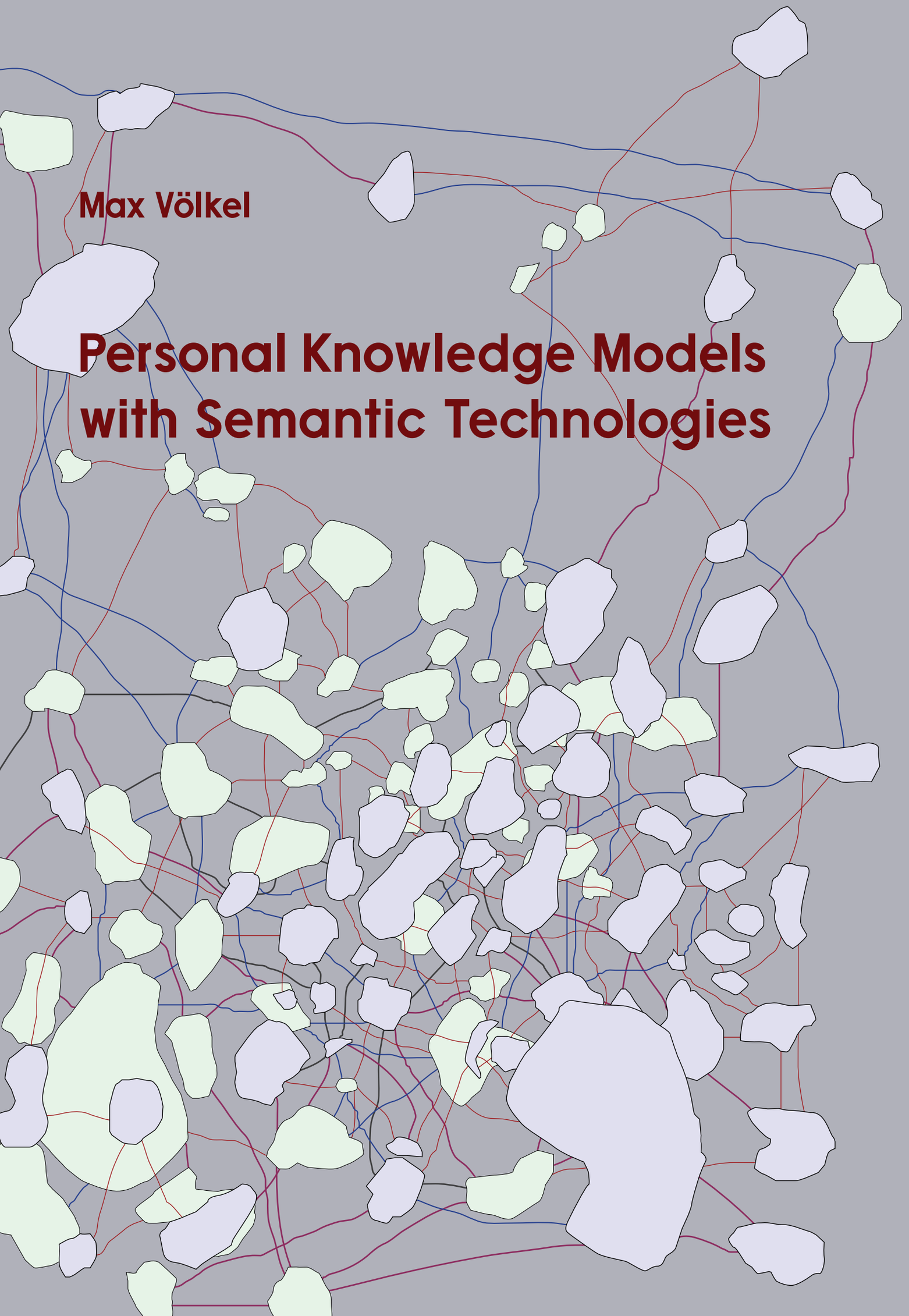
Max Völkel

Personal Knowledge Models with Semantic Technologies



Personal Knowledge Models with Semantic Technologies

Max Völkel

Bibliografische Information

Detaillierte bibliografische Daten sind im Internet über
http://pkm.xam.de abrufbar.

Covergestaltung: Stefanie Miller

Zur Erlangung des akademischen Grades eines Doktors der Wirtschaftswissenschaften (Dr. rer. pol.) von der Fakultät für Wirtschaftswissenschaften des Karlsruher Instituts für Technologie (KIT) genehmigte Dissertation von Dipl.-Inform. Max Völkel.

Tag der mündlichen Prüfung:	14. Juli 2010
Referent:	Prof. Dr. Rudi Studer
Koreferent:	Prof. Dr. Klaus Tochtermann
Prüfer:	Prof. Dr. Gerhard Satzger
Vorsitzende der Prüfungskommission:	Prof. Dr. Christine Harbring

Abstract

Following the ideas of Vannevar Bush (1945) and Douglas Engelbart (1963), this thesis explores how computers can help humans to be more intelligent. More precisely, the idea is to reduce limitations of cognitive processes with the help of *knowledge cues*, which are external reminders about previously experienced internal knowledge. A knowledge cue is any kind of symbol, pattern or artefact, created with the intent to be used by its creator, to re-evoke a previously experienced mental state, when used. The main processes in creating, managing and using knowledge cues are analysed. Based on the resulting *knowledge cue life-cycle*, an economic analysis of costs and benefits in Personal Knowledge Management (PKM) processes is performed.

The main result of this thesis is a meta-model for representing knowledge cues, which is called *Conceptual Data Structures* (CDS). It consists of three parts: (1) A simple, expressive data-model; (2) A small relation ontology which unifies the relations found in the cognitive models of tools used for PKM tasks, e. g., documents, mind-maps, hypertext, or semantic wikis. (3) An interchange format for structured text together with corresponding wiki syntax.

These three parts together allow representing knowledge cues in varying degrees of granularity (number and size of items), structuredness (relations between items), and formality (fraction of items typed with items from a meta-model) in a unified way.

The CDS model has been implemented in Java. Based on this reference implementation several tools for personal knowledge management have been created (one by the author of this work and two external tools). All three tools have been used in a comparative evaluation with 125 person-hours. In the evaluation, the Conceptual Data Structures (CDS) data model has successfully been used to represent and use (retrieve) artefacts in a uniform fashion that are in different degrees of formalisation. Although still research prototypes, the CDS Tools had interaction efficiency and usability ratings compared to Semantic MediaWiki (SMW). Using CDS Tools, users produced significantly more non-trivial triples than with SMW. The created Relation and concept hierarchies can be re-used in a semantic desktop.

Acknowledgements

I thank my professor, Prof. Dr. Rudi Studer, for letting me pursue what I wanted to pursue, taking care of my environment. I thank the whole research group which has a fascinating attitude of open criticism and honest feedback. Of course, I also thank Prof. Dr. Klaus Tochtermann for his valuable feedback. I thank Dr. Andreas Abecker for a prompt and valuable review of the complete thesis.

I thank (in order of appearance) Alexander Grossoul, Mike Sibler, Sebastian Gerke, Benjamin Heitmann, Markus Göbel, Andreas Kurz, Sebastian Döweling, and Mustafa Yilmaz who helped to research and develop various parts of this work with their student research projects (*Studienarbeiten*) and diploma thesis' (*Diplomarbeiten*). I thank additionally (in order of appearance) the students that helped to develop the CDS code base, especially Werner Thiemann, Daniel Clemente, Konrad Völkel, Andreas Kreidler, Björn Kaidel, and Daniel Scharrer. Furthermore, I thank the students that helped to evaluate the final CDS tools.

I would also like to thank a lot of people for reviewing this thesis in various stages. All mistakes that are left are my fault, not theirs. I thank Jens Wissmann and Peter Wolf for reviewing chapter 4. I thank Konrad Völkel for reviewing chapters 1, 3 and 4. I thank Mark Hefke for providing valuable feedback on the economic analysis. I thank David Elsweiler for language and style checking various parts of the thesis.

I thank Heiko Haller for many enthusiastic day-long discussions about the obscurest details of the CDS model. His constant nagging questions helped to shape the CDS model and API.

I thank Stephan Bloehdorn and Denny Vrandecic for many critical discussions at our favourite Sushi place.

And I thank Anke and my parents for being patient with me in the long times where I was sitting at my desk.

Part of this work has been funded by the European Commission in the context of the IST Integrated Project NEPOMUK[1] – The Social Semantic Desktop, FP6-027705. Another part of this work has been done in "WAVES - Wissensaustausch bei der verteilten Entwicklung von Software"[2], funded by BMBF, Germany. Yet another part of this work was supported by the European Commission under contract FP6-507482 in the project Knowledge Web[3].

The expressed content is the view of the author but not necessarily the view of any sponsor.

[1] `http://nepomuk.semanticdesktop.org/` (accessed 06.01.2010)
[2] `http://waves.fzi.de/` (accessed 06.01.2010)
[3] `http://knowledgeweb.semanticweb.org` (accessed 06.01.2010)

Presumably man's spirit should be elevated if he can better review his shady past and analyze more completely and objectively his present problems. He has built a civilization so complex that he needs to mechanize his records more fully if he is to push his experiment to its logical conclusion and not merely become bogged down part way there by overtaxing his limited memory.

Vannevar Bush (1945, p. 108)

Contents

1. Introduction

This chapter motivates this work and provides an overview of the work presented.

1.1. Readers guide

Following the ideas of Douglas Engelbart (1963), this thesis explores how computers can help humans to be more intelligent. More precisely, the idea is to reduce limitations of cognitive processes with the help of external reminders about internal knowledge. That is, the whole process of thinking, taking notes, searching notes, re-using knowledge encoded in personal notes, should become more productive. Users should be supported – but not forced to – to structure, organize and formalise their notes, so that better retrieval options become possible.

1. In the first chapter, the motivation for this thesis and a definition of personal knowledge management (PKM), relating to knowledge management (KM) are given. PKM is characterised as a decision where the user has to weigh the costs against the benefits, i. e.: *Will my effort to record and structure knowledge be rewarded in the future with enough benefit?* This chapter introduces the key terms "*knowledge cue*", and "*knowledge model*" and explains how these artefacts can vary in size, structuredness and degree of formality. The chapter concludes by summarising the solution idea (page 32) and stating the research questions (page 30).

2. The foundations of this thesis are laid in Chapter 2 (page 35). Longer sections highlight relevant concepts and references from *models* in a general sense, and from *documents*. Both have been major conceptual influences for this thesis.

   On a more technical side, file systems, hypertext, software engineering, semantic web, wikis, mind- and concept mapping, and tagging are described. These more technical topics are analysed in depth in Chapter 3.

   A fusion of wiki and semantic web works was the starting point of this work. The decoupling from wiki-content and wiki-user interface has been described already by Völkel (2005a). Later these ideas led the author to start a workshop series on semantic wikis, create the first article on *Semantic wiki* in Wikipedia[1], and then to this thesis.

[1] `http://en.wikipedia.org/wiki/Semantic_wiki` (accessed 06.01.2010)

A relation to existing approaches can be found in the fields of note-taking, personal information management, and knowledge articulation, which are also introduced.

3. Chapter 3 looks into use-cases in PKM, goes over to existing process models of which none describes PKM satisfactory and introduces a new knowledge cue life-cycle model. This model is used to analyse in depth economic factors in PKM from which first requirements for PKM representation models as well as for PKM tools are derived.

   Literature analysis reveals further requirements.

   The existing conceptual models of popular tools from the fields introduced in Chapter 2 are analysed with the intention to find commonalities among them and gather structural features that should be present in future PKM tools simply because people have used them successfully in the past, which also means, they don't have to learn new concepts.

   A similar analysis is performed for commonly used relations in tools that are often used for doing PKM tasks.

   The chapter concludes with a concise requirements table.

4. In Chapter 4 the solution is presented. The solution consists of a simple data model for storing knowledge cues. This model represents content of varying granularity as well as semantic links between them.

   This data model is accompanied with a relation ontology that contains the most-commonly used relations from popular tools used for doing PKM tasks. It derives and explains a hierarchy of relation types, suitable to describe personal knowledge models.

   A *Structured Text Interchange Format* (STIF) is presented as well as ways to transform instances of this model into CDS and vice versa. STIF is a building block in the gradual formalisation of content.

5. Chapter 5 presents different prototypical tools based on the CDS model. The *Hypertext Knowledge Workbench* (HKW) is an editor for personal, CDS-based knowledge models. The editor is based on a generic CDS-API, on which several other tools have been realised by third parties. Those CDS-based tools are also presented.

6. Next, different kinds of evaluation are presented in Chapter 6. A part of the evaluation is the rational analysis of fulfilled criteria. Furthermore, five participants worked together for 125 hours with CDS Tools and a comparable semantic modelling tool. Their resulting knowledge models have been assessed and compared in various ways.

