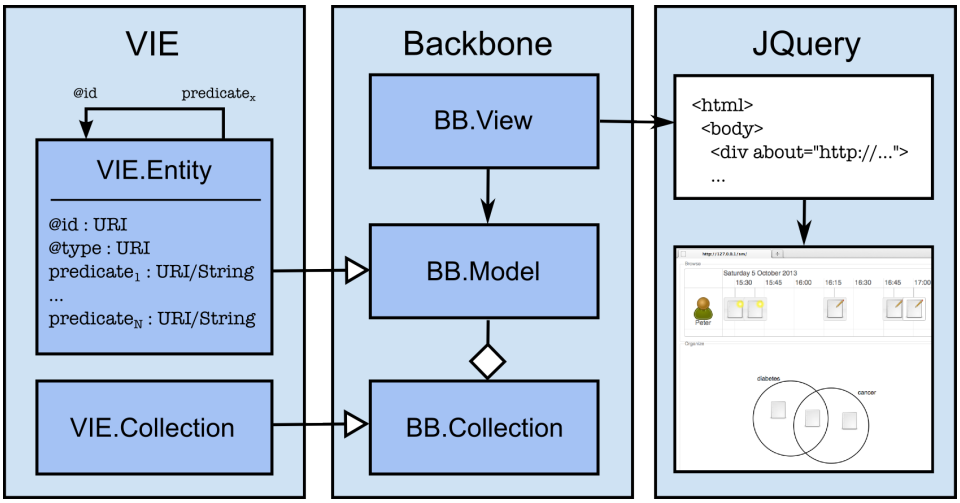


Figure 4.3.: JavaScript frameworks in use and how they interplay: VIE represents a graph data structure via inter-referenced objects of VIE.Entity, which is an extension of Backbone.Model. All objects are kept in VIE.Collection and can be connected to a Backbone.View which leverages JQuery in order to render the browser content.

Additional Utility Libraries

Apart from the aforementioned JavaScript libraries which are inherent parts of the prototype, some additional libraries are used for utility purposes. RequireJS[16] simplifies dependency management of the software modules and libraries and allows for Lazy Loading[17], a design pattern for loading JavaScript files just in time when they are needed by the application. Js-Logger[18] is used for logging and debugging needs. Finally, Twitter Bootstrap[19] complements JQuery in UI design issues.

[16] http://requirejs.org (Visited: 2014-11-20)
[17] https://en.wikipedia.org/wiki/Lazy_loading (Visited: 2014-11-20)
[18] https://github.com/jonnyreeves/js-logger (Visited: 2014-11-20)
[19] http://getbootstrap.com/ (Visited: 2014-11-20)

4.2.2. Means of Semantic Interaction

The prototype leverages two essential standards from the sphere of Linked Data for the representation of semantic data based on Web technologies: JSON-LD and RDFa. Recently, both specifications have been released as official recommendations by the W3 Consortium (W3C).

JSON-LD

The VIE framework allows for flawless integration with external semantic Web services through JSON-LD[20], a W3C standard[21] for the serialization of Linked Data (Lanthaler, 2013). Building upon the JavaScript Object Notation (JSON) it introduces additional concepts in order to map JSON objects keys to URIs. For instance, the `@context` contains namespace mappings and general type definitions. Additional keywords like `@id`, `@type` or `@list` implement node identifiers, data type relations and ordered lists.

RDFa[22], also a W3C standard[23], allows for annotation of XML with RDF. It enables Web developers to markup contents of the document with semantics primarily to make the contents more machine-readable. For instance, contact information on a Web page could be directly imported to the contact list of the visitor. RDFa defines a set of attributes which denote the relation between the documents elements and semantic entities. For instance, the `property` attribute describes the URI (absolute or relative to the namespace) of the predicate between a subject and the contained data.

Listings 4.4 and 4.5 give examples of both represenations and the triples expressed in Turle[24] notation.

[20] http://json-ld.org/ (Visited: 2014-11-20)
[21] http://www.w3.org/TR/json-ld/ (Visited: 2014-11-20)
[22] http://rdfa.info (Visited: 2014-11-20)
[23] http://www.w3.org/TR/rdfa-core/ (Visited: 2014-11-20)
[24] http://www.w3.org/TR/2012/WD-turtle-20120710/ (Visited: 2014-11-20)

```
{
  "@context": {
    "name": "http://xmlns.com/foaf/name",
    "homepage": {
      "@id": "http://xmlns.com/foaf/homepage",
      "@type": "@id"
    }
  },
  "homepage": "http://manu.sporny.org/",
  "name": "Manu Sporny"
}

// N-Triples:
_:b0 <http://xmlns.com/foaf/homepage> <http://manu.sporny.org/> .
_:b0 <http://xmlns.com/foaf/name> "Manu Sporny" .
```

Listing 4.4.: Example of JSON-LD and respective triples: The `@context` object describes the mapping of `homepage` and `name` to URIs.

```
<p vocab="http://xmlns.com/foaf/">
   My name is
   <span property="name">Manu Sporny</span>
   and my homepage is
   <a property="homepage" href="http://manu.sporny.org/">
      my homepage
   </a>.
</p>

// N-Triples:
_:b0 <http://xmlns.com/foaf/homepage> <http://manu.sporny.org/> .
_:b0 <http://xmlns.com/foaf/name> "Manu Sporny" .
```

Listing 4.5.: Example of RDFa and respective triples: `@property`/value pairs can be expressed in RDFa within HTML5, here framed by the namespace declaration `vocab`.

4.2.3. Alternative Runtimes

There exist alternative environments for building semantically enabled applications, most of them running in the Java Runtime Environment (JRE). The following provides an overview of pros and cons of such approaches:

Play

The Play Framework[25] is a Java and Scala framework for building reactive Web applications. Together with the *Imperium* plugin[26] it can be extended to process RDF data. The framework adheres to the MVC architectural pattern allowing for templating HTML markup in views, processing user input in controllers and representing the application logic in models. It merely runs on the server-side while enabling the addition of client-side JavaScript code as assets. Play is scalable in terms of workload and component development. It comes with a RESTful interface and WebSockets by default and allows for big data support.

Play is a good candidate for building Web applications which do not need much processing power on the client side. Its velocity stems from an asynchronous communication model and from the fact that it runs as a compiled Java/Scala application on servers. Client merely serve as thin-clients in the classical sense. However, for the implementation of interactive and rich internet applications one needs to fallback to JavaScript snippets shipped as assets. Since most of the B&P design idea is client-side oriented one would need to re-implement a whole architecture for this relying merely on such assets while the powerful server-side features of the Play framework remain unused.

[25] https://www.playframework.com/ (Visited: 2014-11-23)
[26] https://github.com/mhgrove/Imperium (Visited: 2014-11-23)

ROLE

ROLE (Responsive Open Learning Environments)[27] implements the concept of Mash-Up Personal Learning Environments on top of a semantic model, ie. the ROLE ontology. The semantic model relies on the concept of spaces. Users can create spaces, where they can manage learning content and collaborate with other users. A space can host a bundle of Web widgets, has a list of participants in the space and enables them to communicate and collaborate. Moreover, for supporting complex applications, the widgets can exchange information across browsers by using the inter-widget communication mechanisms developed within ROLE. Messages are composed of URI/value pairs and hence allow for handling semantic data. The widget framework is oriented to work on the client-side with the server functioning as a central point of the widget store, the identity provider and the cross-browser XMPP communication.

In this respect, ROLE is a full-featured widget framework written in Java and JavaScript facilitating individual development of widgets, inter-widget communication and user collaboration. Integration and flawless invocation of widgets from the widget store is one of its main targets. However, its design is fixed to the notion of Personal Learning Environments, ie. widgets serve as small information processing and user interaction units but cannot be used for more ambitious application needs like the highly interactive and multimedial canvases envisioned by the B&P design idea. Additionally, there is no common data base for the widgets as every widget is an island of its own data. The only means of matching data is through the exhaustive communication mechanism which would be cumbersome to apply, for instance, to the Learning Loop Complex in sensemaking.

Callimachus

Callimachus, a wiki-based data management framework, is specifically made for Semantic Web applications. It comes with inherent RESTful APIs, user account management, various authentication mechanisms and most

[27] http://www.role-project.eu/ (Visited: 2014-11-23)

notably with a full-featured user interface which makes building a semantically enabled hyperlinked Web applications a matter of a few clicks. Users with minimal knowledge on Semantic Technologies can stick together an ontology with the integrated editor and create "Create"-, "Edit"- and "View"-templates for RDF classes. Templates are marked up with RDFa with support for variables (ie. parts of the template which need to adapt to changes in the RDF data model), expressions and loops. "View"-templates can be invoked and displayed on the result set of a SPARQL query. Additionally Callimachus supports a set of charts for visualizing the semantic data of an application as well as XProc[28] as a W3C standardized technology for realizing Extract-Transform-Load-pipelines[29].

Callimachus implements a full stack of Semantic Web technologies, including a graph database, RDF/XML RESTful APIs, a SPARQL endpoint, an ontology editor and RDFa templates. These enable users to easily build Semantic Web applications in a "wiki", ie. fast, way. However, Callimachus itself is more an application than a framework. In order to add new kinds of visualization or to implement highly interactive applications, one would need to go beyond the scope of Callimachus. For instance, charts are merely oriented at presenting semantic data in a visually appealing form. There is no API for programming further means of user interaction on top of visualizations. The provided JavaScript API only allows for invoking functionality which is reachable via Callimachus' user menu anyway. Even if it were manageable to implement individual visualizations one would still hassle with setting up a way of communication between them. For instance, interactive features as required by B&P functionally, like dragging and dropping of bits between widgets, could not be implemented easily with Callimachus.

4.2.4. User Interface Design and Workflow Overview

The user interaction workflow is driven by two phases. In the first phase the user collects data, ie. informal learning experiences, which might be textual notes, pictures, Web links and other types of multimedial content,

[28] http://xproc.org (Visited: 2014-11-23)

[29] http://en.m.wikipedia.org/wiki/Extract,_transform,_load (Visited: 2014-11-23)

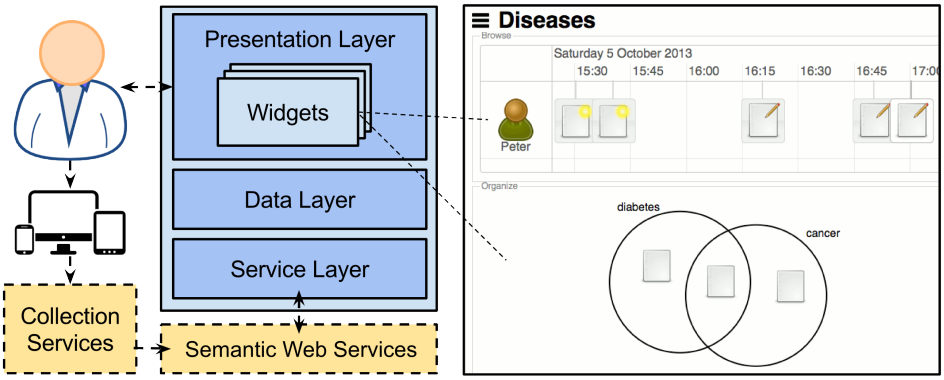


Figure 4.6.: B&P user interface design and workflow overview: The B&P user interface carries on the initial design idea of two canvases which contain widgets (eg. a timeline and a Venn diagram widgets). Widget implementations are part of the presentation layer and react to user interaction events. The Service Layer stores and retrieves application data to and from remote Web services. This external data sources might by supplied by the user's mobile collection services.

with the help of mobile applications in the field. These collection services can be connected to other remote Web services which provide semantic access to the data for the second phase. Here the user operates the B&P user interface to make sense of the collection of learning bits. The data undergo a transformation pipeline through the layered architecture of the B&P framework: the Service, the Data and finally the Presentation layer.

Widgets of the Presentation Layer implement individual visualization and interaction functionality. Bits are displayed uniformly across widgets in order to ease recognition and to ensure consistent appearance when dragging and dropping bits from the upper to the lower widget. A sidebar to the right of the screen pops up on clicking on a bit. The sidebar contains detail information and allows for changing various properties of the bit (eg. the title). Double-clicking a bit opens another Web browser tab in order to dereference its URI if possible.

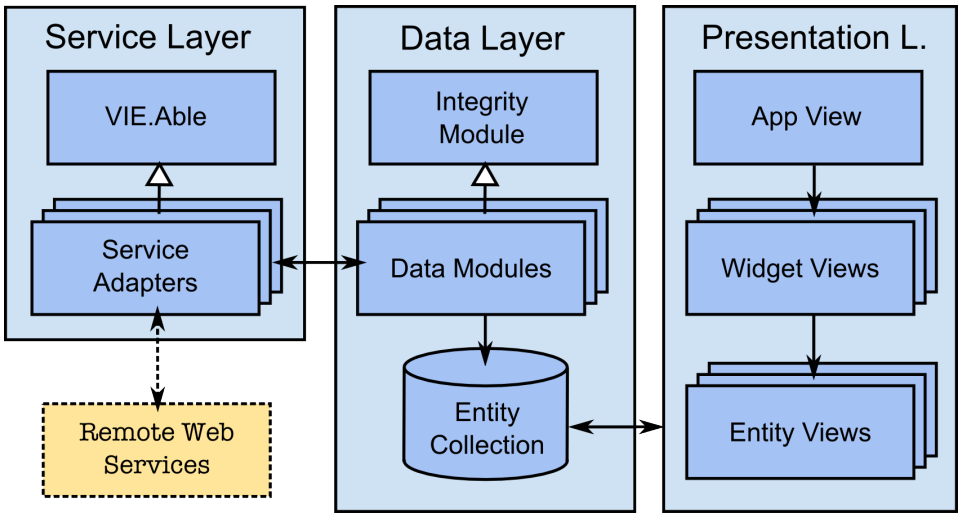


Figure 4.7.: The B&P framework architecture: The service layer wraps remote data services (eg. the Social Semantic Server) to the data layer, ie. its data modules, which curate the entity collection. Views are bound to entities and propagate changes in the data to the user frontend. Users may generate content via external systems, eg. EVERNOTE, which can be fed back into the remote data service.

4.2.5. Framework Architecture

The B&P framework separates the concerns for remote services, data and presentation in a three-layered architecture. The service layer wraps remote Web services and prepares external data sets. The data layer curates the collection of entities of the application. It keeps references between entities consistent to adaptable data integrity constraints and bundles data loading and creation logic. The presentation layer consists of one or more views which build up the user interface from the entities given, leveraging Backbone's declarative event handling mechanism.

Service Layer

In general, the service layer implements VIE's service adapter interface `VIE.Able` which provides extensions for loading, saving and analyzing

```
this.VIE.load({'service' : 'entityDescsGet'})
   .from('sss')
   .execute()
   .success(function(entities) {
      LOG.debug("successfully loaded entities", entities);
   })
   .fail(function(errorMsg) {
      LOG.debug("an error occurred", errorMsg);
   });
```

Listing 4.8.: Example service call using JQuery Deferred Object: `load` initiates the Deferred Object, ie. the `Loadable` which extends `VIE.Able`, parameterized for the specific service call `entityDescsGet`. `from` configures the object to use the registered service `'sss'`. `execute` invokes this service, which transmits the actual server request. `success` and `fail` execute their callback in the case of the respective response type.

content. All extensions implement JQuery's Deferred Object[30], a design pattern resembling the Builder Pattern[31] for configuring the service call with parameters and callback functions. Callbacks are executed as soon as a service call, which is made asynchronously via AJAX, gets a response from the remote server. Services need to be registered with the runtime instance of `VIE`. Listing 4.8 gives an example.

Data Layer

The VIE framework provides the data layer with utility functionality for dealing with semantic data. As all entities are of the type `VIE.Entity` and kept in a single `VIE.Collection` (see fig. 4.7) they are subject to VIE's semantic interaction framework. The data layer is organized in modules each handling an aspect of the application data. Modules may trigger data operation on each other to the extend of their inter-dependency. For instance, application specific data modules for a hierarchical data model can listen to certain entities being added to the entity collection (see lst. 4.10) and

[30] http://api.jquery.com/category/deferred-object/ (Visited: 2014-11-25)
[31] https://en.wikipedia.org/wiki/Builder_pattern (Visited: 2014-11-25)

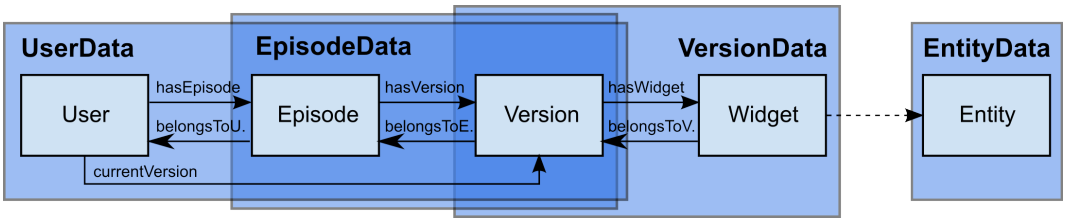


Figure 4.9.: The data model of the framework as it implemented by the respective data modules `UserData`, `EpisodeData`, `VersionData` and `EntityData`. For the data type `Widget` there is no data module on the level of the framework because widgets are application specific. Nevertheless, widget implementations have to adhere to this data model.

trigger the loading of its child entities (see 4.11). Data modules need to be initialized in the data layer's core component `AppData` which can also be used to initialize additional application specific data, eg. the initial widgets to get the user start working with. Apart from this component and the data modules the data layer consists of the Data Integrity Module (`Data`) which subsumes common integrity functionality and `CopyMachine` for cloning of entities. The framework also comes with a couple of predefined data modules which are a requirement of B&P: `UserData` manages user data and related episodes. `EpisodeData` and `VersionData` deal with the respective data types and control for cloning of sensemaking states, ie. the arrangement of entities and widget configuration. `EntityData` is a special data module for common data logic of entities such as for tagging or authorship. However, appropriate application-specific service calls have to be integrated to have these modules synchronizing with external Web services.

Data modules may inherit from the Data Integrity Module in order to impose integrity constraints on their data structure. For instance, having an entity with a relationship to another type of entity may be reflected by the integrity constraint `<relation1, type, relation2>`. Whenever an entity is created or updated the data integrity module should check that the entity has a property named `relation1` with the reference to another entity of type `type`and vice-versa with `relation2`. If the constraint is violated it will be fixed. If the entity is deleted, respective relationships of other entities to the deleted one need to be removed also.

For these two purposes, setting up an integrity constraint and checking

```
var UserModule = Object.create(Data);
UserModule.init = function(VIE) {
    this.VIE = VIE;
    this.VIE.entities.on('add', this.filter, this);
    this.setIntegrityCheck(
        Voc.hasEpisode, Voc.Episode, Voc.belongsToUser);
    this.setIntegrityCheck(
        Voc.currentVersion, Voc.Version);
};
```

Listing 4.10.: Example definition of a data module and initiation of a user entity: First, JavaScript's `Object.create()` initiates `UserModule` with `Data`, the Data Integrity Component, as its prototype. `init` gets VIE's runtime instance and sets up an event listener, which filters entities for user entities, and integrity constraints to episode and version entities.

it upon an entity, data modules inherit two methods from Data Integrity Module:

- `setIntegrityCheck( relation1, type, relation2 )` (see lst. 4.10)
- `checkIntegrity( entity )` (see lst. 4.11)

The framework extends `VIE.Entity` so that it overwrites `Backbone.Model`'s `sync` method, which is used by Backbone to create/read/update/delete (CRUD) an entity to external data services. It can be used to implement generic synchronization functionality which applies to all types of entities. However, data modules might overwrite `sync` once more with data type specific synchronization functionality (see lst. 4.12).

The data layer synchronizes application data with the presentation layer which needs to serve immediate response to the user despite time-consuming asynchronous operations of the service layer. For instance, in the case of loading entities the data layer may provide the presentation layer with preliminary mockup data until the requested entities are passed in through the service layer. In the case of the creation of an entity the data layer may create a representation of it which exists merely on the client-side by giving

```
UserModule.filter = function(entity) {
    if( entity.isof(Voc.User) ) {
        this.checkIntegrity(entity);
        this.fetchEpisodes(entity);
        this.fetchCurrentVersion(entity);
        entity.sync = this.sync;
    }};
```

Listing 4.11.: Data type specific initiation: `filter` is invoked on `add`-events on the entity collection. If the new entity is of type `Voc.User` the integrity constraints are checked and additional data (the user's episodes and the current version the user is working on) are loaded. Finally the entity's `sync` method is overwritten by the module's one (see lst. 4.12).

```
UserModule.sync = function(method, entity, options) {
    if( method !== 'update' ) {
        this.VIE.Entity.prototype.sync(method, entity, options);
        return;
    }
    if( UserModule.hasCurrentVersionChanged(entity) ) {
        UserModule.saveCurrentVersion(entity, options);
    }
    // handle rest of changed attributes by generic sync
    this.VIE.Entity.prototype.sync(method, entity, options);
},
```

Listing 4.12.: Data type specific `sync` method: If the user entity's `currentVersion` attribute changes, this `sync` method invokes specific functionality to save this attribute. For changes to other attributes again the generic `sync` method is used.

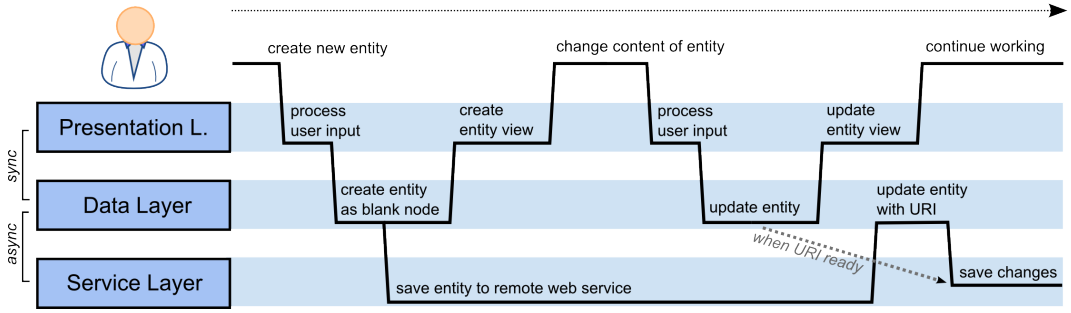


Figure 4.13.: Data flows between layers for creating and changing an entity. Both flows are initiated by the user, processed by the presentation layer and handed over to the data layer. In the first flow the new entity is created as a blank node while the service layer saves the entity to an external Web service asynchronously. In the second flow changes made to the entity cannot be passed to the service layer immediately because the URI is not yet available due to delayed response from the prior Web service call. As soon as the URI is ready, the changes can be saved.

it a blank node[32] URI of the form `_:bN` with *N* being a serial number. The user may continue working with the temporary entity representation. As soon as the service layer hands back the response of the server-side creation of the entity containing its real URI the data layer can transmit any further changes made meanwhile on the client-side representation. See fig. 4.13 for a timing diagram of this data flow.

Presentation Layer

A view in the presentation layer has to keep track of changes of entities which it is projecting to the visual HTML5 frontend of the user interface. Alternatively a view may also listen to new entities added to the entity collection and create new instances of subordinate views. Individual widgets can implement meaningful visualization techniques and interaction affordances on top of the semantics of a bit. The interface of the DOM tree provides fine-granular access to respective DOM elements for either

[32] http://www.w3.org/TR/2014/REC-rdf11-mt-20140225/#blank-nodes (Visited: 2014-11-25)

manipulating its contents or listening for user events triggered upon it. According to Backbone a view also handles user input and hence takes the role of a controller to some extend. For instance, it is feasible to have a main view which controls the instantiation of other views.

The presentation layer consists of the `AppView` which manages the space for the two canvases as well as application-wide events. Essentially it triggers re-rendering of widgets if the user switches versions since each version might have different widgets activated. Each version will get its representation in the DOM tree but only the current version element with its child widget elements is shown by toggling the CSS `visibility` property. Additionally, there are two other types of views which application specific view implementations need to reuse: `WidgetView` and `EntityView`. `WidgetView` functions as a facade[33] between the `AppView` and application specific widgets. It distinguishes between "browsing" and "organizing" widgets and applies respective functionality. For instance, organizing widgets need to handle dropped entities appropriately by the help of the JQuery UI plugin Droppable[34]. For each widget available `WidgetView` has to provide a method to create it.

On the other hand, `EntityView` provides common UI functionality of entities and can be overwritten by type specific views. It provides an appropriate image of an entity given a list of type-to-icon mappings and handles mouse events triggered upon its containing DOM element. Browser-generated click events are wrapped into the framework-specific "bnp:clickEntity" event which contains a direct reference to entity object. Furthermore, using the JQuery UI plugin Draggable[35] `EntityView` makes the DOM element draggable. In order to identify the entity object which belongs to the dragged DOM element, the view also adds RDFa to its HTML markup (see fig. 4.14 and also 4.31).

Finally, `EpisodeManagerView` together with `EpisodeView` manages episodes and versions. In the header above the upper canvas the label of the current episode is displayed. A dropdown box enables the user to switch between episodes and versions. According to the MVC pattern, views are updated

[33] https://en.wikipedia.org/wiki/Facade_pattern (Visited: 2014-11-25)
[34] http://jqueryui.com/droppable/ (Visited: 2014-11-25)
[35] http://jqueryui.com/draggable/ (Visited: 2014-11-25)

```
this.$el.draggable({
    zIndex: 10000,
    helper: "clone",
    appendTo: "body",
});
```

Listing 4.14.: JQuery draggable: `this.$el` refers to the (JQuery wrapped) DOM element of the `EntityView`. `helper: \clone"` instructs `draggable` to create a copy of the DOM element which is moved along the mouse moving. By `appendTo: \body"` the helper element becomes draggable over the whole page.

on changes in the model. Therefore switching first changes the property `Voc.currentVersion` of the user entity which then triggers re-rendering of the canvases by `AppView` (see fig. 4.15).

Operative Components

Apart from the layers there exist a few operative components which stick the system parts together.

- `index.html` is the first file to load which contains initial instructions for the initialization of B&P. Apart from CSS files it loads the external library RequireJS together with its configuration component `main.js`. The body of `index.html` only invokes RequireJS to start the application from `app.js`.
- `Main` configures dependencies between the components leveraging RequireJS. Based on this configuration RequireJS loads JavaScript files in the right sequence avoiding dependency leaks.
- `App` is the applications entry point. It initializes loggers, checks for user authentication, creates the runtime instance of VIE, loads the type system from a Schema.org-file and invokes `extender`. Most importantly `App` initializes the layers by creating and registering services, passing the VIE instance to `AppData.init()` and finally rendering `AppView`. Since data modules are event-driven, `App` also kicks off initial data retrieval by creating/loading the entity which represents the user (see lst. 4.16).

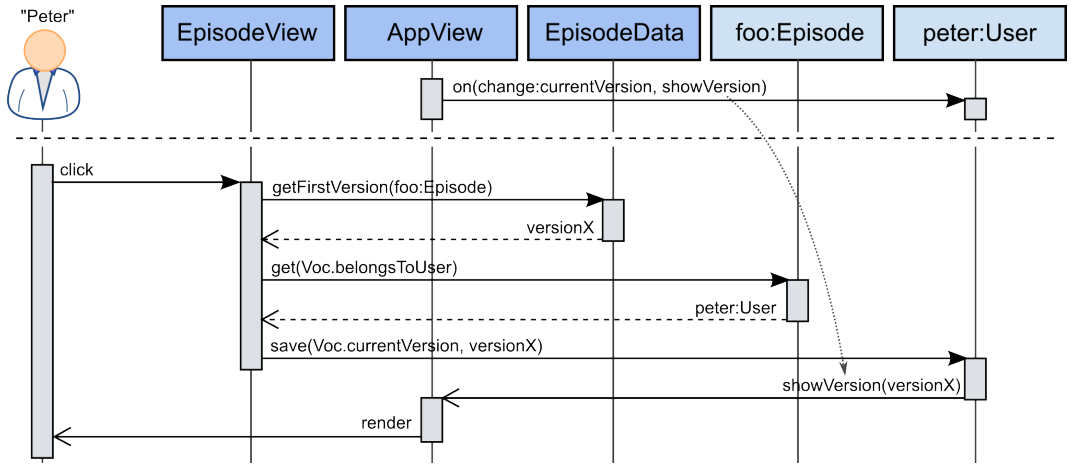


Figure 4.15.: Sequence diagram for switching episodes: The first sequence before the dashed line shows the registration of a `change:Voc.currentVersion`-listener with the callback `showVersion`. The other sequence demonstrates the interaction of components (dark color) and entities (light color) when the user switches to another episode by clicking on the its view.

- `UserParams` is initialized with the login data of the user and made available throughout the whole system in order to give other components a means of identification.
- `Extender` extends the instance of VIE which additional functionality needed throughout B&P. Essentially it overwrites `VIE.Entity`'s `sync` method to use certain generic Web services for creating/reading/updating/deleting entities and adds a special listener function `onUrisReady` which provides the retention of service calls for temporary entities with blank URIs. The function waits for these entities to get real URIs and then executes a callback which might make the service call.

Supportive Components

Finally there exist a couple of components which have mere supportive function for the system.

```
var user = new VIE.Entity;
user.set('@id', UserParams.userId);
user.set('@type', Voc.User);
VIE.entities.addOrUpdate(user);
user.fetch();
```

Listing 4.16.: Kicking off application data loading: The data of the logged-in user, provided by `UserParams`, is used to initialize the first `VIE.Entity`, which is of type `Voc.User`. By adding it to the entities collection loading mechanisms of other data modules will be triggered. Finally, the user's additional properties are fetched asynchronously.

- `LoginFormView` renders the login form when the user has not been authenticated yet and invokes `UserAuth` which contains authenticating Web service calls (outside of the service layer). User data then is written to `UserParams`. For debugging purposes these components can be bypassed completely. However, in that case `UserParams` needs to be filled with appropriate user data manually.
- `Voc` contains variable mappings for URIs. Although URIs should no change, it happens during development. Since URIs are used throughout the whole system, `Voc` provides a central place for defining them.
- `Tracker` is an extension of `Logger` and is used to log user actions to an external Web service.

4.3. Application

B&P comes with a couple of suggestions for widgets. Two of them, the timeline and the Venn diagram, have been implemented with the framework described in the previous section. The timeline is rendered in the upper canvas and displays entities chronologically according to their creation timestamp. The Venn diagram is a widget for the lower canvas and allows for drawing circles and arranging entities spatially. Fig. 4.17 shows the mockup drawings of the two widgets together with its realization as the B&P user interface. Fig. 4.18 shows an example workflow using it.

Since the application is under heavy development by the time writing this thesis the interface has progressed and now contains additional functionality like for searching, sharing and collaboration which is all wrapped up in the sidebar component of the user interface. However, due to that functionality going beyond the scope of this thesis, it is not covered here in detail, although it is worth mentioning that the Bits and Pieces framework is still in use and allows for application specific extensions with richer features than those of the first prototype version presented here. Implementation of widget switching has also been postponed because it would not yield much relevance for the essential functionality of the framework although it would be an important feature for sensemaking as such.

Two views have been added to the presentation layer, `TimelineView` and `OrganizeView`, together with sub-views wrapping visualized entities. As for the data layer, respective data modules have been implemented, ie. `TimelineData` and `OrganizeData`, also with associated data modules for contained entities. Finally, a service adapter for the Web services of the Social Semantic Server has also found its way into the application.