   The chapter continues by presenting a number of approaches related to the different parts of the solution.

7. In Chapter 7, a number of future research themes are identified and presented. Finally, the main contributions of this thesis are summarized.

8. The appendices present the complete set-up of the evaluation as well as detailed result tables.

Notational conventions

- This thesis uses the female form “she” for examples, but male readers should not feel excluded.
- The margin notes allow the selective reader to quicker find relevant parts of the text. Some margin notes also give a short summary of the content. The sequential reader can ignore the margin notes without missing any content.
- Each requirement in this thesis has a number and a short reference-name. Example: “Req. 8 formality levels”
- The different types of ITEMS in the CDS model are typeset in small capitals.

1.2. Motivation

Our world is constantly changing and the rate of change has constantly increased. Today, changes are in large part caused by the humans themselves, due to the growth of their global population and the ability to use technology to change matter, i. e., in agriculture, energy production, goods production, and transportation. Part of the ability to steer the forces of nature and humans in controlled ways can be attributed to the invention of management: "The most important contribution of management in the 20th century was to increase manual worker productivity fifty-fold (Drucker, 1999)."

Great Problems

The fast rate of change in the environment and in human societies causes great wealth, but also great problems, e. g., social problems, economic problems and ecological problems (Vester, 2000). As an example, very few people had foreseen or understood the financial crisis in 2008. Many bankers said in interviews that nobody could understand the complex relationships of markets and financial products any longer.

Humans must tackle the pressing ecological and economical problems. Bush (1945, p. 108) summarised the motivation for this thesis nicely:

> "Presumably man's spirit should be elevated if he can better review his shady past and analyze more completely and objectively his present problems. He has built a civilization so complex that he needs to mechanize his records more fully if he is to push his experiment to its logical conclusion and not merely become bogged down part way there by overtaxing his limited memory."

Drucker (1999) puts it more optimistically and foresees: "The most important contribution of management in the 21st century will be to increase knowledge worker productivity – hopefully by the same percentage. [...] The methods, however, are totally different from those that increased the productivity of manual workers."

What could be methods to increase the productivity of knowledge workers? This translates to efficiency and effectiveness, i. e., also to be able to solve large or hard problems that could not yet be solved at all.

Towards a knowledge-based economy

Due to the high degree of specialisation in our society, efficient knowledge organisation and sharing has become a critical success factor. In 2000, the European Commission issued the *Lisbon Strategy*[2] to stimulate economic growth. The first one of three pillars is:

> ... preparing the ground for the transition to a competitive, dynamic, knowledge-based economy. Emphasis is placed on the need to adapt constantly to changes in the information society and to boost research and development.

[2] `http://lisbon.cor.europa.eu/` (accessed 06.01.2010)

Need for more automation of knowledge work

As our society becomes more knowledge-intensive, future progress depends on efficient and effective ways to automate knowledge processing. Full automation is an unrealistic goal, but partial automation is feasible and necessary. This is analogous to manufacturing, where repetitive or physically hard tasks have been replaced by machines and robots.

1.2.1. Focus on the individual knowledge worker

This section introduces the terms *knowledge*, *knowledge management* and finally *personal knowledge management* is defined.

Knowledge North (2002) defines the terms signal, data, information, knowledge, and wisdom in a layered fashion, one building upon another. By this definition, knowledge itself cannot be stored in information systems, only information can. Maurer (1999, p. 12) states that knowledge resides in the heads of people and the computer can only store "computerized knowledge" which is to be understood as "shadow knowledge", a "weakish image" of the real knowledge. Knowledge cannot be *stored* in an information system. In fact, an information system, such as a computer, can only store data, i. e., bits. Together with data about this data (metadata) a program can interpret the data. In this respect, some of the data becomes information for the computer. The Handbook on Knowledge Management (Holsapple, 2004) makes a clear distinction between a representation and knowledge being represented by it. Knowledge is then defined as the usability of the representation to a human interpreter in a given environment, trying to accomplish a certain task. In this view, a computer can store only information, not knowledge itself. But it can certainly *represent* knowledge as information.

Blackler (1995) sees knowledge *embrained* (conceptual, implicit), *embodied* (tacit, implicit), *encultured* (shared beliefs), *embedded* (in processes) or *encoded* (symbolic, external). So he distinguished between conceptual knowledge that has just not been written down yet and embodied knowledge that cannot be written down. This thesis deals primarily with conversions between embrained and encoded knowledge.

Knowledge management The field of *knowledge management* has investigated since about 1995 (Stankosky, 2005) how people and knowledge work together. North (2007) defines *knowledge work* as work based on knowledge with an immaterial result; value creation is based on processing, generating and communicating knowledge.

Polanyi (1966) makes a distinction between *explicit knowledge* encoded in artefacts such as speech, books or web pages, and *tacit knowledge* which resides in the individual. The SECI-model of Nonaka (1994, p. 57 ff) describes knowledge conversions between tacit and explicit knowledge.

Nonaka's model has four integrated processes:

- *Socialisation* describes a direct transfer of tacit knowledge between humans.
- *Externalisation* is the process from tacit to explicit knowledge. Other researchers call this process *codification* or *materialisation*. This step must happen before any piece of knowledge can be externally stored or even processed by a computer.
- *Combination* or re-combination creates new explicit knowledge out of existing explicit knowledge.
- *Internalisation* is the process transferring knowledge from explicit representations to a humans mind, e. g., when reading a document.

Abecker (2004, p. 26) defines knowledge management as a

- structured, holistic approach
- for sustainable improvement of handling tacit and explicit knowledge (e. g., know-how, skills, notes, documentation) in an organization
- on all levels (individual, group, organization, interorganizational level)
- in order to better achieve one or more of the organization's strategic goals, like decreasing costs, improving quality, or fostering innovation.

Top-down

Knowledge management approaches have mostly focused on sharing of knowledge within organisations and teams. Many of the initial approaches tried to improve knowledge sharing in a top-down manner, by installing central repositories for explicit, codified knowledge. Such a single central database is often either not used by employees or not filled with easy-to-digest content. Overall, the high expectations towards centralised approaches have often not been met (Braganza and Mollenkramer, 2002). Other approaches concentrated on tools such as expert finders and corporate yellow pages, hence less on managing explicit knowledge and more on direct communication among people (socialisation).

Knowledge worker

Personal knowledge management (PKM) In 1958, Peter F. Drucker (1985) was among the first to use the term *knowledge worker* for someone who works primarily with information or one who develops and uses knowledge in the workplace. Schütt (2003) defines a *knowledge worker* based on the works of Drucker (1977) and Taylor (1911): Simplified, workers (doing) are instructed by managers (thinking). These managers have to manage themselves. This self-managing is considered an important characteristic of a knowledge worker.

In the current Western societies, there is a trend from once rather static organisations to virtual organisations with short life-spans. The individual has to change from a loyal employee to a more self-caring, life-long learning, self-marketing entrepreneur.

Management is in general a systematic approach to define goals, measure, define and execute actions and repeat this control loop until the goal is reached. Self-managing involves the problem of fulfilling two roles (executing and managing) and learning when and how to switch between these roles. Typical management problems in PKM are, e. g., time and task management (Grebner, 2009), matching work habits with personal productivity level variations, investing time into personal learning and PKM improvements, and general work-life balance. As an example, Groth (2006) observes that often private homepages of researchers get updated more often than their official homepages. Personal knowledge management focuses on the productivity of individual knowledge workers. Davenport (2005, p. 6) lists members of these professions as knowledge workers: Management; business and financial operations; computer and mathematical; architecture and engineering; life, physical, and social sciences; legal; healthcare practitioners; community and social services; education, training and library; and arts, design, entertainment, sports, media.

Focus on the individual

Knowledge is fundamentally created by individuals (Nonaka and Takeuchi, 1995, p. 59). This thesis favours a bottom-up approach. By supporting the individual in his PKM, overall Knowledge Management (KM) of the organisations where the individual is part of is improved. "The knowledge-based organisation is no more effective than the sum of its knowledge workers' effectiveness." (Davenport in Higgison, 2004).

Increasing the individual knowledge worker productivity increases the productivity of the organisation as a whole. One should focus on the individual and give individual users incentive and benefit before focusing on the social network (Oren, 2006, p.8). Management of personal information and personal knowledge becomes more important as more people work as knowledge workers, where their capital is their knowledge.

Emergence of PKM

The term *personal knowledge* has been discussed in great length by Polanyi (1958), reprinted in (Polanyi, 1998), in rather philosophical ways. Usage of the term can be dated back to at least 1987 "computer-based personal knowledge management systems can combine and integrate a number of important resources..." (National Library of Medicine (U.S.), Board of Regents, 1987, p.40)[3]. Following publications from the hypertext (Schnase, Leggett, Hicks, Nürnberg, and Sánchez, 1993) and the database community (Srinivasan and Zeleznikow, 1990) retained the technical focus. Later publications started to use the term in a broader fashion, e. g., in the context of life-long learning (Frand and Hixon, 1999; Alley, 1999). Soon thereafter the first universities started to offer courses on PKM, in which they taught students, e. g., how to structure information and create knowledge maps (Schreiber and Harbo, 2004).

Definition: PKM

The European Committee for Standardization defines PKM as "A set of concepts, disciplines and tools for organizing often previously unstructured knowledge, to help individuals take responsibility for what they know and who they know" (Allan, Heisig, Iske, Kelleher, Mekhilef, Oertel, Olesen, and (ES), 2004, part 5, p. 12). Higgison (2004) defines personal knowledge

[3]Search via ACM Digital Library, Google Scholar, and Google Booksearch

management as *managing and supporting personal knowledge and information so that it is accessible, meaningful and valuable to the individual; maintaining networks, contacts and communities; making life easier and more enjoyable; and exploiting personal capital.* This thesis defines PKM as follows:

Definition

PKM investigates the use of methods and tools to amplify the abilities of the individual to work better with knowledge. This can mean to

- help recalling previously learned knowledge faster (or at all) when it is required, e. g., with a desktop search engine;
- represent (not store) knowledge in a tool to derive new insights – as it is done today often with spreadsheet applications[4]; or
- strategies for filing ideas to retrieve them when needed.

Comparing OKM and PKM In the remainder of this thesis, the term *organisational knowledge management* (OKM) is used to distinguish it from *personal knowledge management*. Although *knowledge management* (KM) could be used as a super-term for OKM and PKM, most works on KM talk in fact mostly about OKM. OKM is contrasted to PKM by Allan et al. (2004): "Unlike personal KM, which centres on the individual, organizational KM depends upon an enterprise-wide strategic decision to actively manage knowledge through a range of processes, tools and people."

Davenport (2005, p. 138) notes a need for PKM "Perhaps the single most important, yet rarely addressed, knowledge worker capability is the management of the personal information and knowledge environment".

Table 1.1 compares organisational and personal knowledge management.

Perspective of the individual

PKM takes the perspective of the individual. This perspective is potentially better suited to explain individual motivations and behaviour, even in organisational contexts.

Top-down vs. bottom-up

This thesis favours a bottom-up approach. The individual is supported in his personal knowledge management. This indirectly improves the overall knowledge management of the organisation where the individual is part of.

Referring to the SECI-model (Nonaka, 1994), PKM is restricted to the processes externalisation, re-combination, and internalisation – socialisation with oneself is not possible. Thus PKM is "communication with oneself". A more detailed PKM process model is presented in Sec. 3.2.

PKM tries to support individuals in effectively storing, structuring, linking, formalising and retrieving knowledge cues. Thus there is the danger that knowledge islands are created. That might indeed be the case, but

[4] Given some numbers a formulas, it is easy to calculate simulations with n steps or get quickly the results of many calculations, which would be tedious to enter in a calculator. Even statistical analysis or visualisation tools are usually available in a spreadsheet tool.

Typical attributes	OKM	PKM
Perspective	The organisation	The individual
Approach	Top-down	Bottom-up
SECI	Socialisation	Externalisation, internalisation
Degree of structuredness	Structured (“publication”)	Semistructured (“personal note”)
Degree of formality	Semiformal (“forms”)	Informal (“plain text”)
Number of notes per person	Few	Many
Education	Mature	Poor

Table 1.1.: Comparing Organisational and Personal Knowledge Management

these knowledge islands can develop into rich, interlinked knowledge models, which can be shared with others. The result is a knowledge distribution process not in a centralised, but rather a peer-to-peer fashion, thus much closer to existing social processes.

Focus on the individual

According to Gartner, knowledge management is shifting from a focus on enterprise productivity to a focus on individual knowledge worker productivity (Caldwell, 2002). This thesis focuses on the *individual* knowledge worker and ways to make her or him more productive. Top-down approaches emphasize sharing, however, if knowledge is not externalised, there is nothing to share. Bottom-up approaches empower a knowledge worker with tools for externalisation and internalisation (e. g., desktop search and enterprise search).

Degree of formality

Most traditional enterprise KM products have no or poor support for managing truly private and personal notes. The very idea of enterprise knowledge management is usually the sharing of content within teams or even the whole organisation. Psychologically, this turns each entry of the user into a semi-official publication. The author has to carefully judge the social impacts on her professional reputation etc. The net effect for most personality types are few written notes. Those notes that are published in an enterprise KM system will be rather well-structured and well-written. In PKM, a typical user will write many more notes, many of them probably using just the amount of structure needed to be understandable by the author. This notion of a “degree of formality” is explained in depth in Sec. 1.2.4. A tool which supports management of early, rough, unshare-able notes and structures helps to create shareable knowledge as well. Even if only some of them will be shared with others, such emerging digital artefacts might not exist at all if they could not been drafted quickly and informally in the beginning.

Education

In OKM there exist a large number of specialist training courses on best practices, but for PKM there are few such offers. The market consists mostly of courses on time management and “How To”-books. Individuals

thus have to find out everything for themselves (Mitchell, 2005). Toffoli (2002) stresses the importance of soft factors such as the need for education in personal knowledge management which, in his view, includes the ability to write programs for knowledge automation, transformation and filter tasks.

Personal knowledge management and the bottom-up approach have been less explored than organisational knowledge management as a means to improve productivity.

Summary

This section introduced *personal* knowledge management (PKM) as opposed to organisational knowledge management (OKM). PKM can be seen as a sub-field of general KM which looks with the *perspective of the individual* on knowledge management problems. This section motivated why this thesis focuses on the *individual* knowledge worker.

1.2.2. Limits of the individual

This section describes the role of external representations in problem solving and relevant cognitive limits of the individual.

Problem solving The goal of this thesis is to make knowledge workers more productive. In particular, to enable them become better at problem solving, in order to be able to tackle major social, economic and ecological issues. Problem solving has been defined as "any goal-directed sequence of cognitive operations" (Anderson, 1980, p. 257).

According to Jonassen (2000, p. 10) problem solving varies along at least three different dimensions:

- (a) problem type,
- (b) problem representation, and
- (c) individual differences.

An analysis of problem types is given by Dörner (2003). *Complicated* problems have predictable, but difficult to calculate relations. *Complex* problems are those with unpredictable parts. *Wicked* problems (Rittel and Webber, 1973) are those with incomplete, contradictory, and changing requirements, as well as feedback loops. Vester (2000) presents a methodology to tackle complex problems by modelling the feedback loops of the problem domain. For a given knowledge worker and given problem, the only way to make the individual more productive is by working on the problem representation.

"Solving a problem simply means representing it so as to make the solution transparent" (Simon, 1981, p. 303). Zhang and Norman (1994, p. 2) defines the *representational effect* as "the phenomenon that different isomorphic representations of a common formal structure can cause dramatically different cognitive behaviours."

To summarise, internal and external representations heavily influence problem solving performance.

Cognitive limits In knowledge work, people are frequently confronted with two limitations of the human mind: long-term memory recall and short-term memory capacity. Both limits degrade performance on tasks such as learning, understanding, or sense-making. Other limitations such as missing creativity or insufficient ability to structure have also a huge influence on the ability to solve problems, learn and understand, but these limits are not addressed in this thesis.

Long-term memory recall

Long-running tasks require a knowledge worker to remember facts over a long period of time. As recall degrades over time, most knowledge workers take notes to remember themselves later. Today, knowledge workers are flooded with information (Alvarado, Teevan, Ackerman, and Karger, 2003). Often the purpose of unexpectedly encountered facts is unclear, so they are filed for later use.

A typical example of a long-running task is research on a given domain, e. g., for creating a presentation or publication. A researcher will encounter new facts at unexpected times, e. g., when working on another task.

External representations can provide memory aids (Zhang and Norman, 1994, p.32); hence limits of the long-term memory can be overcome partially with tools to help remembering or reconstructing knowledge.

Short-term memory limits

Human short-term memory can hold only around seven objects at a time (Miller, 1956). Later works in psychology coined the term "*cognitive load*" (Sweller, 1988) as a concept to measure how much short-term memory capacity is required for certain tasks. Card in (Jones, Pirolli, Card, Fidel, Gershon, Morville, Nardi, and Russell, 2006, p. 2) even claims colloquially "Most significant cognition is too complicated to fit in the head", which means one must use external representations.

The importance of cognitive load is widely acknowledged. Cognitive load can be influenced, e. g: (1) The ISO standard 10-075 series (ISO, 1991a) gives general advice how to handle *mental workload*. (2) For user interfaces, Shneiderman (1998, p. 75) advises to "Do everything possible to free the user's memory burden".

Large or complex problems can often not be fully represented internally, thus external knowledge representations must help to "swap" concepts. The limitation of the short-term memory can be partly relieved by using external knowledge representations, e. g., by taking short notes, or drawing a diagram or mind-map that help keep an overview over a somewhat larger set of ITEMS and quickly bring each single one into consciousness on demand (anecdotal evidence).

A knowledge worker cannot rely solely on her long-term memory to recall all encountered knowledge and task-related information later on. As a result, she has to create external representations as a memory-aid. Second, if the knowledge involved in the tasks becomes too complex she must work with external representations to reduce her cognitive load. Both of these cognitive limits can be addressed by providing adequate *external, modifiable, scalable representations*.

1.2.3. Shifting cognitive capacity limits with external representations

This section introduces *knowledge cues* as a special kind of external artefact.

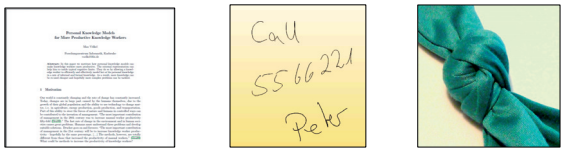


Figure 1.1.: Three examples of commonly used knowledge cues

Introducing knowledge cues The term cue[5] has been used by Haller (2003) to denote a stimulation which re-activates previously experienced knowledge. A *knowledge cue* serves as a *memory prosthesis* (Lamming, Brown, Carter, Eldridge, Flynn, Louie, Robinson, and Sellen, 1994). Fig. 1.1 shows three examples of commonly used knowledge cues: A handkerchief is considered as much a storable artefact as a hand-written note or a digital document. A letter written to a friend that is not intended to be read later again by its author is not a knowledge cue. An email, which is stored in the sent folder, should in most cases be considered a knowledge cue as most authors are well aware of the ability to re-read sent emails. While the knowledge cue is created, its creator has certain knowledge in mind that she wants to be reminded of. This is the purpose of creating the knowledge cue. For person A, an unread email from person B is not a knowledge cue. "If a piece of written material has not yet informed them, then they cannot sensibly file it anyway because its subsequent use or role in their world is still undetermined" (Kidd, 1994).

If the way the key is lying on the desk reminds a person of something, then this is only considered a knowledge cue if it has been placed there on purpose to act as a reminder. If the key has not been put on the desk with the *intention* to re-evoke this knowledge, it is not considered a knowledge cue in this thesis. A knowledge cue can also be a voice recording.

Reinmann and Eppler (2007) use the term "*personales Wissen*" to denote knowledge residing in a person. Conceptual and graphical knowledge can be represented by knowledge cues. This includes some knowledge which cannot be *externalised*, as defined by Nonaka and Takeuchi (1995). Hence, PKM deals with a broader spectrum of knowledge, compared to OKM.

Definition: Knowledge cue

Given all these characterisations, the term *knowledge cue* (as it is used in this thesis), can now be defined. **A *knowledge cue* is**

- **any kind of symbol, pattern or artefact,**
- **created with the intent to be used by its creator,**
- **to re-evoke a previously experienced mental state (activated knowledge), when viewed or used otherwise.**

[5] German term: "Hinweisreiz"

Strictly speaking, an existing artefact can also act as a knowledge cue if it is explicitly associated with a current mental state, e. g., "I take now this red pen which should remind me tonight of investigating Dr. Müller's idea to cancer research". Here the aspect of creation was just the explicit creation of the association, the red pen existed before. The definition includes artefacts created for target audiences of which the creator is a member, e. g., when somebody takes meeting minutes in a team meeting that will be read (among others) also by the person who took the minutes. *Using* includes internalisation and transformation. Colloquially, a knowledge cue is simply a "note to self". It does not need to be self-contained, as it suffices to remind the user of the knowledge that was present when the knowledge cue was created. The knowledge cue might or might not *contain* knowledge, the important characteristic is that it was created with some knowledge in mind and the purpose to re-evoke this knowledge later in its creator's mind.

The concept of knowledge cues explains how external representations can be useful, even if they do *not* contain knowledge. A knowledge cue might make no sense to others but evokes some kind of knowledge in the authors mind. Note that the creation of a knowledge cue is not always an *externalisation* process in the sense of the SECI-model. The knowledge cue is just a reminder for the person about some knowledge. The association between the artefact and a mental state is not externalised. What content do knowledge cues contain? Simple knowledge cues do not *contain* anything. E. g., a knot in a handkerchief does not "contain" anything. Other examples for simple knowledge cues are the letters of the alphabet, entries in a to-do-list, a photo of a colleague, or a thumbnail picture of a book cover. Knowledge cues are usually not self-contained, i. e., their interpretation often needs a lot of context information.

Knowledge is not only encoded in artefacts, but also in the relations between artefacts. More complex knowledge cues can be composed of a set of other knowledge cues, e. g., a picture album can contain a number of pictures. Similarly, a knowledge cue can be represented as a connection between other knowledge cues. In the analogue world, such connections can be represented merely by physical placement or more implicit encodings such as references in text or number systems on index cards.

Limited modifiability and scalability

Analogue knowledge cues are usually easy to create but expensive to modify and query. The most prominent PKM tool in use today is probably pen and paper. Even in the presence of powerful smart-phones and personal digital assistants (PDA), most people still use pen and paper for ubiquitous knowledge cue creation. However, pen and paper clearly has limits. It is not possible to re-arrange items on a sheet of paper without erasing and re-drawing or re-writing it. Second, a sheet of paper has clearly defined size, which limits the growth of a knowledge model. Using more than one sheet also has its management problems beyond a certain number of sheets. Both problems, modifiability and scalability, can be solved with digital knowledge cues.

Digital knowledge cues

Digital knowledge cues can be stored and retrieved on a computer. On a computer, a knowledge cue can be represented as a snippet of text in a file, in a filename, or even in a part of a filename. It can also be a folder, or the fact that a certain existing file has been placed into a certain existing folder. In this respect, the organisation and connection of information can itself be a knowledge cue. In fact, *any kind of manipulable state*, in the computer science sense, *can act as a knowledge cue*. A knowledge cue can be represented as a kind of *content* or as *connections between other knowledge cues*. Digital knowledge cues can contain a countless number of digital files such as documents, images, data bases, executable programs, etc. Digital knowledge models can be modified over and over again and their scalability is usually not limited by physical space but by the human's limited ability to navigate in large knowledge models.

For improved readability, the remainder of this thesis uses the term *knowledge cue* to denote a *digital knowledge cue*, unless noted otherwise.

Bernstein, Kleek, Karger, and monica mc schraefel (2008) introduce the notion of *information scraps* defines as "an information item that falls outside all PIM tools designed to manage it." Information scraps can be used as knowledge cues. A more detailed characterisation is given by Bernstein, Kleek, monica mc schraefel, and Karger (2007):

> Information scraps – small pieces of user-generated data which fit poorly within existing filing paradigms – represent a significant percentage of our information workspace, yet to date HCI research has found no satisfactory solution for them. Scraps run counter to many of the assumptions of our current filing paradigms: for example, assumptions that all data must be given an unambiguous filing location and name immediately upon creation. [...] We define information scraps as short, self-contained notes intended for their author's use. Information scraps typically span a few words to a few partial sentences in length, containing only enough clues to evoke the complete thought in the author's mind. They serve a variety of purposes, including functioning as reminders, memory aids, or holding places for incomplete thoughts.

Bernstein, Kleek, Karger, and monica mc schraefel (2008) list also some examples of information scraps that are not necessarily knowledge cues, if they have not been created to remind of a previously experienced knowledge, such as "Song lyrics and guitar tabs taped on the wall"or "A copy of an academic transcript saved in a text file". In short, information scraps are ill-managed personal information items whereas knowledge cues are explicitly created artefacts that might or might not be managed well.

Again, any entity from labels up to documents, or connections between entities, can be a knowledge cue. What matters, is that the knowledge cue has been created by a knowledge worker. Therefore an incoming email, which has not been read yet, is not a knowledge cue. Once the email has been filed into a folder – or at least been read – re-finding this email can re-evoke some knowledge. Going into the details, the subject line of an email,

of which the body has not been read yet, can be considered a knowledge cue. Ultimately, any collection of bits that a human associated with some mental concepts can be considered a knowledge cue.