4.3.1. Social Semantic Service

In the service layer, the prototype application wraps remote RESTful Web services of the Social Semantic Server[36] (SSS, Kowald et al., 2013), which hosts the so-called Artifact-Actor-Network (AAN, Reinhardt, Moi, and Varlemann, 2009). In AANs, users and digital artifacts can be interlinked and used together with common features of social networks. Low Level Services support sharing and subscribing processes of artifacts, provide functionalities to annotate entities with metadata, to authenticate users and to deal with digital artifacts such as texts or multimedia documents (e.g. pictures and videos) involved in the AAN. High Level Services use given semantic structures formed by Low Level Services. They exploit explicit and implicit (social) relations to provide functions that support a personalized reflection of AANs, to discuss about digital artifacts, to handle collaborative work on digital artifacts and to allow for (re-) structuring of hierarchical and

[36] https://github.com/learning-layers/SocialSemanticServer (Visited: 2014-11-25)

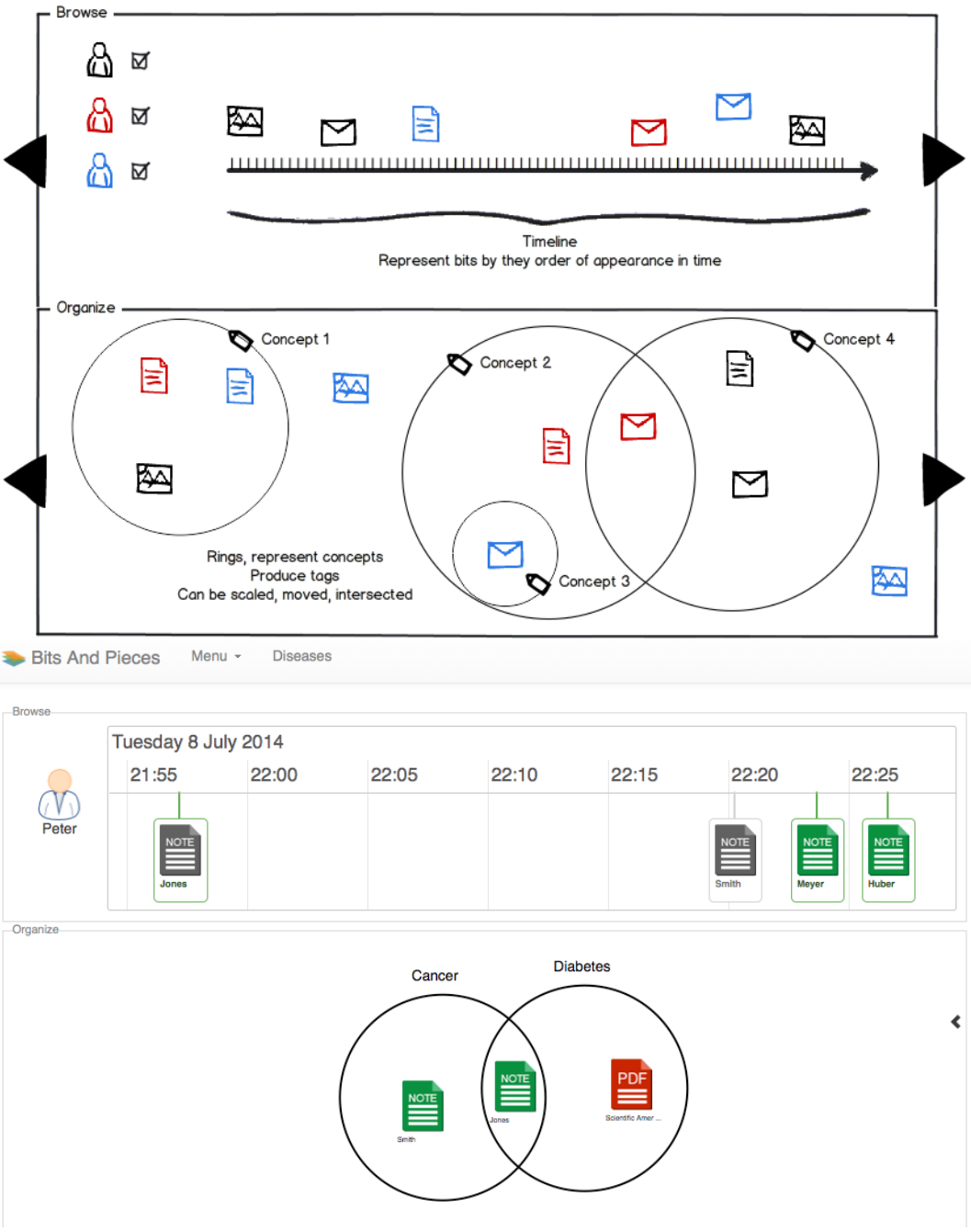
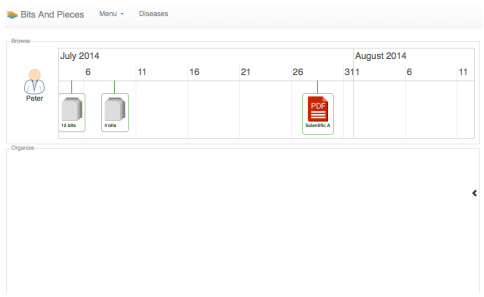
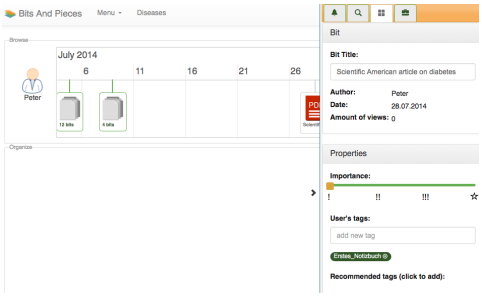


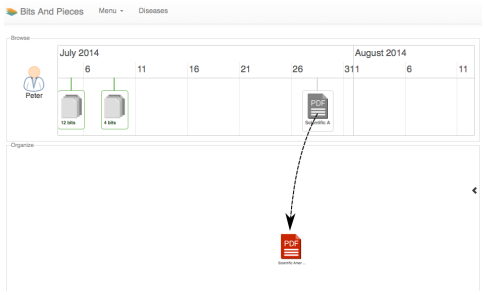
Figure 4.17.: The mockup of the timeline and the Venn diagram widget and how it is realized.



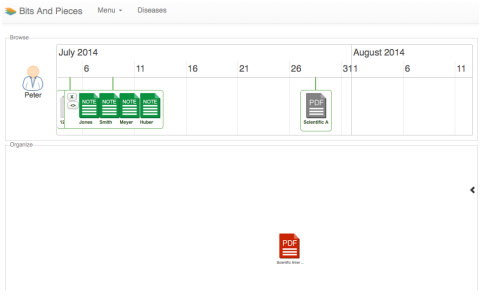
(a) Beginning of sensemaking: Timeline shows two clusters of bits and one PDF bit on a month-wide scale.



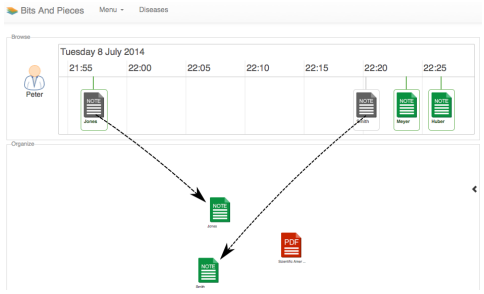
(b) Sidebar pops up on click on the PDF bit showing detail information.



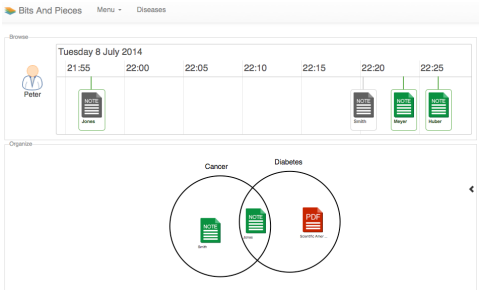
(c) Drag and drop of the PDF bit to the lower canvas.



(d) On click on cluster show bits contained. On click on the "<>" button ...



(e) ... zoom the timeline automatically to show exactly the clustered bits. Drag and drop two more bits.



(f) Draw circles, name them and place bits inside.

Figure 4.18.: Demo workflow of the B&P prototype: The upper canvas contains the timeline, the lower canvas the organize widget. The header contains the label of the episode and a dropdown menu.

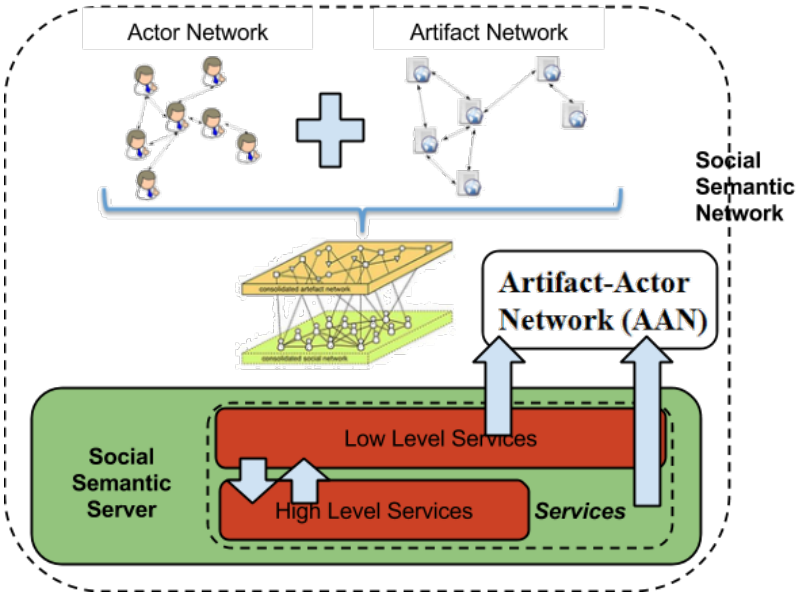


Figure 4.19.: Social Semantic Server and Artifact Actor Networks

ordered collections of digital artifacts (fig. 4.19). The SSS can also mediate data from 3rd-party collection services, eg. Evernote. As for the application SSS backs up all data storage and retrieval needs of B&P since it is also part of the Learning Layers Project and developed in parallel with B&P.

The SSS is written in Java and is accessible via REST or WebSockets. To increase interoperability, services receiving application input or delivering results to the requesting application use JSON to encode and transmit data. For the framework, REST and WebSocket JavaScript libraries are available to be included in applications directly to minimize efforts of tying the framework to custom applications.

The service adapter for B&P consists of `SocialSemanticServer` and `SocialSemanticServiceModel`. The first implements the `VIE.Able` interface and bundles logic for handling server request. The latter describes a subset of SSS services in terms of parameter and result types as well as pre- and postprocessing functions in a concise form. `SocialSemanticService` looks up service descriptions, invokes preprocessing functions on the parameters, transmits the service call to the SSS endpoint and postprocesses the result.

A common preprocessing function is `scrubParams` which makes sure that parameters are of the type (eg. array, scalar number) expected by the service. A common postprocessing function is `fixForVIE` since by the time developing B&P SSS is not serving JSON-LD yet and according transformation of the result data is necessary. The application utilizes the following SSS services (excerpt from `SSConns.js` of SSSClientSide[37]):

- `entityGet` returns generic properties like label, author or creation timestamp of a single entity by its URI.
- `entityDescGet` returns more detailed information for given entity URI such as tags, user events or a thumbnail.
- `entityDescsGet` returns more detailed information for a set of given entity URIs.
- `entityUpdate` updates or adds given properties for an entity URI.
- `uEsGet` returns user events for a given user URI, an entity URI and a time range, ie. user events which were caused by a certain user, on a certain entity within a certain time range.
- `learnEpCreate` creates an episode by its label and returns the URI.
- `learnEpsGet` returns all episodes for a given user URI.
- `learnEpVersionsGet` returns all versions with their entities and circles for a given episode URI.
- `learnEpVersionGet` returns a single version with its entities and circles given its URI.
- `learnEpVersionCurrentGet` returns the version URI which the user is currently working on.
- `learnEpVersionCurrentSet` saves the version URI which the user is currently working on.
- `learnEpVersionGetTimelineState` returns the state of the timeline of a given version URI, ie. the setting of start and end timestamps of the timeline.
- `learnEpVersionSetTimelineState` saves the start and end timestamps of the timeline.
- `learnEpVersionCreate` creates a new version for a given episode URI and returns the version's URI.

[37] https://github.com/learning-layers/SocialSemanticServerClientSide (Visited: 2014-11-25)

- `learnEpVersionAddCircle` add a new circle (coordinates, size and label) for a given episode URI.
- `learnEpVersionUpdateCircle` updates given attributes for a circle URI.
- `learnEpVersionRemoveCircle` removes a circle by its URI.
- `learnEpVersionAddEntity` add a new orga-entity (an entity of type `OrgaEntity` with coordinates and a resource entity URI) for a given episode URI.
- `learnEpVersionUpdateEntity` updates given attributes for an orga-entity URI.
- `learnEpVersionRemoveEntity` removes an orga-entity by its URI.

4.3.2. Type Hierarchy

Prior to the initialization of the application its type hierarchy is loaded into VIE using its `loadSchemaOrg` utility method. Schema.org[38] is an initiative of several major search engines to standardize semantic markup for Web pages which can be parsed by their crawlers. There exist schemata for domains of events, organizations, persons, actions and many more. The whole ontology can be loaded as RDF/Turtle, RDF/XML, RDF/NTriples, CSV or JSON[39]. As VIE comes with a parser for the latter the application-specific type hierarchy has been created in the same format. Fig. 4.20 provides an example definition.

There exist types for `OrganizingWidget`, a widget with "organizing features", `Organize`, the Venn diagram widget, a generic `entity` type which is the parent type of, for instance, `evernoteResource` or `evernoteNote` and a `userEvent` type with a large body of subtypes.

4.3.3. Timeline Data Module

`TimelineData` manages data affairs concerning the timeline widget, ie. of entities of the type `Voc.Timeline` which contain the current visible range

[38] http://schema.org/docs/schemas.html (Visited: 2014-11-26)
[39] http://schema.rdfs.org/all.json (Visited: 2014-11-26)

```
"http://eval.bp/Widget":{
   "ancestors": [ "http://eval.bp/Thing" ],
   "comment": "A widget",
   "id": "http://eval.bp/Widget",
   "label": "Widget",
   "supertypes": [ "http://eval.bp/Thing" ]
},
"http://eval.bp/BrowsingWidget":{
   "ancestors": [ "http://eval.bp/Widget" ],
   "comment": "A widget with browsing features",
   "id": "http://eval.bp/BrowsingWidget",
   "label": "BrowsingWidget",
   "supertypes": [ "http://eval.bp/Widget" ]
},
"http://eval.bp/Timeline":{
   "ancestors": [ "http://eval.bp/BrowsingWidget" ],
   "comment": "A Timeline Widget",
   "id": "http://eval.bp/Timeline",
   "label": "Timeline",
   "supertypes": [ "http://eval.bp/BrowsingWidget" ]
}
```

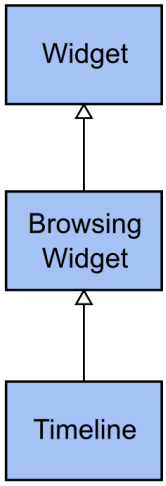



Figure 4.20.: Timeline widget type hierarchy in Schema.org JSON format. `http://eval.bp` is the namespace of the application.

of the timeline in terms of start and end timestamps as well as a reference to the user entity the timeline belongs to. Finally, a timeline entity has a reference to a version since widgets belong to a version in general. Hence, `TimelineData`, by extending the Data Integrity Module `Data`, imposes the following two constraints:

```
setIntegrityCheck(Voc.belongsToVersion,Voc.Version,Voc.hasWidget)
setIntegrityCheck(Voc.belongsToUser,Voc.User)
```

`TimelineData` does not manage the entities displayed in the timeline itself. Instead this is delegated to `UserData` since the user events belong to user entities. In general, the data module uses its version entity to invoke the two SSS services `learnEpVersionGetTimelineState` and `learnEpVersionSetTimelineState`. Finally, the data module provides an interface for copying its data structure in order to enable `CopyMachine` to clone a timeline entity. In sum `TimelineData` consists of the following methods:

- `init` sets up the aforementioned integrity constraints and binds `filter`

to add-events on the entities collection.
- `filter(entity)` filters entities of type `Voc.Timeline` and applies the integrity checks on them. Furthermore, it fetches the timeline state calling `fetchTimelineState` and sets up an event handler function for changes of the time range which calls `UserData.fetchRange`.
- `sync(method, entity)` overwrites the generic `sync` method of `VIE.Entity` to call `saveTimelineState` if `method` is "update" and `fetchTimelineState` if `method` is "read".
- `saveTimelineState(entity)` invokes the SSS service `learnEpVersionSetTimelineState` passing the start and end timestamps of the entity.
- `fetchTimelineState(entity)` invokes the SSS service `learnEpVersionGetTimelineState` and puts the result (start and end timestamps) on the entity.
- `copy(entity, overrideAttributes)` creates a copy of the entity, optionally overriding its attributes with `overrideAttributes`.

`UserData` has been extended to provide user events and associated entities. `fetchRange` calls the SSS service `uEsGet` for retrieval of user event entities between given start and end timestamps. These entities have a `Voc.hasResource` reference to a resource entity. For a bulk retrieval of their properties, `fetchRange` passes their URIs on to the SSS service `entityDescsGet`.

Fig. 4.21 provides an overview of the whole data model of the application.

4.3.4. Organize Data Module

`OrganizeData` bundles data logic for entities of type `Voc.Organize`. As `TimelineData` the data module sets up the integrity constraint that `Voc.Organize` entities must have the relationship `Voc.belongsToVersion` to an entity of type `Voc.Version` and a relationship `Voc.hasWidget` vice-versa. `OrganizeData` also implements a `copy` method for cloning of an organize entity's data structure. Finally, the data module handles the creation of orga-entities and circles and adds them to the entities collection (see lst. 4.22). An orga-entity is a wrapper around the actual resource entity which gets visualized on the canvas. In addition to the referenced resource entity, an orga-entity carries the coordinates for positioning within the canvas space.

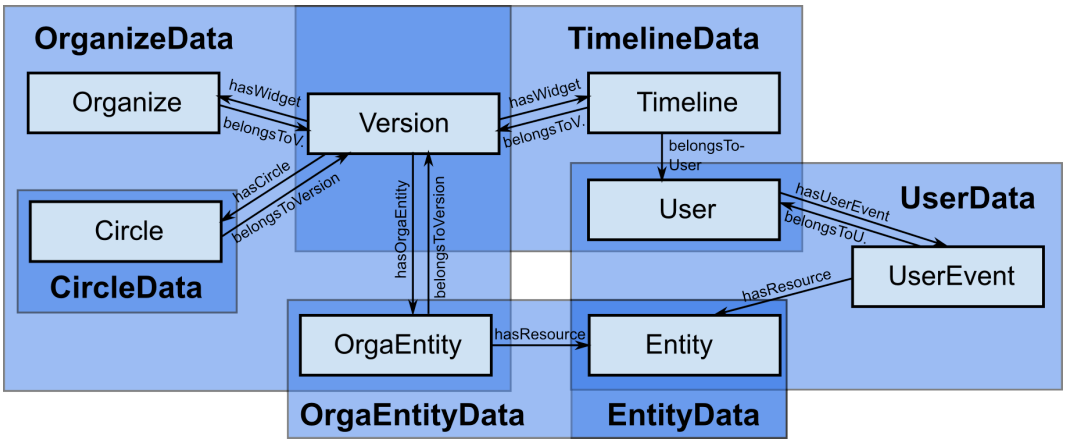


Figure 4.21.: The data model of the application: Data modules (dark blue) frame entities (light blue) which they handle.

Actual create/update/delete operations on orga-entities and circles are delegated to the data modules `OrgaEntityData` and `CircleData`. Both listen for entities of the respective type being added to the entities collection and overwrite their `sync` method with appropriate SSS service invocations (see lst. 4.23).

4.3.5. Timeline Widget

The timeline provides a time-based visualization of entities and allows for interactive configuration of its displayed time range by zooming and panning. If more than one entity overlap in the timeline they are clustered in order to clean up the browsing space. The number of entities contained is rendered as the label of the cluster. Hence, depending on the zoom level of the timeline it can provide a quick overview of the distribution of the entities by letting them be clustered accordingly. Clicking on a cluster expands the box to yield all contained entities in sequential order. Additionally the browsing widget can be zoomed automatically to show all entities of a cluster.

Sketched in the mockup but not realized in the implementation is the possibility to view entities of various users at once. This feature might

```
OrganizeData.createCircle = function(organize, circle) {
    circle.set({
        '@type': Voc.Circle
    });
    var version = organize.get(Voc.belongsToVersion);
    circle.set(Voc.belongsToVersion, version.getSubject());

    this.vie.entities.addOrUpdate(circle);
    circle.save();
    return circle;
};
```

Listing 4.22.: OrganizeData.createCircle: The method receives an organize entity representing the widget and a circle entity (with coordinates, size and label already set by the presentation layer). The type and the relationship to the version of the organize widget are set accordingly. Finally, the new entity is added to the entities collection and saved. `CircleData` processes the entity further (lst. 4.23).

be added in a later stage of the prototype. Instead just the entities of the currently logged in user are visualized. The timeline is read-only, ie. the position of entities cannot be changed. The following list sums up the functional requirements:

- Visualize entities chronologically according to the creation timestamp.
- Widen and narrow the time range (zoom in/out) by scrolling.
- Move the beginning and end of the time range by holding the mouse button down.
- Cluster overlapping entities.
- Show contained entities on clicking the cluster.
- Allow for adapting the time range to show exactly the contained entities.

There exist several external JavaScript libraries for timeline visualizations. In order to assess the right one for the aforementioned requirements, several open source candidates have been evaluated to their modularity and adaptability primarily. Modularity means how reusable parts of the code

```
CircleData.init = function(vie) {
    this.vie = vie;
    this.vie.entities.on('add', this.filter, this);
    this.setIntegrityCheck(
        Voc.belongsToVersion, Voc.VERSION, Voc.hasCircle);
};
CircleData.filter = function(model) {
    if(model.isof(Voc.CIRCLE)){
        this.checkIntegrity(model);
        model.sync = this.sync;
    }
};
CircleData.sync= function(method, model, options) {
    if( method === 'create' ) {
        m.createCircle(model, options);
    } else ...
};
CircleData.createCircle = function(model, options) {
    var version = model.get(Voc.belongsToVersion);
    var data = this.mapAttributes(model);
    this.vie.save({
        'service' : 'learnEpVersionAddCircle',
        'data' : _.extend(
            data, {learnEpVersion : version.getSubject()}),
    }).to('sss').execute().success(function(savedEntityUri) {
        model.set('@id', savedEntityUri, options);
        if(options.success) {
            options.success(savedEntityUri);
        }
    });
};
```

Listing 4.23.: Excerpt of CircleData: On initialization the event handler `filter` for added entities and an integrity constraint is set up. `filter` checks the constraint and overwrites the circle entity's `sync` method. `sync` calls `createCircle` when `method` equals "create". `createCircle` finally invokes the SSS service `learnEpVersionAddCircle` passing the version URI to the parameter `learnEpVersion`.

Figure 4.24.: Propublica Time-setter

are without big changes. This includes that the library shall integrate well. Adaptability means how easily the code can be extended/changed to fit own needs. This includes setting own callbacks for user interaction and changing the appearance.

Propublica Time-setter

Propublica Time-setter[40] provides a bar with ticks for single events, coloured by group. Events can be hidden/displayed by respective group selection. There are two separate buttons for zooming in and zooming out in steps (not smoothly). Double-click on timeline zooms in too. On click on a tick the associated html is displayed below the timeline. Furthermore, there are two buttons other buttons for iterating over the events in chronological order (see fig. 4.24).

As for modularity, the timeline integrates view components which could be reused. Utility functions which calculate intervals and bounds of the bar for various scales could be reused also. Appearance fully customizable by CSS. Data can be ingested in JSON format on initialization and at runtime.

[40] http://propublica.github.io/timeline-setter/ (Visited: 2013-07-19)

However, no parallel, simultaneous events or time spans are possible. Additionally, it is not customizable where and when the detail view appears. This behaviour is hard-coded. Also hard-coded is the user interaction, ie. it is not possible to add one own's interaction hooks. Finally, ticks yield too little possibilities to visual entities appropriately.

Verite Timeline.js

Verite Timeline.js[41] provides timelines which can be moved on mouse down and zoomed via separate buttons. Events can be visualized with thumbnails and in parallel for simultaneous ones. The library offers big support for multimedia content (eg. image and video sources).

Although the appearance can be adapted by CSS overwriting, the whole library is monolithic and highly depends on Verite's Core JS. Data can not be added or changed at runtime. Verite Timeline.js is made for a specific use-case in mind, especially for visualization of events on news sites, which are meant to be static and need not much interaction on the level of entities.

SIMILE Timeline Widget

SIMILE Timeline Widget[42] consists of several bands of different scale of the timeline. On each band discrete events and time spans can be visualized. When one band is moved, the others move synchronously at the various speed relative to their scale. Detail information can be obtained by hovering the mouse over an entity, and the data can be filtered and highlighted on demand.

The appearance of the widget can be adapted via overwriting CSS classes. The programming structure of the timeline is also highly modular so that individual components can reused quite easily. Data can be loaded on initialization and at runtime dynamically. Developers claim the timeline to work also with a big load of events. However, it was not flawless to get

[41] http://timeline.knightlab.com/ (Visited: 2013-07-19)
[42] http://simile-widgets.org/wiki/Timeline (Visited: 2013-07-19)

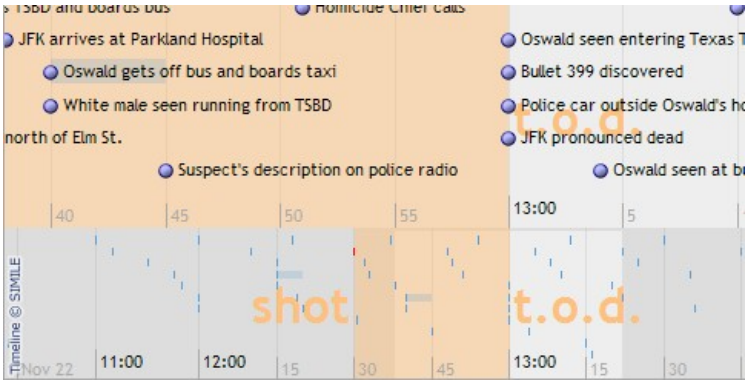


Figure 4.25.: SIMILE Timeline Widget

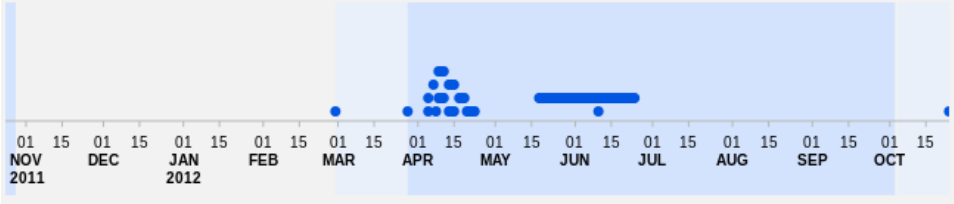


Figure 4.26.: Chronoline

the widget running since it is poorly documented. After all, despite its rich features - also with respect to modularity - the SIMILE Timeline widget has not been maintained since 2009 and hence yields the risk of having unfixed bugs.

Chronoline

Chronoline[43] timelines can be moved mouse down or on click on arrow buttons left and right of the timeline. Events and time spans are rendered as coloured bars, no thumbnail-based visualization of entities is possible. The title of the event is displayed on moving the mouse of it.

Chronoline depends on the popular visualization library Raphael[44] and

[43] http://stoicloofah.github.io/chronoline.js/ (Visited: 2013-07-19)
[44] http://raphaeljs.com/ (Visited: 2014-11-26)

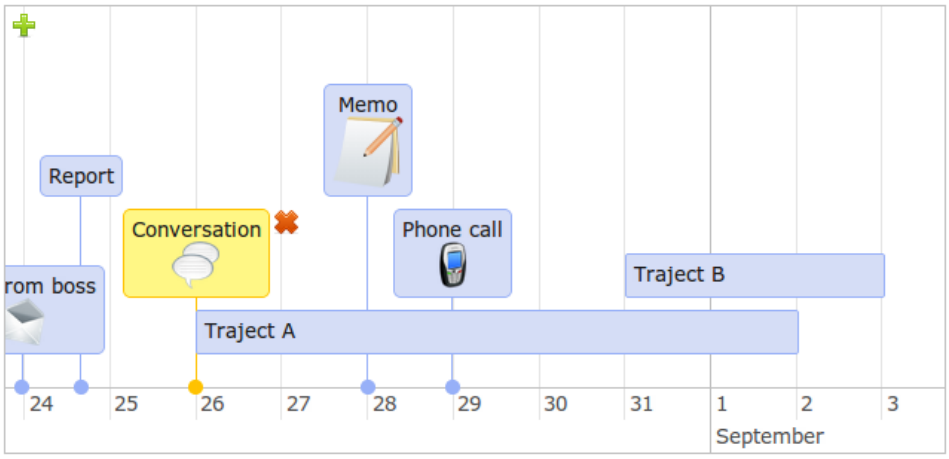


Figure 4.27.: CHAP Links Timeline

probably provides rich possibilities of adaptation of the visual appearance of the timeline. However, the code is monolithic and hard to reuse. Furthermore, Chronoline seems to be quite a young project with little community and hence with a higher probability of containing bugs.

CHAP Links Timeline

CHAP Links Library[45] is a collection of visualization charts, also containing a timeline. It provides interactive control for movement by mouse down and for zooming by scrolling, smoothly scaling from milliseconds up to centuries. Events can be ingested as JSON data and may contain simultaneous events as well as time spans. Events can even be separated in multiple lanes. Most remarkably, the timeline also provides clustering of events as soon as their bounding boxes overlap.

The appearance is highly customizable not only via CSS but also by injecting custom DOM elements. The timeline also offers a well-documented and rich API for controlling it programmatically. Additionally it comes with an event listening system, which specific callbacks can be registered with

[45] http://almende.github.io/chap-links-library/ (Visited: 2014-11-26)

in order to bind application functionality to user events happening on the timeline (eg. when changing the time range). Although not needed by B&P, the timeline also allows for editing of events direcly, either by moving or deleting them. For CHAP Links Timeline outweighs the other libraries in terms of adaptability, modularity and feature support, the `TimelineView` wraps its functionality instead of implementing a timeline on its own.