Personal knowledge models From the perspective of a knowledge cue creator, a knowledge cue represents a part of a mental model. A set of interrelated knowledge cues can represent a larger, more complex mental model. This composition of knowledge cues is itself a knowledge cue (because the composition has also been created willfully to remind its creator of something). However, for compositional knowledge cues the term *knowledge model* is more appropriate. Note that this thesis uses the term *knowledge model* in a much broader fashion than, e. g., in artificial intelligence research.

Symbolic and non-symbolic models

Models can be categorised as symbolic, i. e., consisting of interrelated entities (which act as symbols), or non-symbolic models, e. g., miniature buildings as often used in architecture. Knowledge models are symbolic models. They represent knowledge as the choice of which symbols are used from a set of possible symbols that could be used, plus the relations between these symbols.

A preliminary definition[6] of the term *personal knowledge model* can be stated as follows:

A personal knowledge model is a collection of knowledge cues. These knowledge cues can be linked by directed[7], typed relations. Each knowledge cue can contain a piece of text, even a single word, or binary content. The knowledge model may additionally contain other artefacts than knowledge cues (e. g., unread emails or other digital objects).

The remainder of this work uses the term *knowledge model* to denote a personal knowledge model.

If the knowledge model contains knowledge cues created from others, it is a collaborative knowledge model. If it contains no knowledge cues, it is not a knowledge model.

Using external artefacts (here knowledge cues and knowledge models) allow reducing the cognitive load on the short-term memory (e. g., multiplying two large numbers assisted with paper and pencil) and helps to remember more facts over a longer period of time (e. g., writing a diary, using a lab diary, bookkeeping in a company). However, using external artefacts has also the cost of creating, maintaining and using them. The next section looks into ways to reduce some of these costs and increase the value of the external artefacts.

1.2.4. Automating symbol manipulation

Digital knowledge cues can be processed by a computer and hence allow *automating* manipulation processes. In practice, this automation is, e. g, manifested as full-text search, queries in a database, search- and replace commands, but also in advanced technology such as compilers or inference

[6] The complete definition is given on page **??**.

[7] Directed relations allow to encode undirect relations, but not the other way round.

engines (see below and Sec. 2.6. Automating symbol manipulation can lower the cost of information processing in such dramatic ways that some questions can be answered, which could not be answered before with available resources.

Generic external representations

Seminal articles by Bush (1945) and Engelbart (1963) describe imaginary *tools* that allow an individual to work more efficiently and more effectively with *generic external representations* of knowledge. Both projects let a user create and connect knowledge cues in flexible and sophisticated ways. As a result, a user is no longer restricted to editing linear documents or two-dimensional drawings but can instead create a complex, abstract representation of a mental model.

Degree of Explicitness The distinction of tacit and explicit knowledge made by Polanyi (1966) is still in use today, cf. Nissen (2005). The Cynefin model (Despres and Chauvel, 2000) and the Ba model (Nonaka and Konno, 1998), conclude that external and internal (tacit) knowledge are two extremes on a spectrum. I. e., knowledge resides partially in the minds of people and can partially be codified as external artefacts.

Boettger (2005) describes a formality spectrum from personal notes to documents intended for the general public which states that notes have to be more explicit, the less well known the audience is for the author or the less familiar the audience is with the topic. This notion can be extended by including a person doing PKM, who reads its notes on the same day, or weeks or years later (c f. Fig. 1.2). In this respect, writing for oneself later in time, can be seen as a special audience.

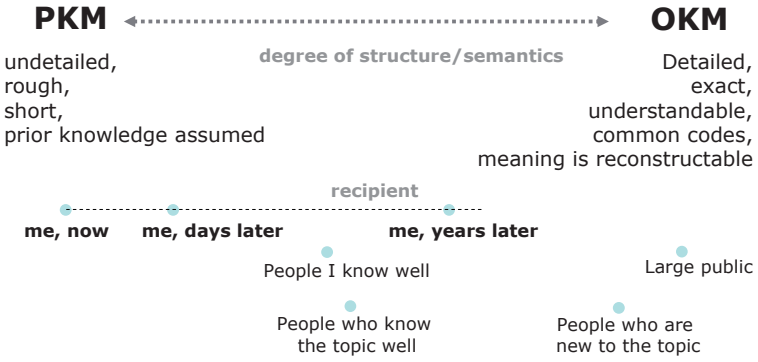


Figure 1.2.: From Personal to Organisational Knowledge Management. Based on Boettger (2005). An individuals at different points in time is compared to different social groups.

Degree of Structuredness Storing explicit knowledge, e. g., encoded as text and hypertext, in a computer allows retrieval, e. g., by full-text search and by browsing links. Structured databases and semi-structured document formats (e. g., XML) allow answering queries from the user with short, concise answers.

In databases, the user has the choice to put, e. g., a *street address* into a single field or represent it more fine-grained as *name*, *first name*, *street*, *number*, *city*, *zip code* and *country*. The fine-grained representation allows the user to perform much more elaborate queries, e. g., to find out which addresses in the database have the same city. In fact, the user adds *structure* to the data and tells the computer where to segment the data. The computer still has no idea, e. g., that Berlin is located in Germany. Hence a query for all cities *located in Germany* would yield no results, as "being located in" is not a concept in the database. If the user realises this, she can work around this by asking more complex queries, e. g., list all address entries in which the field city-name has the value 'Berlin', ignoring capitalisation. The information is there, it is just not very easy to use.

Knowledge cues can be in different degrees of structuredness. The term structuredness has been used by Boehm, Brown, Kaspar, Lipow, MacLeod, and Merritt (1978) in the context of software artefacts.

Background: Semantic technologies By representing knowledge in a formal knowledge representation language, e. g., in OWL (Schreiber and Dean, 2004) or RDFS (Dan Brickley, 2004), the computer can deduct new knowledge and answer queries about concepts.

Ontology

A central concept in the area of semantic technologies is an *ontology* (Staab and Studer, 2009). An ontology is a "formal, explicit specification of a shared conceptualisation" (Gruber, 1993). Technically, an ontology consists of a set of formal statements and natural-language labels for the concepts. A *conceptualisation* is a mental model of a certain aspect of reality. A conceptualisation is *shared*, if several parties agreed on it.

Personal ontology

Obviously, in PKM the *shared* part of the definition cannot be fulfilled. Nevertheless, researchers have adopted the term "personal ontology" for "ontologies" created and used by a single person. The term has also been used by Huhns and Stephens (1999)[8], Chaffee and Gauch (2000)[9], Park and Cheyer (2006)[10], and Dix, Katifori, Poggi, Catarci, Ioannidis, Lepouras, and Mora (2007)[11].

Ontologies and personal ontologies are created to encode knowledge in re-usable form. As such, they also remind the original author about this knowledge. Hence ontologies can be seen as a special kind of knowledge model.

Reasoning

Reasoning, also called *inferencing*, is built on the idea to declare a set of true formal statements in a formal language and let the computer deduct further true statements.

[8] Imagines a personal ontology mostly as an index on physical and digital information artefacts.

[9] A reference ontology for annotating websites is mapped to a personal ontology, in order to let the user browse annotated sites using his personal perspective.

[10] Within the IRIS semantic desktop

[11] Trying to convert personal digital artefacts into a resource for automatic user task-support, such as filling out web forms.

For such an input set of formal statements many names are used: *fact base*, *knowledge model*[12], *rule base* (if a rule-paradigm is used), or commonly *knowledge base*.

A more detailed introduction to semantic technologies is given in Sec. 2.6. Semantic technologies deal mostly with formal statements, e. g., there is usually no way to store structured text in an ontology. In PKM, informal knowledge needs to be represented, too.

Degree of formality Although explicit and structured, externalised knowledge can still vary largely in its degree of formalisation (cf. Sec. 3.2.2, *Augment* processes). Externalised knowledge can be anything between, e. g., a loose collection of keywords, a weakly structured text, an informal graph or hypertext, up to a highly structured knowledge representation and fully formalised knowledge base.

Researchers of formal models and semantic technologies acknowledge the existence of a "formality continuum" ranging from informal textual descriptions up to mathematically proven formalisms (Uschold, 1996, p. 3). Hussmann (1997) presents *hybrid formal-pragmatic specifications* in software engineering. Staab, Studer, Schnurr, and Sure (2001, p. 28) call it "degrees of formal and informal knowledge". Lethbridge (1991b) defines the two extremes of the formality spectrum for representing a concept: The most informal representation contains only bit maps of natural language text. An example would be an encyclopedia article about the moon. The most formal representation is expressed only via links to other completely formal concepts. These links have also to be completely formalised. An example would be the concept *moon* with a RELATION *orbits* to a concepts such as *earth*. The entities *orbits* and *earth* would themselves be linked to other RELATIONS, etc. This concept of a *most formal representation* suffers in practice from the problem of endless recursion. Such aspects are discussed in Sec. 2.1.

Structuredness and formality

Remark: In general, structure in knowledge representations tells a computer how to segment a larger chunk of knowledge. Formal knowledge representations tell a computer how different segments are related – if the meaning of the representations meta-model is understood by the software. Such relations can themselves be less or more formalised by successively answering questions such as:

- Are two entities related at all?
- If they are related, is it a directed relation?
- What formal type can be assigned to the relation?

It is possible to have a high degree of structure but no or low degree for formalisation. Formalisation always requires some degree of structure (representing knowledge as distinguishable entities), but can still vary for a given degree of structuredness.

[12] Note how this thesis uses *knowledge model* to denote a different thing, namely structured aggregations of personal digital knowledge cues.

The topic of semi-structured data has been explored in the XML community. The topic of semi-formal data has received little attention in the past. Chapter 4 describes a formalism for knowledge models allowing to represent different degrees of structuredness and formalisation.

Degree of formality in PKM

PKM often deals with knowledge that is somewhere in the middle of these extremes. E. g., note-taking is a core activity of PKM. An individual creates an external representation for internal concepts. Later, the external representation is internalised again to re-activate the knowledge in the individual's mind. If somebody writes a short informal note to himself it is often completely meaningless to others. The knowledge is thus not fully externalised – yet this note is an external reminder about some knowledge that the author would otherwise forget. E. g., a short note like "coffee" could mean anything from "buy coffee", or "don't forget to have a coffee with an old friend next Tuesday" to "download and install a tool called coffee". For a knowledge worker some marks on paper represent parts of a mental model (Kidd, 1994, p. 187–188). People often use their desks as a filing system where informal piles coexist with explicitly filed documents (Malone, 1983). In many cases, the knowledge is thus not fully externalised – Yet this note is an external reminder about some knowledge that the author would otherwise forget.

Oren (2006) requires to leave users their freedom and do not constrain them into rigid schemas. Kidd (1994, p. 190) goes even further and concludes a tool should concentrate on capturing and reproducing the appearance of marks made by knowledge workers *rather than interpreting them*. However, not interpreting any semantics of a user's marks seriously limits automatic processing, e. g., searching, browsing, transforming, and exporting. The important point is to not *force* the user to formalise, but to *offer* her the possibility to formalise in addition to capture and reproduce free-form content.

Humans are often not able to write down directly a problem description in pure logical formulas or other precise mathematical notation. It is a central aspect of problem solving to formalise a given problem gradually until it is fully understood. Until now, few tools or methodologies to support such systematic formalisation are available. Instead tools either support coarse-grained document-centric information management or give the user a limited set of modelling constructs for more fine-grained decision support, e. g., argumentation tools. The process of knowledge externalisation is currently poorly supported. Traditional knowledge articulation methods, like writing emails or slides, filling out word processing or spreadsheet templates, are all in a limiting, simple, linear form.

Maier and Schmidt (2007) describes a knowledge maturing process which starts with very informal, personal artefacts (idea stage), which are then discussed in a community until they are ready for formalisation. Afterwards the content can be taught to others in a systematic manner. In this respect, the PKM processes (as described in Sec. 3.2.2) support the first steps of the knowledge maturing process.

The higher the desired level of automated processing, the more formality

is required (Uschold, 1996, p. 3). The next section looks into the relation between formalisation effort and its benefit.

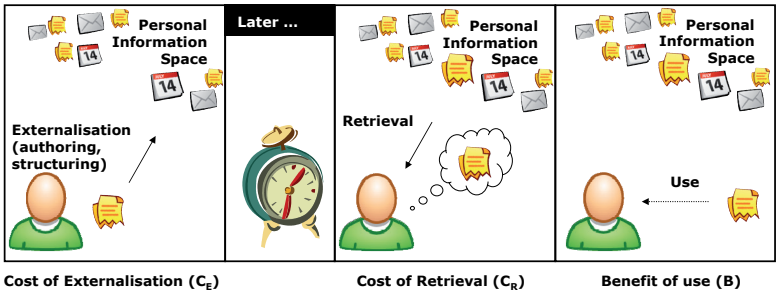
1.2.5. Economic considerations of modelling effort vs. benefit

There is a general trade-off problem between modelling effort and benefit.

First of all, it is clear that note-taking and modelling cannot only push cognitive limits, but also make knowledge workers more productive. How? As Davenport (2005, p. 72) notes, "the most effective knowledge workers reuse their own knowledge all the time." After having an insight (knowledge has been created) people often make a note in order to act as a reminder in case they forget their idea (externalisation).

Knowledge cue creation and usage

The process of managing knowledge cues can be simplified as (cf. Fig. 1.3):



Goal: more benefit than costs ($B > C_E + C_R$)

Figure 1.3.: Simplified model for cost and benefit analysis in PKM

1. Some parts of the implicit knowledge become external. Knowledge cues are created. The user has externalisation costs C_E.

2. Time passes by and the author might forget some or all details of the articulated knowledge.

3. At a certain moment, in the context of a certain task, the user initiates a retrieval process. The user reads the search results, and refines the search query. After some steps, the user either found one or several matching knowledge cues or cancels the search with no result. The process of reading through a list of search results is part of the search costs. If the knowledge cue is long in size, the time to read through it takes longer. If the desired knowledge is only a part of the artefact, reading through the artefact is thus additional search cost. All these costs are retrieval costs C_R.

4. If results were found, the user has some benefit from having available the external knowledge or from remembering knowledge from the knowledge cue. This benefit B depends highly on the current task and cannot be defined in general (Iske and Boekhoff, 2002).

The basic hope of a person doing PKM is to have more benefit B in the end, than the effort it took to externalise C_E and retrieve C_R the personal notes. Formally, the goal is:

$$C_E + C_R < B$$

More structure → better search

The more structured and the more formalised knowledge is, the more powerful become ways to work with this knowledge in a computer.

Better organisation, structuring, and formalisation of content is expected to lower the costs of retrieval (Glushko, 2006). For sentence retrieval tasks, more structure gives better results (Bilotti, Ogilvie, Callan, and Nyberg, 2007). Highly structured and formalised information sources are easier to search and process.

Adding more structure and metadata to a note enhances the probability for re-finding it, but also adds to the externalisation costs. Similarly, the more formal statements are in a knowledge base, the more thinking and modelling effort has gone into their creation. E. g., writing a post-it note is easy to do, but retrieval does not work well if there are thousands of them. On the other hand, creating a formal ontology, e. g., in Protégé (Noy, Sintek, Decker, Crubézy, Fergerson, and Musen, 2001) is very costly, but offers many more ways to query knowledge bases precisely, compared to merely structured ones. E. g., "List all cities with more than 100.000 inhabitants in Germany" instead of the much vaguer "List all documents which contain the term 'Germany' in the section 'Geography' " – and then manually try to find whether the article is about a city and what the population number might be.

Decision under uncertainty

If a person writes down a note today, she can usually not predict well, if or when that note is going to be needed again (Bruce, 2005). For rather short-term simple notes like shopping lists, the prediction is easy. For more long-term complex subjects such as product ideas or research notes, the prediction of the usage context becomes much harder. The benefit of remembering the right knowledge at the right time differs much depending on the usage contexts. If it will be useful in the future, she has to make sure the note will be found again in the right contexts. Deciding whether to store something at all, or at which degree of formality, is done by subconsciously weighting costs – the "least average rate of probable work" Zipf (1949) – and benefits Bruce (2005). PKM has to keep the balance between externalisation costs, finding costs, and anticipated value. **The user must have the freedom to decide how much effort to use for knowledge modelling.**

Current tools do not allow spending effort in arbitrary amounts: E. g., Either text is created via a text editor, without any formal metadata. This requires high effort at retrieval and re-use time. Or an ontology is created using an ontology editor, shifting all effort in the authoring phase. *But where is the tool allowing to create "something half text, half ontology?"* There is no generic model describing the needs and possible solutions for this unified representation of informal and formal knowledge, allowing stepwise formalisation.

A more detailed economic analysis of effort and benefit of PKM processes is given in Sec. 3.3.

1.2.6. Summary

The goal of this thesis is to make individual knowledge workers more productive. They face two prominent cognitive limits, namely short-term memory load and long-term memory recall. These limits are addressed with the concept of a *knowledge model*, an external, manipulable, digital artefact. More structured and more formal knowledge is cheaper to work with, but more costly to create. Hence a knowledge model must be able to express and use knowledge at various degrees of formality. The user must have the freedom to decide how much effort to invest in her knowledge model. Knowledge models are not static; users are expected to change them all the time. Hence knowledge models must allow users to add, remove, change, and incrementally formalise knowledge cues all the time.

The remainder of this chapter states the research questions of this thesis and presents a brief overview of the solution.

1.3. Research questions and contributions

This thesis tackles four research questions. Initially, the problem needs to be understood:

1. **Which factors influence costs and benefits in PKM?**

   This question is answered in Chapter 3 with an in depth-analysis of requirements for a PKM tool. Besides literature research, this includes an economic analysis of costs and benefits in PKM processes (see Sec. 3.3).

   The result argues for using personal knowledge models with the right level of granularity and range of expressiveness. Compared to classic knowledge representation, this thesis aims to reduce the articulation costs.

   This leads to the second question:

2. **What is a suitable model to represent and use artefacts in a uniform fashion that are in different degrees of formalisation?** The model should fulfil a number of requirements, which are gathered in Chapter 3.

   A Personal Knowledge Model formalism has been designed, able to represent semi-formal content and allowing stepwise formalisation (cf. Sec. 4.1).

   As a means to represent structured text and transform these structures into more formal constructs the *Structured Text Interchange Format* (STIF) has been developed (cf. Sec. 4.3).

A particular design problem, namely how to deal with a high number of semantic relations without confusing the user and how to map various existing structures into a unified yet extensible representation leads to the third research question:

3. **What is a unified top-level ontology for personal knowledge models?**

   To manage a large number of relation types, a top-level ontology of personal relations – the CDS relation ontology – has been developed from literature studies and analysis of existing tools (cf. Sec. 4.2).

   Finally, in order to evaluate the usefulness of the personal knowledge model, one has to ask:

4. **How can a tool for using personal knowledge models be built?**

   Chapter 5 describes the realised CDS implementation and a tool for PKM based on the CDS model.

The evaluation (Ch. 6) spans all four research questions. The relation between contributions and chapters is depicted in Fig. 1.4.

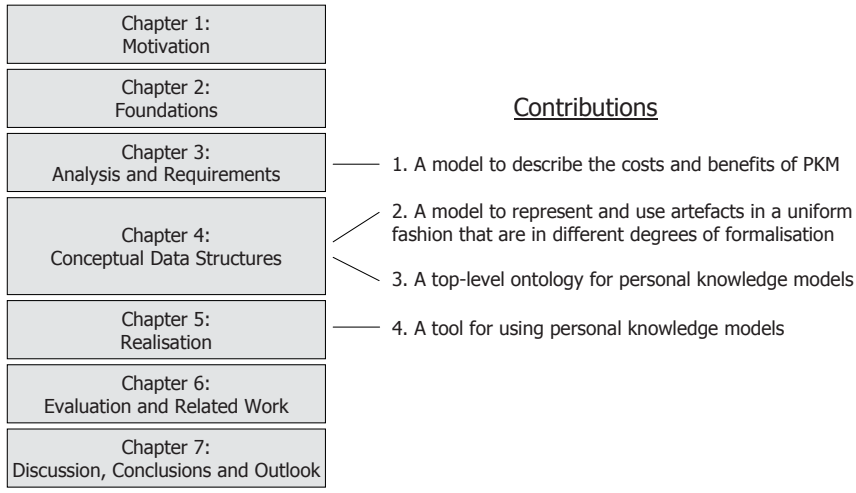


Figure 1.4.: Mapping from chapters to research goals

1.4. Solution overview

This section gives an overview of the complete solution which is developed in the following chapters.

To let the user profit from semantic technologies without the burden to use only formal knowledge, the approach of this thesis is to allow a friendly and useful coexistence of knowledge in different degrees of structuredness and formalisation. Additionally, it should be (cognitively) easy to formalise knowledge step-by-step (cf. Fig. 1.5, right). There is a trade-off between acquirability of a knowledge representation language and its expressive power (Gruber, 1989).

A knowledge model can be seen as a superset of documents and formal ontologies (cf. Fig. 1.5, left), because it contains both formal assertions as well as non-formalised content in one representation. Two definitions are

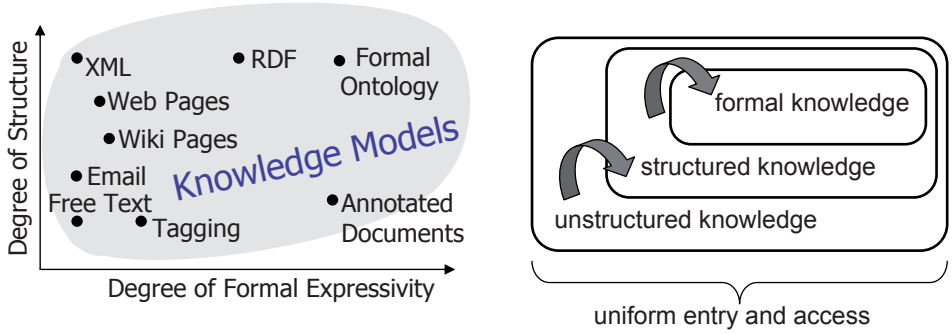


Figure 1.5.: Knowledge models unify different levels of formality

central for the solution:

Definition: Personal knowledge model

Definition: A personal digital knowledge model is a digital artefact which represents a set of knowledge cues. The knowledge cues can vary in size, structuredness and degree of formality.

Definition: Knowledge cue

Definition: A knowledge cue is either

- **a piece of content, containing plain text, semi-structured text, or arbitrary binary content such as images or desktop objects, or**
- **a connection between other knowledge cues. Such connections can be unspecified relations, directed hyperlinks and formal statements.**[13]

The core idea of this thesis can best be described as a brief series of design decisions:

[13] Formal statements are represented by linking term-like knowledge cues to each other.

- Cognitive limits in short-term memory and long-term memory knowledge are shifted with the use of explicitly represented knowledge in a knowledge model.

- The knowledge model must be

  flexible (i. e., easy to re-structure),

  expressive (i. e., allow but not force to represent formal knowledge) and

  generic, i. e., not domain-specific. Instead of constraining the user to pre-defined schemata, e. g., as in an address book or todo list, a tool should expose modelling on both layers, data and schema, to the end-user.

- Existing explicit digital knowledge should be importable into a knowledge model. And the representation formalism of the knowledge model should be easy to learn. For both reasons, the knowledge model should be a super-set of existing familiar formalisms to represent PKM knowledge in a computer.

- For economic reasons, it should be easy to structure and formalise knowledge in a step-by-step fashion and to give the user benefit for each level, e. g., retrieval should combine simple full-text retrieval with structured and formal queries.

Taken together, these ideas mean to expose the power of general-purpose modelling to knowledge workers for their everyday tasks. As an analogy, today, end-users can do numeric modelling in spreadsheet tools, using mathematical relations between cells as well as informal colouring and annotations. A user can either use a spreadsheet just like a sheet of paper or link different cells into complex formulas. This thesis aims at exploring and providing the same ease of use and mix of formality degrees for knowledge cues within a knowledge model.

A second analogy are 3D-CAD-tools which help construction engineers to design cars, by giving them an external, manipulatable model which they could not represent in their heads. Similarly, personal knowledge models can help users to develop complex concepts and ideas by providing a navigable, modifiable semi-semantic space.

The concept of semantic personal knowledge management as described in this chapter has been published by Oren, Völkel, Breslin, and Decker (2006).

2. Foundations

This chapter introduces a number of research fields that are referenced in later parts of this thesis. The sections on models, documents, hypertext, and semantic technologies have been the core influences for the CDS model. Fig. 2.1 gives a high-level overview about this chapter and shows which parts are used where. Entries marked with a plus (+) are analysed in detail in Chapter 3. Some parts give general background information, while other parts directly influenced the CDS model. The graphic has been created with the iMapping tool (cf. Sec. 5.3.1).