`TimelineView` consists of `UserView` and `ClusterHelper`, which consists of `EntityView` and `ClusterView` (see fig. 4.28). The view is bound to the timeline entity, which holds runtime data of the timeline such as the start/end timestamp of the current range, and to the user entity, which holds user events. `UserView` renders the representation of the user entity which `Voc.belongsTo` the timeline entity. `ClusterHelper` manages the transformation of entities to visible representations on the timeline, either as a `EntityView`, if an entity can be rendered as such, or as a `ClusterView`, if more than one entities are too close to each other. `TimelineView` consists of the following methods:

- `initialize` initializes an instance of `ClusterHelper` and binds event listeners to
  - changes of the set of entities in `Voc.hasUserEvent` of the user entity to call `changeEntitySet`.
  - changes of the start/end timestamps of the timeline entity in order to call `rearrangeVisibleArea`.
- `changeEntitySet` checks for added and removed entities in `Voc.hasUserEvent`. However, not the user events are to be visualized but the resource entities they reference, ie. the actual learning bit. Added resources are passed to `addEntities` and removed ones to `removeEntityView` of the `ClusterHelper`.
- `addEntities` calls `ClusterHelper.addEntityView` for each resource entity. If the timeline has just been initialized so that the set of new resource entities is the first ingest to the chart, the algorithm checks for the chronologically last entity and calls `browseTo` for this one.
- `browseTo` browses the timeline to the given entity.
- `render` actually creates the timeline chart object of CHAP Links Timeline and connects the timeline-specific event `rangechange` to the timeline entity. That way any moves and zooms made to the timeline by

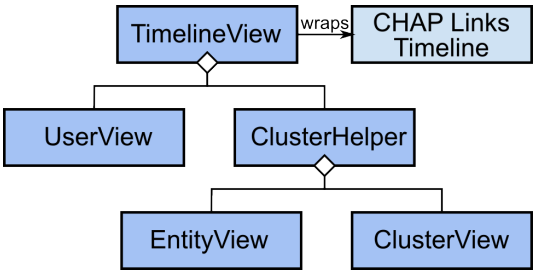


Figure 4.28.: The architecture of `TimelineView`

the user are passed on to the timeline entity and `ClusterHelper.reclusterByRangeChange` is called to calculate clusters for the new visible range. `render` also creates the `UserView` for the user entity and displays it next to the timeline. Finally, the user entity's `Voc.hasUserEvent` user event resources are ingested via `addEntities`.

- `rearrangeVisibleArea` reads changed values of the start/end timestamps of the timeline entity and passes them on to the timeline chart. However, if the change has come from a user-initiated event on the timeline chart, it is not propagated again to the chart. `ClusterHelper.reclusterByRangeChange` is called for the new visible range.
- `expand` is called on clicking the "<>"-button on a cluster. The resulting `zoomCluster` event contains the respective `ClusterView` instance. The timestamps of the first and the last entity of the cluster are used to set the timeline chart range accordingly.

4.3.6. Venn Diagram Widget

The Venn diagram widget allows for arranging entities spatially by dragging and dropping them within the canvas. Circles can be drawn around them and the label of the circle can be changed. Circles can also be dragged and dropped along with contained entities and moved within the canvas. Circles may overlap and entities may be placed inside the intersections. When moving one of such overlapping circles the other ones need to move too in order to keep the arrangement of entities within the overlapping circles. Functional requirements found here:

- Visualize entities spatially.
- Move entities spatially by dragging and dropping.
- Delete entities from the widget.
- Create and delete circles by double clicking into the canvas space.
- Change and move the label of the circle.
- Move circles spatially by dragging and dropping. Contained entities need to move along with it. Overlapping circles with entities contained in the intersections need to move also.

The view for the Venn diagram widget is called `OrganizeView` for the external visualization tool it is wrapping is named `Organize.js`. This tool provides the Venn diagram features of spatial arrangement of entities and circles and has been developed in the course of the Learning Layers Project. For its implementation, there were at least two possibilities: either using HTML5 Canvas or using SVG. Considering that with SVG, each SVG element is available within the DOM and therefore open to native event handling, and taking into account that such event handling does not exist in HTML Canvas, SVG was a natural choice.

There are several libraries available for easing the development of web applications using SVG: SVGKit[46], PERGOLA[47], Raphael[48], various JQuery SVG plugins and SVG.js[49]. After analyzing these libraries, SVG.js was the chosen one, since it is flexible, provides a rich set of library functions, has a good documentation and meets the needs of the Venn diagram widget. Along with this library, some plugins were also used:

- svg.draggable.js - for implementing the dragging and dropping functionalities within the SVG canvas.
- svg.foreignobjects.js - for the inline text input on the SVG canvas, which is necessary for entering the labels for each circle. In this case, a HTML text area was used as SVG foreign object.

`Organize.js` also supports user-driven changes. Data is not merely provided by the application but also by the user who may create, move, resize and

[46] https://github.com/SVGKit/SVGKit (Visited: 2014-11-26)
[47] http://www.dotuscomus.com/pergola/ (Visited: 2014-11-26)
[48] http://raphaeljs.com/ (Visited: 2014-11-26)
[49] http://www.svgjs.com/ (Visited: 2014-11-26)

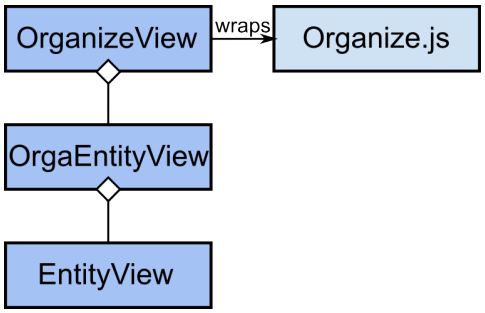


Figure 4.29.: The architecture of `OrganizeView`

delete circles or move and delete entities from the canvas. These changes are propagated by the component through DOM events which are caught by the wrapping view and then translated to further operations on the data layer. For the translation, `OrganizeView` internally maps IDs of SVG elements generated by `Organize.js` to entities, ie. `OrgaEntitys` and `Circles`. Additionally, the visualization of `OrgaEntitys` is managed by `OrgaEntityView`, which contains an `EntityView` of the referenced resource entity (fig. 4.29). `OrgaEntityView` renders this `EntityView` internally and wraps the result in an SVG representation which is then passed on to `Organize.js`.

`OrganizeView` consists of the following methods:

- `initialize` binds the event handler function `changeStuff` for the `change`-event to the version entity the organize entity belongs to.
- `render` calls `loadOrganizeCanvas` of `Organize.js` which initializes the SVG canvas. Additionally, `stealthContainer` is created and appended as a sibling node to the canvas. This container is hidden to the user and contains the `EntityViews` of `OrgaEntityViews`. Events coming from the SVG canvas (eg. `ClickEntity` can be translated to native click events on these hidden `EntityViews`. Other parts of the application can catch these events which would not be possible with the original events from the SVG canvas (see fig. 4.30). Finally, entities and circles are retrieved from the version entity and passed to `addEntity` and `addCircle` respectively.
- `changeStuff` checks whether there are changes to the sets of `Voc.hasEntity` or `Voc.hasCircle`. For each added orga-entity `addEntity` or

`addCircle` is called respectively to its type (`OrgaEntity` or `Circle`).

- `addEntity` (initial lower-case letter) adds an orga-entity to the view, coordinates already provided, by creating an `OrgaEntityView` of it, getting the SVG rendering and passing it to `createAndDropSvgEntity` of `Organize.js`. Additionally, the `EntityView` wrapped by the `OrgaEntityView` is rendered into `stealthContainer`.
- `addCircle` (initial lower-case letter) adds a circle entity to the view by calling `drawCircle` of `Organize.js`. Both `addEntity` and `addCircle` set up the mapping between entity and ID of the SVG element.
- `AddCircle` (initial upper-case letter) is bound to the `AddCircle`-event of the SVG canvas which is triggered when the user creates a new circle. It calls `OrganizeData.createCircle` with the generated data in order to create the circle entity which is mapped to the ID of the SVG element finally.
- `ChangeCircle` is bound to the `ChangeCircle`-event of the SVG canvas which is triggered when the user makes changes to a circle by either moving, resizing or relabelling it. The changed data is saved to the respective circle entity.
- `RemoveCircle` is bound to the `RemoveCircle`-event of the SVG canvas which is triggered when the user deletes a circle. It calls `destroy` of the respective circle entity.
- `ChangeEntity` is bound to the `ChangeEntity`-event of the SVG canvas which is triggered when the user moves an orga-entity. The changed data is saved to the respective orga-entity.
- `RemoveEntity` is bound to the `RemoveEntity`-event of the SVG canvas which is triggered when the user deletes an orga-entity. It calls `destroy` of the respective orga-entity.
- `ClickEntity` is bound to the `ClickEntity`-event of the SVG canvas which is triggered when the user clicks on an orga-entity. The event is translated to a native `click`-event and triggered on the `EntityView` contained in `stealthContainer`.

There is neither an event nor a method `AddEntity` (initial upper-case letter) since entities can only be added via dragging and dropping an entity from the upper widget. This case it handled by `WidgetView` (see lst. 4.31). Fig. 4.32 shows how the application components play together by example of dropping an entity to the Venn diagram widget.

```
render : function() {
    ...
    this.stealthContainer = $("<div style=\"display:none\">");
    this.$el.append(this.stealthContainer);
},
addEntity : function(entity) {
    var view = new OrgaEntityView({model:entity});
    this.stealthContainer.append(view.resourceView.$el);
    var id = this.organize.createAndDropSvgEntity(view.getSvgData());
    view.setSvgId(id);
    ...
}
```

Listing 4.30.: StealthContainer: The `render` method creates the hidden DOM element, stores its reference in `OrganizeView.stealthContainer` and appends its to the DOM element (`this.$el`) of `OrganizeView`. `addEntity` appends the DOM element of the `EntityView`, ie. `view.resourceView.$el`, to the `stealthContainer` from where events can be triggered and propagated through the application.

```
organizeBody.droppable({
    drop: function(event, ui) {
        var id = ui.helper.attr('about');
        var offset = $(this).offset();
        var orgaEntity = {};
        orgaEntity[Voc.x] = ui.offset.left - offset.left;
        orgaEntity[Voc.y] = ui.offset.top - offset.top;
        orgaEntity[Voc.hasResource] = id;
        OrganizeData.createEntity(that.model, orgaEntity);
    }
});
```

Listing 4.31.: JQuery droppable: `organizeBody` is the (JQuery wrapped) DOM element containing the Organize Widget. `drop : function()` is called as soon as the user drops the dragged element (see lst. 4.14). `ui.helper` contains a clone of the dragged element which has been marked up with RDFa (the attribute "about") by `EntityView`. `that.model` is the organize widget entity, `orgaEntity` is created with the coordinates of the dropping position and a reference to the dragged resource entity.

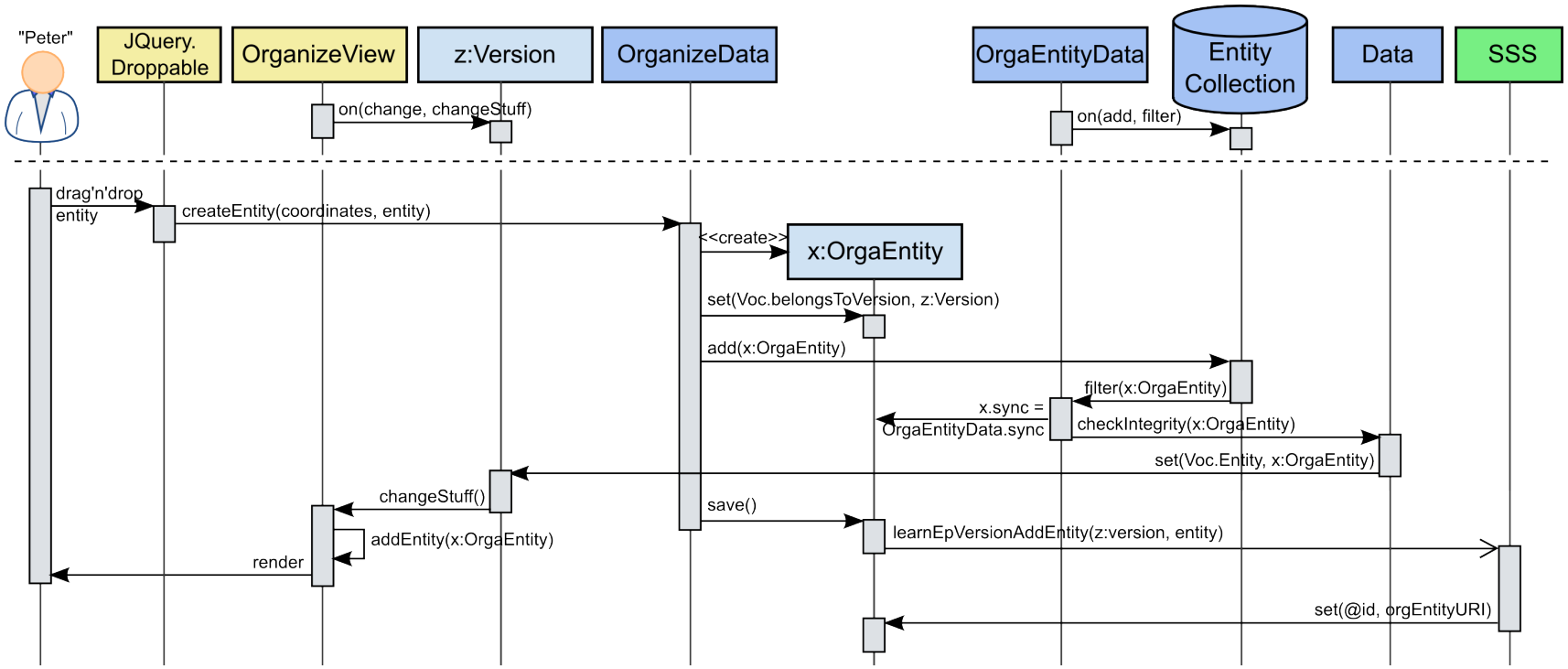


Figure 4.32.: Sequence diagram of dropping an entity to the Venn diagram. Components have been coloured by the architectural layer they belong to: yellow - presentation layer, blue - data layer, green - service layer. Dark blue - data modules, light blue - entities.

5. Evaluation

Bits and Pieces has been evaluated on two levels: on the level of the usability of a first functional prototype including the timeline and the organize widgets from the previous example, and on the level of the framework's technical implementation. The first was part of the co-design research approach (Tomberg et al., 2013) conducted in the course of the Learning Layers EU-Project to investigate sensemaking in the context of informal learning. It consisted of user tests with four representatives from the health care domain. The results of this study have also been published by Dennerlein et al. (2014). The second evaluation tested the implementation of the qualities of sensemaking, which were elaborated and leveraged throughout this thesis, against the qualified view of two experienced Web development experts.

While the evaluation of the user interface revealed some flaws on the surface level which partly can be traced back to the framework design, the results of expert interviews yield an overall match of the theoretical foundation and its implementation.

5.1. Usability Study

The four participants in the usability study were two diabetes specialist nurses (DSN), one healthcare assistant (HCA) and one doctor (GP). The main goal of the study was to investigate informal learning at the workplace. Hence, the procedure consisted of two phases:

1. a two-week collection phase for gathering bits and pieces of information (e.g. notes, Web links, pictures) related to a chosen realistic learning need (e.g. find out about methods for reversing diabetes) alongside the usual workday using Evernote

2. a supervised sensemaking phase using a prototype of B&P which consisted of a timeline widget and an organize widget.

Both phases included a short training of the respective tool. Additionally to the mobile Evernote App, the participants were introduced to sending notes via email to their Evernote account due to their rather low technical affinity and hardly available mobile devices. Then the task for the collection phase was formulated the following: "Please, collect all learning experiences, discoveries of knowledge gaps, interesting findings, unsolved solutions, raised questions, noticed faults, learnings (human and/or material caused) etc. alongside the normal workday in Evernote notes: i.e. picture, text, video, audio, picture annotation (marking stuff in pictures) and scanning of documents".

The latter phase took place two week later. Its central part was a fifteen minutes thinking-aloud usage of B&P. The participants' task was to make sense out of their own gathered bits: i.e. to first remember their informal learning experiences based on the bits with the help of time-based cues in the timeline widget and afterwards structure them semantically with the help of the circles in the organize widget. The task was formulated the following way: "We'd now like you to use the B&P tool to explore and organize the material that you gathered during the collection phase. The aim of this study is to give you the chance to use this tool in a realistic context and to gather your feedback on the tool and its effectiveness". Hence, the second phase closed with a semi-structured interview asking about the participants' general attitude towards the prototype, its usability and performance and the efficiency of the tool's support for remembering and sensemaking their past experiences. The thinking-aloud and interview records were transcribed, paraphrased and categorized in an inductive, iterative process (Mayring, 2000).