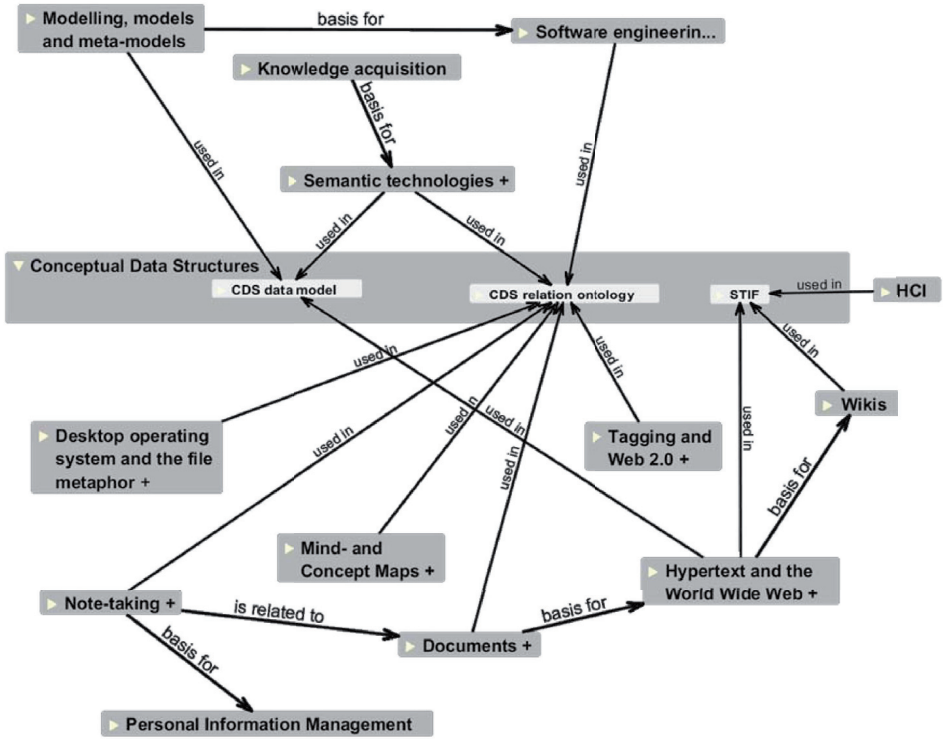


Figure 2.1.: Overview of the Foundations chapter. The parts shown above the *Conceptual Data Structures* are the formal parts. Below, more informal and person-centric topics and tools can be found.

2.1. Modelling, models and meta-models

The term *model* is central for the definition of *knowledge model* (c. f. 1.2.3). Generic *modelling processes* are later refined into a *knowledge cue life-cycle* (c. f. 1.2.3).

Define: Model

Models have three characteristic properties (Stachowiak, 1973, p. 131):

Mapping property: Each model has an original. Such an original might exist, have existed or be planned to exist. An original can also be another model.

Truncation property: A model represents not all attributes of the original, but only the ones that seem relevant to the model creator and model user. A model inevitably also has some properties not present in the original.

Pragmatic property: A model is created for a certain purpose, i. e., for a certain audience, a certain time span, and to solve a certain task.

Stachowiak (1973) distinguishes a number of models, classified into graphical models (images, drawings, and diagrams), technical models (e. g., miniature versions of real-world physical objects), and semantic models, consisting of symbolic entities. Semantic models are further subdivided into internal models (perception, cognitive models) and external models (spoken language, script, braille). External models consist of symbols and rules for using them.

This thesis deals with external models represented in a computer, i. e., finite, discrete models. A discrete model consists of a set of entities and relations between them. Not all entities need to be connected to other entities.

Modelling Modelling, the act of creating a model, is a core activity of human reasoning. Johnson-Laird (1983) proclaims "humans are model-builders". According to the widely used theory of constructivism (Maturana and Varela, 1987), each person creates his or her own mental model of reality. These mental models help to make predictions about the future state of the world and thus help to plan actions. Mental models range from simple physical models like "things fall down" up to complex theories for, e. g., stock markets or medical models about cells.

The first step in modelling is the definition of a set of model elements (E) and its mapping to the problem domain. The elements in a model must be distinguishable from each other, like elements in a mathematical set. The simplest possible modelling language is thus an enumeration of distinguishable symbols. i. e., it is an act of modelling weather a table is modelled as five entities (one "board" and four "legs") or just one entity (one "table"). A mapping from models to reality is always outside the scope of the model itself. The next step is the choice or design of a suitable meta-model for the task at hand.

Application development, document writing and even simple note-taking can be considered as modelling activities. The processes between a human and his PKM tool can be summarized as *modelling*. The CDS framework is designed as a suitable meta-model for modelling in the context of PKM tasks.

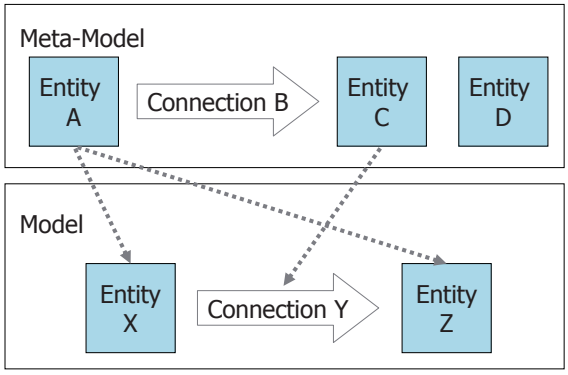
Formal Model of Models Let E be a set of model entities. A connection between two entities can then be represented by a tuple $(x, y) \in E \times E$.

To allow several connections between the same two elements, a finite set of connection labels R is defined as a subset of $\mathbb{N}$ together with a function c that maps each $x \in R$ to a pair of entities, i. e., $c(x) : R \to E \times E$. A set of labelled connections can be defined as $C = R \times E \times E$ with $\forall (n, x, y) \in C : c(n) = (x, y)$, where n is the label for a connection (x, y). A model M is then a set of entities and a set of connections, i. e., $M = E \cup C$.

Mapping this to simple node-and-link diagrams, E are the different nodes, and $E \times E$ are the links connecting the entities.

Meta-Model *This paragraph gives some definitions from the author of this thesis, based largely on concepts from meta-modelling in software engineering as published by the* Object Management Group *(OMG), i. e., in the UML standard (OMG, 2007).*

A meta-model MM is a model with the purpose to define the semantics of another model. A *typed model* is a set of entities and relations that are mapped to entities in a meta-model. The meta-model entities represent *types of entities* and *types of relations*. The meta-model defines semantics for instances of a type. Fig. 2.2 shows the general concept of layered models.



Legend: Boxes are items. Big arrows are relations. Thin dotted lines are *has instance* links.

Figure 2.2.: Modelling layers

Definition: Formal Model

Definition: A model M is a *formal model*, if all elements in M (all entities and all connections) are typed with entities from the same meta-model. If the semantics of the meta-model are defined, then the type-assignments state the semantics of the model. E. g., RDFS can be used as a meta-model for RDF triples. With this definition, a *semi-formal model* can be defined as a model in which some elements are typed with entities from the same meta-model. The *degree of formality* can be defined as the fraction of typed elements, cf. 1.2.4.

A set of meta-model elements together with rules how to use them – i. e., definitions of syntax and semantics – is often called a *modelling language*.

Recursion

A meta-model can again be described by a meta-meta-model, etc. There is in principle no limit on the level of meta-models used. To end the recursion, meta-models are often "defined in itself", which means they break the layering and assign entities within one layer as types to entities on the same layer. Definitions of such models "defined in itself" [1] are usually accompanied by rich natural language descriptions.

The process of modelling layers could be repeated ad infinitum, but in practice rarely more than three layers are used: model, metamodel and meta-metamodel.

Meta-model examples Imagine a simple node-and-link diagram with nodes labelled *Claudia*, *Dirk*, *SAP* and *SAP Research*. In addition, imagine a meta-model with entities like *person*, *organisation*, *employs*, and *owns*.

Now the node-and-link diagram can be turned into a typed model M, by relating each node and each link to a node in the meta-model. This typed model can now be used for a number of automatic tasks: E. g., M can be checked for validity ("Is each company owned by at least one person?") or automatically new knowledge can be deducted ("If a person owns a company A and that company owns company B, then the person also owns B").

An example for meta-modelling in the Unified Modeling Language (UML) can be found in Fig. 2.4 on page 46.

The metamodels used in software engineering usually offer a rich system of concept and relationship types to be used in instance models. Types in a meta-model are sometimes structured in an inheritance hierarchy, so that lower concepts inherit properties from their parents according to the semantics of the particular modelling language.

Software engineering metamodels usually strive for a clear separation of layers in order to define precise semantics, while other metamodels do not, e. g., RDF Schema (RDFS) allows meta-modelling in a way that classes can be also be instances.

Protégé by Musen (1989), cf. Rubin, Noy, and Musen (2007), is an example of a tool that allows authoring on two modelling layers, meta-model and model. Constraints posed in the meta-model are enforced by the user interface when modelling the instance data.

[1] An example can be found in the Java type system. A call of `getClass()` in a type `Person` returns an object of type `Class`. A call of `getClass()` on the `Class`-object returns again *Class*. See also Fig. 2.4.

2.2. Documents

Documents have been used for several thousand years now. The packaging format of knowledge – documents – did not change much. Although documents are an established means of communication, their creation is costly, slow and not always needed. Often only small parts of a document are needed to answer a given information need.

High costs, slow distribution

A French team of over 50 researchers analysed the term document in depth (Pédauque, 2003) and gives three co-existing definitions of the term "document": (i) *Document as form*, where a document is seen mostly as a container, which assembles and structures the content to make it easier for the reader to understand it. (ii) *Document as sign*, which emphasizes the argumentative structure of the content. Also, a document that can be referenced acts as a sign (placeholder) for its content. (iii) *Document as medium*, concentrates on the "reading contract", which is the intention or assumption of the author what will happen with the document.

There has been a debate about the "true nature of text". In 1990, some authors (DeRose, Durand, Mylonas, and Renear, 1997, reprint from 1990) believed in the OHCO-hypothesis. This stands for text as an "ordered hierarchy of content objects". Later, some of the same authors (Renear, Mylonas, and Durand, 1993) argued against this simplification.

An *information atom* is the smallest unit of content which can be interpreted without a document's context (but, of course, requiring background knowledge). E. g., for text, information atoms are single words.

Information Atom

Of course, an information atom can *act as* a knowledge cue (cf. 1.2.3).

How to model the structure and content of documents? A document can be seen as a structured entity, containing a number of information atoms. By "packaging" the information atoms together, an interpretation context is created, which influences how readers interpret it.

Packaging

Layers in a document Aspects of information in a document are:

Reference-ability Once a document is published, the reference can act as a placeholder for the content expressed within. A reference to a document can act as a meta-symbol on top of the symbols (knowledge cues) the document contains.

The usage of document references as symbols allows a document to "participate" in conversations, which probably lead to scholastic methods and modern academia.

Metadata Each document is written by a number of authors for a certain audience with a certain goal. By sending this process metadata along with the document, the reader has the ability to put the document in context and interpret it better. Such metadata is used by the reader as a frame of reference for interpretation and for search.

Sequential Navigation A document can typically be read from start to end by navigating through all contained knowledge cues. This is a

simple yet effective strategy to scan completely over a body of information.

Visual structure A document is not only a stream of sentences, but uses type-setting, i.e. bold, italics, different font styles and size, and placement of figures. Using only the visual structure, references can only point to page numbers. They can change when the document is, e. g., re-printed with a wider margin.

Instead of focusing on the visual properties of documents, such as distribution of content on printed pages, this thesis looks at the logical structure encoded by visual properties.

Logical structure The visual structure is used to encode a logical structure consisting of, i. e., paragraphs, headlines, footnotes, citations, and title. The logical structure makes it possible to reference smaller, meaningful parts within a document, i. e., "Sec. 4.2".

Argumentative structure On top of the linear content, a document follows an argumentative structure to convey its content to the reader. Argumentative structures appear on all scales. A typical structure is the "Introduction - Related work - Contribution - Conclusion"-pattern of scientific articles. On smaller scales, patterns like "claim–proof" and "question–answer" are used.

Content semantics Documents' contents mean something. Building upon logical and argumentative structure, the author encodes statements about a domain within the content.

Legal aspects This thesis focuses on documents for PKM and hence ignores legal aspects of documents, e. g., think of contract documents or intellectual property rights.

Digital documents Buckland (1997) argues, it's even harder to define the term "digital documents", e. g., in former times people used "log tables" to look up logarithmic values. Today, one would likely use a functionally equivalent software tool. The on-screen rendering of such a tool could be considered a document, too. Buckland (1997) sees a trend towards defining a document in terms of function rather than physical format. By following this trend, everything that behaves like a document *is* a document.

Structured text

Structures in text serve many purposes. They ease navigation and often the structure of a text reflects the structure of its content. In information systems, text appears either as part of an entry in an data base system or as a document, i. e., as a file. In the beginning of computerisation, documents were created with a typewriter or word processor. Then a printout was handed to the editor, which would typeset the whole text from scratch. Later the process has been streamlined and the document authors are now able to produce typesetting-quality files. As a result, *end-users are now used to create highly structured documents*. This allows using structured text as means to create structured knowledge cues, as described in Sec. 4.3.3.

Observations from analogue to digital documents

Prominent examples of digital documents are text processors files, hypertext documents and PDF files. Digital documents differ in many ways from analogue documents. In digital documents the visual structure is sometimes separated (e. g., via CSS, cf. Sec. 2.4) from the logical structure. This makes it possibly to execute queries based on the logical structure and, e. g., generate automatically a table of contents or return "all footnotes that contain a hyperlink". Additionally, other documents can now deep-link into a document, e. g., by using named anchors. From a reader's perspective, this effectively means that the granularity in digital documents is smaller compared to analogous documents.

A document has to be stable in time in order to become something reference-able. Only in this way people can cite the document without having to copy the content. This is not the case for all online documents and web pages: The content at any given URL can change at any point in time. Digital documents can therefore only replace or at least mimic classical documents in two ways: (a) a trustworthy source manages the web server and promises not to change the served content (a part of the business model of digital libraries) or (b) documents are sent as messages to the recipients, e. g., messages on email mailing lists. Nevertheless, with the advent of hypertext, the number of links between documents or parts thereof increased dramatically when documents became digital.

Augmented Digital Documents Documents can be annotated, e. g., to make the argumentative structure of a document explicit (Peter, Sack, and Beckstein, 2006). Hennum (2006) describes ways to encode argumentative structures in RDF. One basic observation is that modelling a document as a strict tree, i. e., as in XML, does not allow modelling overlapping regions. Hennum (2006) does not discuss the discourse structures themselves in much detail. This gap is filled by Groza, Handschuh, Möller, and Decker (2007) which describes a small yet expressive ontology for argumentative structures, which is based on Rhetorical Structure Theory (Taboada and Mann, 2006). Groza et al. (2007) models argumentation at the sentence level.

In a similar manner the metadata – e. g., who wrote the document when and why – can be made explicit. A system for annotating and relating documents in a visual way is described by Maier, Archer, Delcambre, Hy, Annareddy, Cassel, Gangula, Teng, Fox, and Murthy (2006). He goes beyond showing annotations next to a document, as the annotations themselves are forming a document on their own. Phelps and Wilensky (2000) describe the concept of *Multivalent Documents* for uniformly annotating different content types with rich annotation types. Annotated documents, stored together with their annotations, can be seen as a knowledge model.

Even more complex forms of digital and augmented documents are discussed in Sec. 2.4 on hypertext research.

2.3. Desktop operating system and the file metaphor

In 1981, the *Xerox Star Workstation*, one of the first personal computers, was released (Friedewald, 2000). It pioneered the WIMP-metaphor (window, icon, menu, pointing device) and placed digital documents, represented as little icons, in the heart of the user interaction. Files in the computer were modelled close to physical documents.

Since then, documents remained the dominant paradigm for user interaction, information exchange and archival. This is problematic, when search results return references to long documents, instead of shorter – and maybe even reference-able and annotation-able – information objects.

For more considerations on the effect of document granularity on costs in PKM see Sec. 3.3.

In file systems, a single folder name needs to be unique only among other files and folders in the same parent folder. As a result, a single folder name cannot locate a folder. Only a full path starting with the file system root denotes exactly one folder. E. g., the folder name "`My Pictures`" does not work, but "`C:\Documents and Settings\My Pictures`" does.

Semantic file system

The basic principles of file system explorers are analysed in Sec. 3.5.3. An extension of a standard file system to a semantic file system has been published by Bloehdorn, Görlitz, Schenk, and Völkel (2006).

2.4. Hypertext and the World Wide Web

Ideas from hypertext systems and hypertext research have been a strong influence for this thesis.

Remote web-sites play also an ever-increasing role in information work, as more and more personal data is stored in shopping accounts, online banking applications, social networking sites, online email, or online games. The model of the web has proven to be able to represent all kinds of content in a unified way.

Hypertext has been invented by Engelbart (1963) in his NLS (oNLine System) system, created in the AUGMENT project[2]. Based on this, an active hypertext research community created feature-rich models of hypertext. The *Dexter Hypertext Reference Model* (Halasz and Schwartz, 1990) presents a unified model with composite entities and n-ary links from analysing various hypertext systems, among them *NoteCards*, *Neptune*, *KMS*, *Intermedia* and *NLS*.

Usability of Hypertext

Hypertext is good if there is a large body of information organized into numerous fragments, the fragments relate to each other, and the user needs only a small fraction at any time (Golden Rules of Hypertext by Shneiderman, 1989).

[2] The features of NLS were so interesting that researchers have re-created the system with modern web technologies, see `http://hyperscope.org` (accessed 06.01.2010)

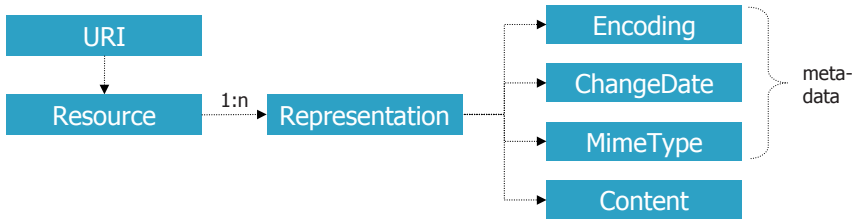


Figure 2.3.: The web model: REST

Web concepts REST is a term coined by Fielding (2000). It describes an *architectural style*, – that is a set of constraints on *connectors*, *components* and *connections* – which describes the conceptual model of the World Wide Web (WWW). REST is an important concept to understand web-based programming. REST describes a set of addressable *resources* which are manipulated by sending self-describing *representations* to them (cf. Fig. 2.3). One of the REST constraints is "hypertext is the engine of application state", which means each *representation* should contain the URIs of related *resources* (Fielding, 2000, p. 82). There is no defined way to model *typed* relations between resources.

REpresentational State Transfer (REST)

The WWW instantiation of REST consists of transport via the HTTP protocol, addressing via URIs and resource representation via markup languages such as HTML and XML.

WWW

URI A *Uniform Resource Identifier* (URI, Berners-Lee, Fielding, and Masinter, 2005) is an identifier for a resource in the WWW. A typical URI is a web address like `http://xam.de` but non-resolvable names[3] such as `urn:xam.de:20090828-08.31.17.053-0` are also URIs.

Representations WWW *representations* are character streams plus metadata describing the encoding, type of content and other metadata such as the last modification date. In practice, there are many more metadata fields, e. g., to control caching or compression of content.

Web content *This paragraph introduces popular web content formats which have been used as input for the Structured Text Interchange Format (STIF) model described in Sec. 4.3.1.*

Almost all web pages are written in HTML.

Hypertext Markup Language (HTML)

HTML is an instance of the Standard Generalized Markup Language (SGML, ISO, 1991b) language, which is a meta-language to describe languages. XML is a restricted and simplified form of SGML. It is introduced in Sec. 2.5.

HTML uses *inline-markup*, which is markup included in the sequence of characters. HTML provides elements for different purposes: text format-

[3] These are names that *should* never return a representation. However, for practical reasons described by Sauermann, Cyganiak, and Völkel (2007), resolvable URIs should be used in most cases.

ting, linking, document structure, user input forms, and semantic elements, e. g., "`address`".

The following content formats are frequently used in the WWW:

HTML 4.01 This version of HTML was published in 1999. The official URL is `http://www.w3.org/TR/html401/` (accessed 06.01.2010). It specifies three sub-types: *strict*, *transitional* and *frameset*.

HTML 4.01 Strict A trimmed-down basic version containing few presentational attributes. It contains 55 different element definitions.

HTML 4.01 Transitional This is HTML 4.01 Strict extended with *more* presentational attributes that were used in older HTML versions, e. g., `<center`, `<font>`, or `<strike>`.

HTML 4.01 Frameset This is HTML 4.01 Transitional *extended* with *frames*.

XHTML 1.0 Strict, Transitional, Frameset The same as their HTML 4.01 counterparts but with the additional requirement to be valid XML documents. This requires, e. g., an XML header plus a normalised way to write element attributes. XHTML 1.0 Strict (Pemberton, 2000) defines 77 elements. XHTML 1.1 is the subsequent recommendation edited by Altheim and McCarron (2001) which describes a modularised view on HTML.

CSS Cascading Style Sheets (CSS, Lie and Bos, 1999) is a W3C standard for defining the visual appearance of a web document written in one of the HTML variants. CSS defines things such as colours, fonts, border styles, margins, and space between elements. CSS allows separating content and presentation to a high degree.

Web programming *The following technologies are used in the CDS editor prototype, described in Sec. 5.2.*

Asynchronous JavaScript and XML (AJAX)

Traditional web applications ran all code on the server side, resulting in a slow and flickering user experience. Each time the user clicked, she had to wait for a complete page reload from the server. Network latency thus really added up as user wait times. Using JavaScript in the end-user's browser, often does not requires a page reload at all for navigation operations. If a user clicks somewhere, JavaScript requests only the needed data from the server and modifies the loaded page in the browser – without a page refresh. The result is a much more fluid user experience. This technique is often called AJAX[4]. There are essentially three kinds of state in an AJAX web application:

DOM The *Document Object Model*[5] (DOM) is the object-oriented, hierarchical model of web page rendered by a browser. A DOM can contain hidden elements as well, i. e., those currently not visible by the user.

[4] `http://en.wikipedia.org/wiki/AJAX` (accessed 06.01.2010)
[5] `http://www.w3.org/DOM/DOMTR` (accessed 06.01.2010)

Script JavaScript is loaded from a server and is run in the browser. JavaScript may create arbitrary object structures and is able to manipulate the DOM. Furthermore, in AJAX-enabled browsers, JavaScript is able to make calls to a web server to send or retrieve further data.

Server There are two parts of the server: The static part serves HTML pages, images, CSS files and JavaScript code; the dynamic part answers requests from AJAX applications. The distinction between the two parts is not always clear, as the AJAX application can also request static resources and the normal web resources can also be generated dynamically, e. g., dynamic images.

AJAX is used extensively in the realised CDS editor prototype, HKW, described in Chapter 5.

2.5. Software engineering

In the discipline of software engineering, modelling approaches are used at every level. The two most popular modelling languages, XML and UML, are introduced in this section.

Extensible Modelling Language (XML) The conceptual model of XML is the *XML info-set*[6] model. XML is a document-oriented model in which all content is represented in a hierarchy of labelled elements. The leaves of the tree can be (empty) elements, attribute values, or longer pieces of content (text nodes). XML is used widely for exchange of data between systems written in different programming languages and for communication of software modules over the internet (web services). XML document structures can be described with a *Document Type Definition* (DTD) or an *XML Schema* (Thompson, Sperberg-McQueen, and Gao, 2008). These schemata describe not only types but also the grammar, i. e., the nesting rules, of typed elements.

Unified Modelling Language (UML) The UML (OMG, 2007) is a visual language for describing software systems. It was initially created to improve communication among developers and later also used for model-driven design (MDD) and model-driven architecture (MDA)[7]. UML is much richer than the simpler, data-base oriented Entity-Relationship-Models from Chen (1976).

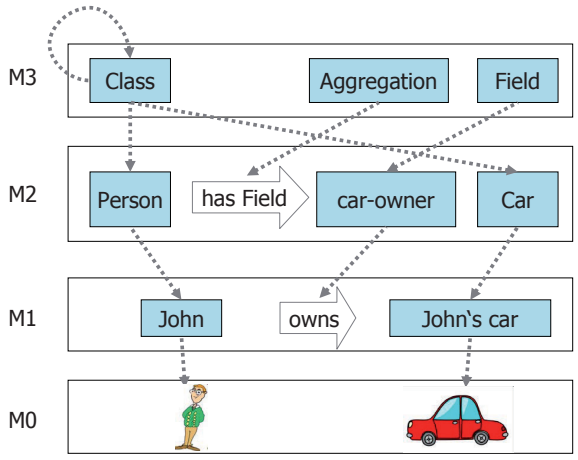
The most popular diagram type is the *class diagram* which talks about classes, inheritance, instances, attributes, visibility modifiers, and methods. UML is neither a language to describe *arbitrary* languages nor a suitable language for knowledge models.

OMG Meta-modelling stack

The OMG has defined four modelling layers (cf. Fig. 2.4 and *Recursion* in Sec. 2.1). This layering is widely used in industry.

[6] `http://www.w3.org/TR/xml-infoset/` (accessed 06.01.2010)

[7] MDA is a trademark of the Object Management Group (OMG), which created UML.



Legend: Dotted gray lines are *has instance* links. The images of John and John's car should mentally be replaced with the *real* John and his car.

Figure 2.4.: The four UML meta-modelling layers (simplified)

The four classical layers are:

M0 The reality. It contains real objects, like John or John's car.

M1 Digital objects that represent physical or conceptual objects, like a Java object modelling *John's car* and *John* himself.

M2 Classes of objects, like the class *Car* or *Person*. Defining behaviour of car-instances. E. g., the fact that they have an owner.

M3 Meta-classes. Here the concept of a *Java Class* itself is defined. To avoid further layers, this layer is defined in itself.

From each higher to a lower layer are *has instance* links.