5.1.1. Results

On average, the participants gathered eleven information bits over a period of two weeks (n=4; M=10.5; SD=2.52). The number of collected bits during the collection phase was rather low due to the voluntary participation, the

workload the clinicians were facing during study time and problems with the unfamiliarity in using Evernote which influenced the study participant's motivation and/or ability in recording their own traces of informal learning experiences. The sensemaking phase worked out well except that the time needed for acquainting the participants with B&P was slightly underestimated. This resulted in more time spent on introducing B&P and a reduced sensemaking time for some of the participants.

In general all four participants were positive about the B&P tool, i.e. the study participants understood and appreciated the underlying idea and purpose. While for the study each participant had to choose a real learning need they currently faced, their motivation in using the tool appeared to vary. For the GP it was important to gather evidence for her revalidation, others saw it as an opportunity to pursue personal learning challenges or collecting information with the main purpose to share with others. For the latter, therefore, participants used Evernote mostly to gather todos for, eg., reading and saving documents, Web links, etc. B&P was observed as being a responsive tool by three of the participants, although only easy to use by two of them.

Timeline Widget

One participant's thinking aloud protocol clearly indicated that the timeline is working properly to prompt remembering informal learning experiences:

> "That's the card sort photo, I remember that [...]. Just check I've remembered all the things I thought about last time."

However, although the idea was generally appreciated, most of the participants wanted extra cues to increase the chance to find and remember an informal learning experience, such as information on the involved persons or institutions, location, content and relations to other events. This confirms the design decision to go for a widget-based framework easing the implementation of additional widgets with focus on required cues.

Furthermore, an application of the action-oriented approach to sensemaking (Dervin, 1998) could also be found:

> "[D]o I still need this? If I do need this, what am I going to do with it?"
>
> "What you really need, is to be able to go back to something you've collected, go into it, read it and then to be able to have a link where you could make your own notes and they would then be saved there, so that when you brought them down, your notes would be organised."

Consequently, most participants requested to directly manipulate the title, tags or importance of bits and attach a comment, reminder or deadline favorably accessible via context menus. As for the narrowing down of information, timeline anchors, searching functionality as well as filters for quicker browsing and comparing of bits was desired. These feature requests reflect the need for interactivity while sensemaking and support the respective design decision.

Organize Widget

Overall, the interactive nature of sensemaking via the organizing and categorizing of bits into circles was valued by the healthcare professionals, particularly the opportunity to break topics down further was appreciated. In the same lines, the request to create circles within circles, to enable the creation of subcategories, was formulated as well.

One participant was especially successful in her sensemaking effort as she managed to meaningfully group her collected informal learning experiences on diabetes treatment according to their influence on blood pressure:

> "[P]eople's blood sugar levels have gone down through exercise as opposed to diet."

So, skim reading in the sensemaking phase resulted in a first internal representation of her learning, which led to the creation of a corresponding external categorization in the circles "Results from diet" and "Results from exercise" and their overlap representing results from both. It turned out that people needed more time to actually grasp the content of collected bits, understand their relevance and re-evaluate them. The lack of time for this in

the study design may have hindered their chance to engage in sensemaking. It is likely that the involved representations would have been enriched with more data, adapted and matured over time.

Finally, there was a strong demand of two participants to make the transition of pieces of information from the timeline to the organize widget permanent, i.e. the dragged bits disappear from the timeline. One participant puts it this way:

> "I'm thinking that because it's still up on the timeline but I've actually dragged it down into Concept One, [...] so I might come back to that later and think "oh well it's still on the timeline", but in actual fact I've actually already dragged it down into Concept One."

This feature request is in line with underlying theory assumptions that once sense is made of a bit (ie. interaction happened on it), the user expects an appropriate change of the representation.

5.1.2. Discussion

Although it was not the main goal of the study to investigate the feasibility of the sensemaking qualities, some could be seen as approved by the users. First, their thinking-aloud underpins the *individuality* of sensemaking. They reviewed their gathered bits and tried to make sense out of them on their own. Second, their strong request for more cues than the time-based one proves the *context-sensitivity* of sensemaking. Concerning the *enactivity* several indications exist which support its importance. Participants generally appreciated the dragging and dropping of bits around in space and asked for more interactive features several times such as timeline anchors for quick and efficient navigation and direct commenting of bits.

As for the *continuity* and the *reification* quality, these could not be evaluated in the short time available on the one hand, and due to lack of more widgets allowing for other - reifying - structures. The *bipolarity* could not be validated because there has only been tested the one interface with the two canvases. Next, the participants consecutive switching between the browsing and the

organizing widget give at least a weak suggestion that sensemaking indeed is a *reciprocal* process of interchanging bottom-up (circle formation due to found bits) and top-down sub-processes (finding other bits to fit into the circle). However, the amount of collected bits was too low at all to assess that properly.

5.2. Expert Interviews

In order to assess whether the qualities of sensemaking are reflected in the implementation of the framework two expert interviews have been conducted. Both experts are graduates in the field of computer science and have long-term experience in Web development technologies. Before the interview, both experts were introduced to the qualities of sensemaking and to the implementation of the framework. They were also given time to investigate the source code on their own. Expert 1 has been working with us on top of the B&P framework for four days before the interview. Though it must be said explicitly, that the expert was not involved in the development of the framework but participated in the work for future improvements. Expert 2 has not been working with the Learning Layers project at all, not to mention with B&P, before. Because of this, more time was assigned to the interview of Expert 2 for code introduction, whereas Expert 1 had been working with the code the days before already. So the interview with Expert 1 lasted approximately one hour while the one with Expert 2 took about two hours.

At two distinct sessions both experts were asked to give their opinion on the implementation of the qualities of sensemaking while looking at the framework's source code. Our hypotheses for each quality to be reflected in the implementation were:

- **Individuality**: due to the user's data per se.
- **Context-sensitivity**: due to extensibility of the widget set.
- **Bipolarity**: due to two canvases shown in parallel, one for information foraging, one for information organizing.

- **Reciprocity**: due to the reciprocal influence of the canvases on each other, finding representations in the lower canvas on top of data in the upper one, and finding more data on top of representations in the lower canvas.
- **Continuity**: due to continuous preservation of the sensemaking state into an anytime serializable data structure.
- **Enactivity**: due to responsiveness and interactivity hooks on each bit.
- **Reification**: due to having structures as well as bits stored as entities in the base collection.

Detail questions were interposed where needed to clarify standpoints. With the permission of the interviewees, the interviews have been recorded, transcribed, irrelevant talking removed and finally paraphrased. The paraphrased transcripts can be found in the appendix A.

5.2.1. Results

The overall picture of the resulting judgement by the two experts is clearly positive in most points. Some bigger caveats were uttered in the course of context-sensitivity and reciprocity, mostly due to the early state of the prototype and lacking features. The statements in detail:

Individuality

Both experts confirm that individuality is more a matter of the data used rather than an implementation feature, for instance Expert 1:

> "From the standpoint of the user, the individuality should be there, because you are dealing with the persons individual data that has been collected." (l. 36-37)

Context-sensitivity

In the view of Expert 1, context-sensitivity is "limited to the time-frame" because "there are no other possibilities to organize the data" (l. 40-43) but admits that this "is due to early version of the prototype of the framework and due to not having enough resources to implement other contexts" (l. 47-49). On arguing for context-sensitivity being implemented by the extensibility of the set of widgets, Expert 1 certifies that "the codebase is more or less ready to provide other widgets" (l. 46). On the other hand, Expert 2 is sceptic about the hard-coding of widget creation in `WidgetView` (l. 282). Explaining that this is because the design feature for switching widgets left/right was not implemented yet, he concludes that "in this initial prototype that's ok" (l. 292). So due to lacking widgets, the experts can only affirm our hypothesis in relation to the framework's prototypical state.

Bipolarity

This quality is clearly approved by both experts. Expert 2:

> "You have the timeline. [...] The user can find a bit. So that's true, you got information foraging. And you can really organize the bits, place them in circles, so I think it's also covered. And both canvases can be used in the application at the same time. Yeah, I think that's ok." (l. 380-387)

Expert 1 additionally points out that if widgets "would be placed in the same spaces on the screen" (l. 130) depending on whether they are of "browsable type" or "organizing type" that this distinction would help the user to understand the two roles of the canvases, hence the two poles of sensemaking.

Reciprocity

Expert 1 says that "both approaches could be applied" (l. 96), referring to bottom-up and top-down processes although he adds that this is a

hard question unless "a fuller functional prototype with some additional functionalities" would be developed "[s]o that it would become more clear" (l. 97-99). So he sees the foundation for reciprocity being realized.

Expert 2 identifies the feature of dragging and dropping bits from the upper to the lower canvas as an implementation of reciprocity (l. 319) which indirectly matches our intention because this feature at least enables the building of representations upon given data. Additionally Expert 2 highlights the extensibility of the widget set as the foundation to a later-to-come switching feature of widgets which allows for finding better matching widgets (representations) (l. 324-325). These statements partly confirm our hypothesis.

Continuity

Expert 1 confirms that by continuously saving the state of the sensemaking continuity "is fully covered by the prototype" (l. 79). From a framework perspective he also attests the same saying for the future implementation of other widgets: "I mean it's one of the basic logics of the framework." (l. 88). Expert 2 investigated deeper whether developers are forced to implement the data saving interface in data modules (l. 296) which is not the case on a code-level. However, stressing that VIE's graph-like database allows for serializability of any kind of data which, together with the asychronous service layer, the expert confirms continuity:

> *Interviewer:* "So do you think that continuity is represented in this?"
> *Expert:* "My answer would be yes. But could you just show me the code where you save the state? [...] [reading code] Ok that's actually clear. And the continuity is pretty clear." (l. 308-313)

These statements affirm our hypothesis.

Enactivity

Throughout the interviews the terms "enactivity" and "interactivity" have been used synonymously for easing the understanding. Here, Expert 1 stresses that the framework does not allow for bit-specific interactivity, "[f]or example, if one bit is a video the framework doesn't allow you to play that video in some embedded element" (l. 53-54). So he would expect future improvements in that direction. Furthermore, it has been argued that `EntityView` "is making every bit interactive" (l. 240), including their dragging and dropping. Expert 1 appreciates that:

> "[...] it's some single entry point for interaction with the entity, and any places it's put, because the bit is being rendered inside the DOM too. So it makes perfect sense and should be a good solution." (l. 75-77)

Expert 2 highlights positively that other plugins could be used also to extend interactivity (l. 247). While these statements prove the interactivity hooks of bits, we missed to assess the framework's responsiveness in term of this sensemaking quality unfortunately.

Reification

Expert 1 affirms the question whether the reification of structures could be possible with the framework, eg. letting episodes appear as entities in widgets, but reminds of his statement concerning interactivity at the same time:

> "Judging by the logic of the framework that should be doable without too much work. And it should be working. [...] this is an entity but in addition it's an episode entity. [...] it's acting as if it was an entity, but when we again go back to the interactivity, [...] it should show you that there are some multiple things connected to it and somehow visualizes that to the user in some nice way." (l. 115-120)

So his only concerns are that in his mind a "bit" should be a simple thing whereas reification makes it become complex stuff. However, he admits that the framework could go beyond that notion. Perhaps due to our short-handed description at l. 163-167, Expert 2 did not get the idea of this quality right as l. 350-354 suggest. It might be that his understanding was more focused on take-up of sensemaking, working upon a previous sensemaking state, instead of reifying representations into data for another sensemaking process. However, the foundation for allowing reification, ie. having everything - structures as well as bits - stored as entities, is finally approved useful for preservation (l. 378). After all, our hypothesis can be seen as granted by the experts.

5.2.2. Discussion

The expert interviews draw a quite positive picture of the implementation of the sensemaking qualities. For most qualities they confirm our respective hypothesis. Individuality is seen implemented by leveraging the user's personal data collected via Evernote. The two canvases are also understood as reflecting the bipolarity of sensemaking. While continuity could not be tested well by the usability study, it was finally approved by the experts as a foundational feature of the framework by the division of the data from the service layer. Implementation of enactivity was approved as dragging and dropping is realized in the scope of the entity as the "single entry point for interaction". Reification, being an abstract but nonetheless essential quality of sensemaking, was certified by the experts in the principle design of the framework though concrete widgets would be needed. For the same reason, the experts could not clearly affirm context-sensitivity since this quality relies upon a heterogeneous set of widgets serving various contexts which could not be implemented at the current state of B&P. Likewise, reciprocity, which was hard to understand for the experts, only got a weak kind of confirmation, mainly due to sparse widget functionality which obviously made it difficult to apply and visualize this quality of sensemaking.

However, if more widgets could have been implemented as well as the feature of instant switching of widgets left/right, as it is envisioned by the framework design, the expert might have found context-sensitivity and

reciprocity better represented. On a code-level at least Expert 1 pointed out that the framework has the foundation for fixing these issues.

After all, these results indicate that B&P is going into the right direction for supporting sensemaking effectively. As for context-sensitivity and reciprocity future improvements must include the implementation of more widgets and the switching of them. These features were also missed by the users in the usability study (5.1.2). More widgets would also give more insight into the feasibility of the framework in terms of reification. More detail information on an entity, especially bit-specific, should also be provided by the framework as Expert 1 suggested. Again, this request is also in line with that from the users who asked for more direct interaction with bits.

6. Conclusion

The thesis' research question was how software can support sensemaking on the Web. In order to answer this question, this thesis takes a strict theory-driven approach. First, theory on sensemaking from the perspectives of Dervin, Russell et al. (1993) and Pirolli and Card (2005) has been summarized in order to frame the field. Dervin's Sensemaking Metaphor helps in understanding an individual's sensemaking of the world as a matter of communication, whereas the approach of Russell, Pirolli and colleagues views the topic in the terms of human-computer and human-information interaction. Within this field, seven "qualities of sensemaking" have been named as landmarks: Individuality, context-sensitivity, bipolarity, reciprocity, continuity, enactivity and reification.

Referring to the introductory example from the beginning, their essential role for sensemaking can be explained as follows: For the sensemaking of "sensemaking" (1) I was working on my own (*individually*), collecting literature from Web catalogues, making notes in my text files. (2) My way of sensemaking highly depended on the *context* of my data, ie. publications from various scientific fields, and on the context of my goal, ie. writing a thesis. Hence I was looking for papers based on authors, categorizing them, creating sample outlines. (3) My whole sensemaking began with mere literature search and ended with a structure of publications fitting my need, ie. it spanned between the *two poles* of information foraging and information organization. (4) Throughout this process I was creating rough categories, trying to find more data which might fit and eventually refining these categories, so the data influence my structure building, and the structures influence my data search *reciprocally*. (5) The whole process took several weeks, *continuously* taking down and up my sensemaking in-between several times. (6) There weren't hardly any longer periods of time where I found myself merely thinking about the topic. Instead I was *actively* scratching one

itch after the other, searching, categorizing, writing things up. (7) Finally I did put my categories of publications on sensemaking in the outline of the chapters on theory and related work, more or less *reifying* categories to subsections.

These seven qualities allowed for systematic evaluation of existing user interfaces for sensemaking on the Web. Four approaches, CoSense (Paul and Morris, 2009), ScratchPad (Gotz, 2007), Coalesce (Ryder and Anderson, 2009) and Apolo (Horng et al., 2011), were reviewed by looking into their functionality in detail and sheding light on them from the perspective of the qualities. There, individuality and reciprocity scored the highest, whereas context-sensitivity and reification seemed to be the most under-represented ones. Hence, the thesis' approach was especially focused on the two latter qualities to implement.

Given this review, non-functional requirements and user interface design features for the novel widget framework approach - Bits and Pieces (B&P) - were deduced from the seven qualities. Individuality is represented by leveraging the user's own data. Context-sensitivity is enabled by an extensible set of widgets, each possibly serving the requirements of a specific context. Bipolarity is reflected by the two types of canvases, one for each pole of sensemaking. Reciprocity comes by presenting the two canvases on top of each other, almost literally allowing for the reciprocal interchange of top-down and bottom-up sub-processes. Continuity is maintained by continuous preservation of the sensemaking state in the background. Enactivity demands the user interface to be responsive and to make every entity interactive. Reification is made possible by leveraging an openly structured graph database which allows for representation of data and meta-data in RDF. The B&P design idea was also developed in the strand of the EU project Learning Layers[1] and validated with representatives from the health care sector.

The main part of the thesis was to describe the implementation of the framework in detail. It is implemented in JavaScript on top of cutting-edge Web technologies such as HTML5, JSON-LD and RDFa and relies on Backbone, VIE and JQuery as Web frameworks. The software architecture is organized into three layers: the presentation, the data and the service

[1] http://learning-layers.eu (Visited: 2014-11-18)

layer. The presentation layer consists of view components for rendering and visualizing content as entities, widgets or user interface control elements. The data layer covers data logic and is organized in modules for various aspects of the user data (eg. user events, circles, versions, etc.). Data modules are backed up by services in the service layer. In the case of the current implementation this layer makes heavy use of services of the Social Semantic Server (Kowald et al., 2013). Furthermore, two widgets have been developed on top of the framework: The first visualizes entities by their creation timestamp along a timeline, the other enables the user to create circles and place entities spatially inside them as in a Venn diagram.

Finally, a user study based on this prototypical application of the B&P was depicted in the last chapter and the results were discussed with respect to the qualities of sensemaking. Additionally, two Web technology experts were interviewed for their standpoint on the implementation of the qualities on the level of source code and architectural design. While the user study revealed that Bits and Pieces is able to support sensemaking of one's own learning experiences collected on the Web on a user interface level, the expert interviews confirmed that the qualities of sensemaking are well reflected in the framework implementation.

These two sets of positive results underpin the feasibility of the theory-driven approach and the feasibility of B&P as a framework for sensemaking on the Web. Hence as the final outcome of this thesis and as an answer to the research question drawn in the beginning, it can be stated that a software tool can claim to support sensemaking if it implements Bits and Pieces' seven qualities of sensemaking.

6.1. Contribution

With the elaboration of the proposed seven qualities of sensemaking, this master's thesis provides a theoretical framework for sensemaking in the context of HCI and for the development of respective user interfaces. Based on these qualities, it derived, designed and implemented a generic widget framework easing the creation of context-specific applications for sensemaking on the Web.

6.2. Limitations

The proposed software framework B&P does not solve the question which user interface is best suited for sensemaking on the Web. This is mainly due to the context-specific quality of sensemaking. Instead, the framework just allows the implementation of specific widgets while giving examples as the timeline and the Venn diagram widget. This might be seen as a limitation or as a strength.

Furthermore, the framework could only be tested on a small scale with the two aforementioned example widgets. However, as one of the experts pointed out, more widgets would be needed to assess the feasibility of the approach in full extent.

This thesis does not cover one aspect of sensemaking which might also play an essential role: the social aspect (Weick, 1995). Sensemakers also influence each other by sharing certain artifacts. B&P does not contain sharing functionality specifically. This could be implemented as another service in the service layer which merely adds more base data to the application, though originating from others. However, it remains unclear to which extent such a sharing feature would influence the whole individual sensemaking process.

6.3. Future Work

Apart from the extension of the widget set together with the switching feature, specific functionality for certain types of bits will be added on a global level, eg. providing for video playing, document rendering etc., instead of the mere meta-data overview currently displayed in the sidebar.

Another global user interface feature to be taken into consideration is a generic filtering function which enables the user to filter bits from the canvases independently from the widget implementation. For instance, one could hide all bits which do not contain PDF documents irrespective whether the bits are visualized in a timeline, a Venn diagram or any other kind of widget.

On the level of code, there is a considerable amount of repetitive pattern in the data modules, eg. the declaration of the *filter* method as a callback for a specific entity type being added to the base collection. It might be feasible to model the application domain using an ontology language and compile it to JavaScript data modules.

Finally, the service layer could be extended to support the smart invocation of Web APIs which implement Hydra (Lanthaler, 2013), the vocabulary for interoperable and hypermedia-driven Web APIs[2]. No more API-specific wrapping into the service layer would be needed then and B&P could be even better engaged for sensemaking on the Web.

[2] http://www.hydra-cg.com (Visited: 2015-01-15)

Appendix

Appendix A.

Expert Interview Transcripts

A.1. Expert 1

Sensemaking is a kind of process of sorting out bits and pieces in order to achieve a certain kind of goal. The data you are dealing with is huge and heterogeneous. You need to interact with the data and find structure. Then you're going to try to find data which fits into that structure. It's a kind of interchange of bottom-up and top-down processes. Bottom-up is finding structure based on data, and top-down is find data which fits into the structure. There may be data which do not fit into the structure, residues so to say. Then it might be needed that you change your structure. So you evolve the structure more and more by interchanging this top-down and bottom-up process. That clear?

Yeah.

Now I have come up with a couple of qualities of sensemaking. These are the individuality: So sensemaking is a personal process. It's always your own goal you try to achieve, it's specific to you. Context-sensitivity is specific to the situation, to the goal. Enactivity, or interactivity: Sensemaking is grounded in action, which means that you have to interact with the data, you have to do something with it. The continuity quality is that sensemaking can start and stop any point, so as soon as you found that you have reached the goal. Or you might come back to sensemaking and proceed from that point, or start from one else's sensemaking for instance, or start with a predefined structure.

Ok.

Reciprocity is what I've mentioned already, interchange of top-down and bottom-up processes.

Mhm.

Reification is meant that the output of a sensemaking's structure can again serve as input to sensemaking. So you find for instance the structure of category, and in the later step you might use that category as a kind of data. So in another sensemaking

session, you might find meta-categories. So the category then is reified, it is made to a thing.

Ok.

Bipolarity means that sensemaking is between these two poles of information foraging and information organizing. The one end is more concerned with searching for information, and by doing that you start finding structure, and continuously it grows and more and more into the organizing pole of the sensemaking.

Ok.

Fine, so let's start with the individuality. So the general question is: Does the framework in it's current implementation come up to this quality of sensemaking of individuality.

I suppose yes. From the standpoint of the user, the individuality should be there, because you are dealing with the persons individual data that has been collected.

Ok. So for the context-sensitivity. Do you think that the framework serves context-sensitivity in the term of that it can be adaptable to context?

If you try to use the context-sensitivity based on the timeline representation of the Bits and Pieces then it has context-sensitivity, time-frame context-sensitivity and as there are no other possibilities to organize the data, then this context-sensitivity is limited to the time-frame. So it has limited context-sensitivity.

So from a framework perspective, could one implement more tools which provide more context-sensitivity in your sense?

Well, why not, the codebase is more or less ready to provide other widgets with other structure of data or other context widgets, so yes. It should be doable. I'd say, that context-sensitivity is limited is due to early version of the prototype of the framework and due to not having enough resources to implement other contexts. So this is the first step to do something and it makes sense.

Ok, fine. For the enactivity, or interactivity: In your opinion, does the framework support the implementation of an interactive application?

Well, to some extent. For example, if one bit is a video the framework doesn't allow you to play that video in some embedded element, or just have the preview of the file. So the only interaction of the moment is getting some limited amount of information about this bit. And well, you could define interaction as dragging it into the organize canvas. That's also an interaction at some point of view. You could be interactive but the ease of data representation might be improved, it could be a bit more bit specifics. It might not be a limitation of the prototype itself even but might be a feature. But that would probably benefit the user, it would be great to get a quick preview of the bits somehow, just to understand what you are dealing with. The timeline widget is kind of limited to the time

ranges, so when you are looking at some information that was collected two months ago, you might not remember the specific context.

Mhm. In your mind, how hard would it be to implement a bit-specific representation on top of the framework?

Well, it would take some time plus as the framework is consuming a service, it depends on the services also. The service should be providing the tools for the framework to get its job done. So where the tools are provided it shouldn't be too hard or too time-consuming to provide some additional information in some nice way based on the type of the bits and pieces there.

Ok, concerning the drag and dropping you also mentioned. In the EntityView, we've got the draggable [method] which is called after rendering the entity, so effectively everywhere I render this EntityView it will be draggable. Do you think this is a feasible approach in order to support the interactivity in any point of the application?

Sure, well it's some single entry point for interaction with the entity, and any places it's put, because the bit is being rendered inside the DOM too. So it makes perfect sense and should be a good solution.

Ok, as for continuity, so that one can start and stop sensemaking at any point.

I think this one is fully covered by the prototype, in regard to this organize canvas. Everytime I do something, its state is being saved to the database. In addition there are possibilities of creating multiple versions, so I can save some states, start a new one, do some additional sensemaking and go back to some of the previous state and see what I was doing. So I guess it's even providing more than just a simple continuity. It's even providing some versioning.

Ok, and now again from a framework perspective, if one implements another widget based on the framework, do you think the continuous preservation of the state could also be used in this new kind of widget?

Sure, I mean it's one of the basic logics of the framework. And this logic is enforced by the framework. I suppose yes, it should be.

Ok. For the reciprocity. Interchanging of bottom-up and top-down sub-processes. Do you find it represented in the implementation of the framework?

I suppose by bottom-up you think that the pieces are being browsed and look at and some decisions are made based on that, then there the organize canvas is used somehow to group something and make some logical conclusion based on top of that and then that one, the result of it can also be applied to those initial bits and pieces to re-organize the logic of the canvas. Then I suppose both approaches could be applied, but that is really a hard question to answer. So in order for it to be easier to answer, a fuller functional prototype with some additional functionalities might have to be extended. So that it would become more clear.

Fine, then we come to next abstract quality. It's the reification, so the output of sensemaking could serve as the input to sensemaking itself. Do you find that represented in the implementation?

I mean it depends on which standpoint you take, so for example there is this information and the sensemaking is done with it, then there should be the possibility to save the results of your sensemaking and then reuse it within the process of some other sensemaking. I'd say to some extend this prototype offers that. Unfortunately it's impossible to apply the result of one sensemaking process to another. So you cannot create bits from the result of the canvas and reuse it on another canvas. Maybe that would be just a super-result. [laughs] But I'm not sure whether that makes sense for the system.

It might. That's what reification is about. But could you imagine that for instance the episode which represent some kind of the result of a sensemaking could be displayed in the timeline? As an episode entity. And you could use these episode entities, drag and drop them, and organize them. So in a kind of meta-episode. Or a super-result as you put it. Could you imagine that the framework could support that?

P: Judging by the logic of the framework that should be doable without too much work. And it should be working. Again if we're talking about entities, this is an entity but in addition it's an episode entity. And it's again draggable, droppable, it's acting as if it was an entity, but when we again go back to the interactivity, instead of just showing you one thing it should show you that there are some multiple things connected to it and somehow visualizes that to the user in some nice way.

Yes, there would be another widget needed to visualize the episode and its connected stuff. And for the bipolarity, which is the sensemaking ranging from information foraging, up to organizing. Do you think the framework represents this bipolarity?

Sure, this bipolarity is there, so you can search for the information, browse the information then you organize it to these episodes and then you can search again some more. So the current prototype is serving that purpose.

There is also this "browsable widget type" and this "organizing widget type" we've been talking about. Does it make sense that other widgets inherit one of these types, in order to run into one of these two polarities?

Yes, well, and if they would be placed in the same spaces on the screen, then that would surely provide an information to the user that this upper end of the screen is for getting the data, and the bottom end of the screen is for managing the data.

Ok, fine.

A.2. Expert 2

Sensemaking deals with large amounts of information, which is unstructured. Here we deal with bits and pieces the user collected through Evernote. Users want to find a representation for aspects of their data, which depends on the goal. The goal needs not be pre-defined, it's maybe just an idea. Users try to find a representation for the data and given this representation they try to find more data which fits into that representation. Data which does not fit the representation are the residues. These lead to a representational shift. The user adapts the representation, looks for a new representation. Sensemaking works in a loop of a bottom-up process which is going from the data to a representation, and a top-down process, getting from a representation to the data. Finally a good representation fulfills a goal. Sensemaking also is an interactive process, so you must do something.

Ok.

Sensemaking is a ongoing process. There is no defined start or end. The input to the sensemaking process can be any kind of data, it need not be the absolute raw data. It could be something more advanced. as soon as the goal is reached, the user might stop immediatly. he maybe has found some structure and somehow can use it. These are the qualities of sensemaking I've developed in the thesis 1. Individuality: Sensemaking is a personal process. You mostly do it for yourself. There is no collaboration per se.

You are specifically avoiding influence coming from other people. You want to make sense alone.

Other people might be also doing sensemaking and then they're sharing. It's not so inherent to sensemaking itself, the social aspect. 2. Context-sensitivity: Sensemaking is specific to the sensemaker's situation, to his goal, to the kind of data he's dealing with, to the kind of device which is used.

You are targeting no specific context.

3. Enactivity: You have to be active, you have to do something while sensemaking. It is not a passive process, merely thinking. 4. Continuity. It never starts or stops. It is an ongoing process, it might stop at any point, it might start at any point. 5. Reciprocity: It's the interchanging of the bottom-up and top-down processes, finding the representation from the data and finding data on top of a representation. 6. Reification: a representation might serve as the input for the sensemaking process. So it might become data. For instance, if you tag something and then in a later phase you use the tags themselves to do more sensemaking, group the tags or something. So in sensemaking what is data and what is representation is not so crucially devided. 7. Bipolarity: There are two phases in sensemaking: First, information foraging which is narrowing down the data and second the organizing phase where you more start with adding structure, relating the data. But both phases are kind of a continuum, so there is no hard border line between those two. The information foraging grows into the organizing. It's overlapping.

Probably they exist one to each other in the same time.

So what I then try to do, is to put these qualities into the software design of the framework. So we have the two canvases, they shall reflect the bipolarity of sensemaking. The second design feature is the widget-based extensibility which shall reflect the context-sensitivity and also the reciprocity. These two parts can be switched, so you can have another widget here which enables you to structure the data in a graph or something instead of these rings.

You can use the same canvas also but for other bits for which you want to define some other common context.

Yes. And this set of widgets is extensible.

How has a widget to be implemented? How extensible is it?

It has to be implemented in JavaScript and anything you can use in JavaScript, be it HTML or SVG. The communication between these two widgets is going over this framework and has to be implemented by the widget. As for context-sensitivity one can implement a widget which fits the own context, or apply a widget which fits your context. For instance, you could have a map, with the geographical coordinates. It depends on the context of the data. By having this set of widgets extensible and switchable this context-sensitivity should be fulfilled. Reciprocity should also be reflected by these widgets because you ...

You're adding always something from the top widget to the bottom widget or you're moving some bit out of the bottom widget.

Yes, you build up a representation given the bits in the upper one or find new bits given the representations but ...

But you can also manipulate the bits in the lower canvas and change the way they relate each other. So if for instance you move this bit from the circle to another circle.

Yes, you have this representation, and you find - because you have the representation - that this bit doesn't fit here, so you are working top-down. But by switching those widgets, you're also changing the representation in a top-down process. So in this way the reciprocity is also represented. The term representation is quite broad: A circle is a representation and the whole widget itself is also a representation of data. So reciprocity accounts also when you decide - given a representation, ie. a widget - to switch to another widget. As for interactivity the idea is to have everything interactive. Every bit here can be clicked and moved. You can do something with the bits. What you can do depends on the widget itself. For instance on the timeline you cannot move a bit around because it is fixed to its creation timestamp. But the framework enables widgets to make the bits interactive. However, a built-in feature you can drag and drop bits between those two widgets. As for the continuity, the framework is continuously preserving the state of the application in the background. So when you're drawing circles, you might close

the browser and when you re-open it again, the circle will be there and you do not have to save or reload anything.

Ok.

As for the individuality which is due the data that it is an individual process. Here it are the collected learning experiences. So I would say individuality is part of the framework but not as explicit as other features. Reification is also a matter of data, reflected in the data structure, that you can have a representation serve as input for the further sensemaking process. Due the reification, a graph database is used which enables you to build up a relation between two entities arbitrarily. The data is not pre-structured. Anything can be represented in a graph and so a representation, a part of the graph, might serve as the object of another relation in that graph. That's how reification should be fulfilled. So now for the non-functional requirements: This extensibility here demands that the widgets must be independent modules. They can be exchanged at any time. The responsiveness is needed by the interactivity, meaning that the application has to respond immediately to user input. Otherwise the user wouldn't get the experience of interacting with the data.

The user gets easily frustrated.

So something like processing and loading times to server is put into the background and kept away from the user's experience. And for this continuity thing it is important that the application state is serializable at any point. What ever you do it must result in a concise state of the data which can be saved to the server. For this a graph based infrastructure is the most convenient, as you can put anything into a graph. You add another triple. For instance, if the user moves a bit it is saved immediately. But as sensemaking is not confined to moving something, it could be any action. So it has to be serializable at any point. I think then a graph is the easiest way to fulfill that.

Probably. It can come up to a solution, why not. But will there be a graph database in the background?

Yes, on the client-side there is a kind of a graph database. It is not a real graph database. It's a semantic interaction framework which the prototype is built upon. So in the next section I would ask you for these qualities and show you how they are reflected in the code and you can tell me your opinion. So let's look into the code. As for the interactivity, this EntityView is making every bit interactive. Whatever you can do with a bit you can code in this EntityView, or in the subclasses of it, UserEventView, OrgaEntityView.

Ok, drag and drop from one canvas to another, is it also in the EntityView?

It is implemented through the JQuery Draggable plugin. So the EntityView is making this HTML element draggable.

You can extend it, the kind of manipulation, by using other plugins.

Yes, you can use whatever you need to extend the enactivity.

Ok, and you have click events, ok.

And every bit can handle the click-event and by default displays the detail view. And defer is for opening the URI of the bit.

This is then the double-click?

Yes. What I wanted to show here is this draggable [method]. Draggable is just calling the JQuery Draggable.

Ok, you're calling within the view the JQuery Draggable plugin. That's ok.

This makes every entity which extends the EntityView draggable. So for instance this UserEventView extends the EntityView.

Ok.

Of course on the widget-level there can be more interactivity, but this is out of scope of the framework. The WidgetView here has the droppable [method]. Here it sets the JQuery droppable for the organize element. It is kind of the glue between the framework and the organize widget.

Ok.

As for individuality, that's in the data, it is a kind of emergent property of the widgets you use.

Actually the user is using his own data. And you are individually specifying for which user the data is loaded.

Yes. As for the context-sensitivity, the widgets need to be independent so that one can extend the set of widgets. This is actually done by having these modules.

Where do you define that the lower canvas should hold the circle widget?

There is this AppView, which handles the overall structure of the application. It listens for a model of the type "version" being added to the entity collection. Then is calls this filter function in the context of this ...

How do you define which widget to show in the bottom canvas?

Well, switching is not implemented. At the moment it is just possible to have two widgets.

If as a developer I create another widget. How do I add it? How do I define that it should be displayed instead of the circle widget?

In your data structure, on the server side. There you define the data for your new widget and its parameters. And link this widget object to the version where is should be displayed.

Is a version a class where you hard code the widget names?

Yes it's hard coded now. We have this WidgetView which determines which view it creates by the type of the entity, eg. timeline or organize widget type. If you have another widget you would have to add it here. But this organize widget type is an abstract type. You can have another widget whose type inherits from it. So the widget view will check for that and then will call createOrganize. One would need to add another check which kind of organize widget should now be rendered. In this implementation we have only one widget so, I didn't elaborate too much on that. So the createOrganize just creates the Organize view widget which is the circle drawing widget. Perhaps here in the part one would make a more educated selection which view should be rendered.

I was only concerned if it is possible. But in the code it is possible. In this initial prototype that's ok.

Ok, as for the continuity which said that everything is serializable at any point.

For this you call the services and the data modules have to implement this saving.

Do you force the modules? What happens if you do not implement it?

It does not enforce anything. It might be that you don't want to save anything to the server. It might be that you want to keep it on the client side. But that would harm the continuity feature. If the user stops sensemaking he might lose the state. Continuity is more implemented in the architecture of the whole. That we have the separation between the data layer and the service layer. The service layer wraps the asynchronous communication. The developer does not have to hassle with the asynchronous stuff, he just saves the model, saves the data to the service layer and the service layer handles that. And as soon as the entity is saved and updated by the server the data module is informed. By having this kind of graph based structure developers also can easily store stuff there because it is key-value based. This also accounts for the serializability of the data. Of course a service has to be implemented anyway. The data cannot be saved magically somehow. So do you think that continuity is represented in this?

My answer would be yes. But could you just show me the code where you save the state?

The service layer is organized so that when you save something you put it into the save method which wraps all the save services. Here it checks what kind of entity it is and then looks for the right Social Semantic Service to call.

[reading code] Ok that's actually clear. And the continuity is pretty clear.

Then it executes what is defined in "success". We have this method chaining here. And this is the jQuery Deferred Pattern. And here in the fetchWidgets method we get an array of widgets and these widgets are added to the base collection of entities.

Ok.

Ok, now for the reciprocity.

You got it covered I think with drag and dropping? So you can drag and drop a bit in the lower canvas?

Perhaps, yes, I would also say that drag and dropping accounts for the reciprocity. Of course that switching between widgets would also be needed. it's not implemented but ...

But still the part of code where you showed me the organize view is created. If we extend it, put there another view, that would also support reciprocity.

Yes, that would enable it. So what would be needed is that switching features, that you can have several widgets there. But I think it would not be too hard to implement it into this framework as you just have to extend this AppView and the WidgetView part where they decide which widget to show.

Mhm.

The reification is having representation serve as the input to the sensemaking which I thought is represented by the graph database. Everything is an entity and so for instance the timeline widget also has a representation as an entity in the base collection.

Wouldn't it also be a feature that you are sharing your own results of the sensemaking process to other people? Isn't that also reification?

Well if the other people work upon that results. If they treat this outcome as a whole. I could also have these episodes - each episode has versions and widgets - but the episode could also serve as an entity. I don't know whether it would make sense to have for instance episode bits represented in the timeline. That you can make sense upon your episodes or upon your versions.

Then are continuity and reification related? So you actually have some outcome of the sensemaking process when you stopped. And at a later point of time you started again and work on it, so the previous outcome is now serving as a input afterwards.

Yes, surely. But the reification is on a structural data level and the continuity is more on a time-based level. Both rely on the data being serializable.

Ok.

Of course one would have to implement a widget for that which can represent these outcomes, and you can deal with these outcomes. But this should be possible in your opinion?

I think yes. I mean you could export it, the data structure of an outcome. It's actually stored in Social Semantic Server and you fetch it again. Or in the case when you have several widgets, let's say you have a circle and graph widget, that's actually in the same data on the Social Semantic Server, so you're manipulating about the same data. Is such a scenario possible?

Yes, that's possible, it would mean for instance you have the circle widget and then I take the data and represent in a graph widget where instead of circles I have links between stuff which was in the same circle. But this is not what is meant by reification. It is more that you are reusing the structure found, working on that structure, as a kind of a higher level of data then.

Think with the saving and loading functionality you got it covered with that.

What do you think about that everything is an entity?

I'm a little bit concerned about it. It is ok at one point. You get more flexibility maybe, that's ok.

As soon as an object becomes complex, that is that is has more than one value, then it should become an entity. A tag for instance has a user relation and a timestamp. Then it should be an entity.

What are the pros for making a tag an entity? Do you need it in the framework to be an entity?

That's application specific, whether I need a tag to be an entity. From the framework perspective everything possible should be an entity.

Does it really make sense to make the whole interface of entities?

It depends, what you would like to do. This sidebar itself has not much data in it. So there should not be an entity for the sidebar. Not everything needs to be an entity of course, only things which have data in it. For instance the OrganizeWidget is an entity because it has got this "hasCircles", "hasEntities". Somewhere there must be stored which circles and which entities are there in this widget. It is a kind of storage space for the widget data.

Maybe we say it like that. Things that need their state to be preserved, should be entities.

Ok, bipolarity. Do you find bipolarity represented?

I have to recall ... information foraging. You have the timeline.

Just for searching bits, narrowing down.

Ah, you're remembering it's somewhere in between here, so you go there.

For instance.

384 Ok, that's covered.

385 the user can find a bit. So that's true, you got information foraging. And you can really
386 organize the bits, place them in circles, so I think it's also covered. And both canvases can
387 be used in the application at the same time. Yeah, I think that's ok.

388 **Ok. So I think we are gone through everything. Thank you for your time.**

389 No problem.

Bibliography

Cheng, Wen-Huang and David Gotz (2008). "Context-based page unit recommendation for web-based sensemaking tasks." In: *Proceedingsc of the 13th international conference on Intelligent user interfaces - IUI '09*. New York, New York, USA: ACM Press, p. 107. ISBN: 9781605581682. DOI: 10.1145/1502650.1502668. URL: http://dl.acm.org/citation.cfm?id=1502650.1502668 (cit. on pp. 20, 22).

Dennerlein, Sebastian et al. (2014). "Making Sense of Bits and Pieces: A Sensemaking Tool for Informal Workplace Learning." English. In: *Open Learning and Teaching in Educational Communities*. Ed. by Christoph Rensing et al. Vol. 8719. Lecture Notes in Computer Science. Springer International Publishing, pp. 391–397. ISBN: 978-3-319-11199-5. DOI: 10.1007/978-3-319-11200-8_31. URL: http://dx.doi.org/10.1007/978-3-319-11200-8_31 (cit. on p. 85).

Dervin, Brenda (1998). "Sense-making theory and practice: an overview of user interests in knowledge seeking and use." In: *Journal of knowledge management* 2.2, pp. 36–46 (cit. on pp. 6, 8, 16, 25, 34, 87).

Dervin, Brenda (1999). "Chaos, order and sense-making: A proposed theory for information design." In: *Information design*, pp. 35–57 (cit. on pp. 7, 15–17).

Fayyad, Usama M. (1996). "Data mining and knowledge discovery: making sense out of data." In: *IEEE Expert* 11.5, pp. 20–25 (cit. on pp. 2, 5).

Fischer, Gerhard et al. (1994). "Seeding, Evolutionary Growth and Reseeding: Supporting the Incremental Development of Design Environments." In: *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*. CHI '94. Boston, Massachusetts, USA: ACM, pp. 292–298. ISBN: 0-89791-650-6. DOI: 10.1145/191666.191770. URL: http://doi.acm.org/10.1145/191666.191770 (cit. on pp. 2, 5).

Gotz, David (2007). "The ScratchPad." In: *Proceedings of the 16th international conference on World Wide Web - WWW '07*. New York, New York, USA: ACM Press, p. 1329. ISBN: 9781595936547. DOI: 10.1145/1242572.1242834. URL: http://dl.acm.org/citation.cfm?id=1242572.1242834 (cit. on pp. 19–22, 33, 98).

Grünwald, Szabolcs and Henri Bergius (2012). "Decoupling Content Management." In: *developer track, WWW2012 Conference, Lyon* (cit. on p. 45).

Horng, Duen et al. (2011). "Apolo: Making Sense of Large Network Data by Combining Rich User Interaction and Machine Learning." In: *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, pp. 167–176. ISBN: 9781450302678 (cit. on pp. 19, 29, 31, 33, 98).

Klein, Gary et al. (2007). "A data-frame theory of sensemaking." In: *Expertise out of context: Proceedings of the sixth international conference on naturalistic decision making*. Mahwah, NJ: Lawrence Erlbaum Associates, pp. 15–17 (cit. on pp. 5, 14, 16).

Kowald, Dominik et al. (2013). "The Social Semantic Server." In: *I-SEMANTICS (Posters & Demos)*. Ed. by Steffen Lohmann, pp. 50–54 (cit. on pp. 63, 99).

Lanthaler, Markus (2013). "Creating 3rd Generation Web APIs with JSON-LD and Hydra." In: *Proceedings of the 22nd International World Wide Web Conference, Rio de Janeiro*. ACM Press, pp. 35–37 (cit. on pp. 47, 101).

Mayring, Philipp (2000). "Qualitative content analysis." In: *Forum: Qualitative social research* 1.2 (cit. on p. 86).

Morris, Meredith Ringel, Jarrod Lombardo, and Daniel Wigdor (2010). "WeSearch : Supporting Collaborative Search and Sensemaking on a Tabletop Display." In: *Proceedings of the 2010 ACM conference on Computer supported cooperative work*. ACM, pp. 401–410. ISBN: 9781605587950 (cit. on p. 25).

Paul, Sharoda A. and Meredith Ringel Morris (2009). "CoSense." In: *Proceedings of the 27th international conference on Human factors in computing systems - CHI 09*. New York, New York, USA: ACM Press, p. 1771. ISBN: 9781605582467. DOI: 10.1145/1518701.1518974. URL: http://dl.acm.org/citation.cfm?id=1518701.1518974 (cit. on pp. 19, 24–27, 32, 98).

Pirolli, Peter and Stuart Card (1999). "Information foraging." In: *Psychological review* 106.4, p. 643 (cit. on p. 12).

Pirolli, Peter and Stuart Card (2005). "The sensemaking process and leverage points for analyst technology as identified through cognitive task analysis." In: *Proceedings of International Conference on Intelligence Analysis*. Vol. 5. Mitre McLean, VA, pp. 2–4 (cit. on pp. 5, 9, 12–17, 97).

Reinhardt, Wolfgang, Matthias Moi, and Tobias Varlemann (2009). "Artefact-Actor-Networks as tie between social networks and artefact networks." In: *Collaborative Computing: Networking, Applications and Worksharing, 2009. CollaborateCom 2009. 5th International Conference on*. IEEE, pp. 1–10 (cit. on p. 63).

Russell, Daniel M. et al. (1993). "The cost structure of sensemaking." In: *Proceedings of the SIGCHI conference on Human factors in computing systems*, pp. 269–276. ISBN: 0897915755. DOI: 10.1145/169059.169209 (cit. on pp. 5, 9–12, 14–17, 33, 39, 97).

Ryder, Brendan and Terry Anderson (2009). "'Coalesce'." In: *Proceedings of the 21st Annual Conference of the Australian Computer-Human Interaction Special Interest Group on Design: Open 24/7 - OZCHI '09*. New York, New York, USA: ACM Press, p. 289. ISBN: 9781605588544. DOI: 10.1145/1738826.1738877. URL: http://dl.acm.org/citation.cfm?id=1738826.1738877 (cit. on pp. 19, 22–24, 33, 98).

The Reactive Manifesto (2014). URL: http://www.reactivemanifesto.org/ (visited on 11/22/2014) (cit. on pp. 41, 42).

Tomberg, Vladimir et al. (2013). "A Sensemaking Interface for Doctors' Learning at Work: A Co-Design Study Using a Paper Prototype." In: *ECTEL meets ECSCW 2013: Workshop on Collaborative Technologies for Working and Learning*, pp. 54–58 (cit. on pp. 35, 85).

Weick, Karl E (1995). *Sensemaking in organizations*. Vol. 3. Sage (cit. on pp. 5, 15–17, 100).

Westenthaler, Rupert and Olivier Grisel (2012). "Automated linking data with Apache Stanbol." In: *developer track, WWW2012 Conference, Lyon* (cit. on p. 45).