2.6. Semantic technologies

Semantic technologies have numerous application areas. The two most prominent application areas are *data integration* and *reasoning*. A prerequisite for both are *data representation* and *ontologies*.

The Resource Description Framework (RDF)

Data representation in RDF The Resource Description Framework (RDF, Hayes, 2004; Klyne and Carroll, 2004) is the basic representation format for knowledge on the semantic web. RDF defines an extensible, graph-based model for integrating distributed, heterogeneous information sources. It reuses URIs for addressing (see Sec. 2.4). For a discussion on choosing good URIs for the semantic web see Sauermann et al. (2007). RDF was originally defined as a format to describe meta-data *about* resources on the web. It was not intended to *contain* the actual content of web resources. Fig. 2.5 shows the RDF data model together with the notion of Named Graphs (Carroll, Bizer, Hayes, and Stickler, 2004) as used in SPARQL

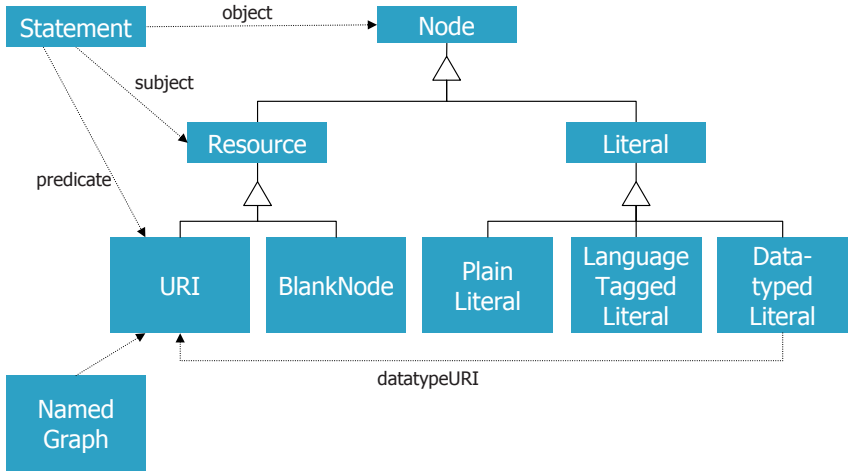


Figure 2.5.: The semantic web model: RDF

(Prud'Hommeaux, Seaborne, Seaborne, and Prud'hommeaux, 2007). Each *triple* in RDF consists of URIs (U), *blank nodes* (B) and *literals* (L) and is of the form $(U, B) \times (U) \times (U, B, L)$.

Namespace prefixes

To abbreviate URIs in technical documentation as well as in data formats, RDF uses a namespace-mechanism, similar to the one used in XML. The following example shows how namespaces are defined and used in the *Turtle*-notation (Beckett and Berners-Lee, 2008).

A namespace is defined as a pair of prefix and expansion URI. The prefix must end with a colon (":"). No prefix may be defined twice. Within a Turtle document, each occurrence of the namespace-prefix is then replaced with the expansion URI. As an example, this RDF triple:

```
<http://www.semanticdesktop.org/ontologies/2007/09/01/cds#hasDetail>
  <http://www.semanticdesktop.org/ontologies/2007/08/15/nrl#inverseProperty>
    <http://www.semanticdesktop.org/ontologies/2007/09/01/cds#hasContext> .
```

given the namespace definitions:

```
@prefix cds:  <http://www.semanticdesktop.org/ontologies/2007/09/01/cds#> .
@prefix nrl:  <http://www.semanticdesktop.org/ontologies/2007/08/15/nrl#> .
```

can be shortened to

```
cds:hasDetail nrl:inverseProperty cds:hasContext .
```

Beside its complexity and technical nature there are two other problems hindering its usage for PKM tools:

Problems with binary data Although RDF can conceptually contain binary data, stored in an `xsd:base64Binary` data-typed literal, there is no defined way to relate URIs non-ambiguously with content. Current RDF triple stores like *Sesame*[8] or *Jena*[9] are not designed to store larger binary chunks either. Programming libraries for RDF lack ways to describe, access, or change the content of web resources themselves.

[8] `http://www.openrdf.org` (accessed 06.01.2010)
[9] `http://jena.sourceforge.net` (accessed 06.01.2010)

Ill-suited for authoring A second problem with RDF is its lack of authoring tools. These can be divided into two classes: (1) generic: the user can change the schema at runtime, and (2) fixed-schema: the schema is pre-defined. An example of a fixed-schema tool is an address book editor which outputs its data in a fixed RDF format. Authoring generic RDF without a pre-defined schema is very flexible, but has usability issues: E. g., each RDF resource can have none, one, or multiple labels. It is an application level task to decide how to handle this. RDF can be called an *assembly language for data*, which can represent almost everything but lacks higher-order features to make it efficient for direct interaction with humans.

The strengths of RDF are (a) a well-defined process for merging several data sources, and (b) the ability to represent arbitrary graphs. Dealing with RDF directly requires quite a technical mind set, e. g., thinking about the distinction between literals, blank nodes and URIs.

Embedding RDF in HTML

There are several competing proposals on embedding RDF in HTML (cf. Sec. 2.4): *RDFa*[10] and *eRDF*[11]. In the CDS editor Hypertext-based Knowledge Workbench (HKW), the ideas from eRDF are used. A technical comparison between eRDF and RDFa can be found in Appendix A.1.

Ontologies An ontology is a formal, explicit specification of a shared conceptualisation (Studer, Benjamins, and Fensel, 1998).

In this thesis, an ontology is understood as a data structure in which all elements are formally typed (as defined in Sec. 2.1) with elements from an ontology modelling language. An ontology is usually a shared model of a certain domain upon which several parties have agreed on.

Ontologies in PKM

This definition can be stretched, so that a knowledge model created by a single person that contains only formally typed elements is also considered an ontology. This single person could be considered as two entities, me_{now} and me_{later}, so one single person agrees with itself to use a certain formal, explicit specification later on again. In this respect, knowledge models can be seen as a corner-case of ontologies. However, to avoid this definitional nit-picking, the term *knowledge model* is used in this thesis. Also, ontologies usually contain no document-like content, which is considered a normal case for knowledge models.

Ontology modelling languages An ontology modelling language is a language used to type elements (using the layering explained in Sec. 2.1). Several ontology modelling languages (sometimes also called ontology representation languages) are relevant in this thesis:

RDFS RDF Schema (RDFS, Hayes, 2004; Dan Brickley, 2004) is both an RDF vocabulary and a schema. As an RDF vocabulary, it defines formal semantics for certain triple patterns. As a schema, it allows modelling class (`rdfs:subClassOf`) and property hierarchies

[10] `http://www.w3.org/TR/xhtml-rdfa-primer/` (accessed 06.01.2010)
[11] `http://research.talis.com/2005/erdf/wiki` (accessed 06.01.2010)

(`rdfs:subPropertyOf`). RDFS has no clear separation of modelling layers; a class can be an instance of another class. Classes in RDFS are extensionally defined sets.

RDFS uses an open world assumption: everything not explicitly excluded could be true. RDFS has no way to state negation, therefore the answer of a reasoning engine to the query "Is triple (a, b, c) true in this model?" can only lead to "Yes" or "Not as far as I am aware, but it still could be true".

OWL OWL (Schreiber and Dean, 2004), and OWL2 (Hitzler, Krötzsch, Parsia, Patel-Schneider, and Rudolph, 2009) are more complex and more powerful languages than RDFS. OWL is a description-logic-based, declarative formalism. They are designed at the borderline between computability and expressivity. OWL 1 defines three *languages* with different expressivity/computability features. OWL 2 introduces three *profiles* which achieves the same effect. OWL allows negation, has an open world assumption, and mandates a strict separation of modelling layers. Classes are defined intensionally. OWL supports *optionally* inverse relations.[12]

NRL The NEPOMUK Representation Language (NRL, Sintek, van Elst, Scerri, and Handschuh, 2007) has been created in the NEPOMUK project for modelling ontologies on the semantic desktop (see end of section). The main difference to RDFS and OWL is a closed world assumption: If a fact is not stated as true, it is considered false. This matches expectations of users better, as a local desktop is indeed a closed world with a limited, known, processable number of files. NRL integrates the concept of multiple graphs, similar to the Named Graphs defined by Carroll et al. (2004). Being able to manage multiple graphs makes data management much easier and connections between different data-sets can formally be defined. NRL supports optionally inverse relations.

Data integration Data integration is a core application area for ontologies. The schema part of an ontology (T-Box) can be seen as a definition of a data model, i. e., the instances (A-Box) – or as unified view on several data models.

Imagine two data bases, one has a table for *customers* which have a *phone number*, the other one has a table *client* with the attribute *contact information*. Using ontologies, one can state that

(*client*, *is the same as*, *customer*) and

(*phone number*, *is a sub-property of*, *contact information*).

Such a global *mapping ontology* helps people to understand the structure of the two databases better, but foremost it can be used to automatically transform data or queries[13], so that queries can be posed across both databases.

[12] This becomes relevant later, when the requirement for mandatory inverse relations comes up.

[13] Which one is transformed is an implementation detail.

This would allow a user to query for the *contact information* of *customer*s and retrieve both the *phone number* and the *contact information* fields from the *customer* and the *client* table. Of course, asking specifically for the *phone number* would return only values of this field.

Data integration in PKM can help in three kinds of tasks in semantic-based PKM (cf. Sec. 3.2.2: import, export):

1. For integrating data from different applications that are used for creating knowledge cues. This is explored in semantic desktop research (see below).

2. For integrating different versions of somebody's own conceptualisations. E. g., to enhance the precision of full-text search if the user started to use another term for an old one, so that the query for "Universität Karlsruhe" returns documents that contain "KIT".

3. For integrating different knowledge models (or parts thereof) created by different people.

Reasoning In the context of the semantic web, *reasoning engines* (also called *inference engines*) are often used to deduct further knowledge from a knowledge base. Many inference engines are based on an interpretation of a set of triples as a set of formal statements. The semantics depend on the chosen formalism (e. g., RDFS, OWL Light, OWL DL, OWL2, F-Logic, . . .). An inference engine is a program that implements the semantics of a given ontology language. There are two basic technologies used. One way, called *forward chaining* or *materialisation* computes all facts than can be computed and stores them. For some ontology languages, the amount of facts than can be inferred is infinite. E. g., even the empty ontology in OWL DL has an infinite set of logical true statements. For such cases, or for performance reasons, *backwards-chaining* is used, which means a query from a user is used to recursively trigger computation until the program can either find a deduction tree from given facts to the query or know that no such tree exists in a given set of facts.

There are two possible principle benefits that knowledge workers can get from reasoning:

1. Reasoning can make knowledge workers more efficient by letting them find knowledge cues or derived conclusions *quicker*.

   E. g., if a user took a number of notes about two topics A and B, and later decides that B can be considered a sub-topic of A. Then, after adding a formal super-topic statement (A is-a-super-topic-of B), a reasoning engine can return also the notes with topic B when the user queries just for A. This allows the user to forget the old notes and the old topic, as they will be brought up when she searches for the new topic.

2. Reasoning can make knowledge workers more effective by letting them find conclusions where *none could be found manually*. Here the de-

rived knowledge is – although just mechanistically derived from user-stated knowledge – so new that it is an insight for the user to see it. Most spreadsheet calculations fall already in this category.

Semantic Desktop The semantic web is currently evolving in two places: As semantic web and as semantic desktop (Decker and Frank, 2004; Decker, Park, Quan, and Sauermann, 2005).

For effective PKM, a user needs access not only to his personal notes, but also to existing more structured items on his desktop, like address book entries, tasks, or arbitrary office files.

Computers still provide limited support for capturing and managing structural relations between knowledge units in a natural and pervasive manner across desktop applications (Wiil, 2005).

The European project NEPOMUK[14] researched the *social semantic desktop*. It contains two components to *unify access of desktop objects*. The *Aperture*[15] project transforms diverse file formats and application objects into RDF and full text. Another component, the *Beagle* desktop search engine allows searching in a unified way across all crawled data. The tool presented in Ch. 5 is also part of the NEPOMUK project.

2.7. Note-taking

This section presents studies on paper-based, physical note-taking as well as on digital, software-based note-taking.

Paper-based approaches Existing approaches to personal note management are paper-based approaches such as sticky-notes, paper notebooks, or a *Zettelkasten* (Luhmann, 1992).

The paper-based approaches are hard to automate. In a Zettelkasten one has to traverse the links from note card to note card manually. A Zettelkasten can be seen as a predecessor of wikis, which are introduced in Sec. 2.9.

On paper it is especially costly to *change* the content of notes or relations to other notes. Sometimes a complete note has to be rewritten. Also there is no ability for full-text queries or semantic queries.

Paper notebooks

There are few published studies on the structure of physical personal notes. A study by Dienel (2006) examined personal note books used by engineers around 1850 and later. Typical entry types were ideas, projects, addresses, to-do lists, meeting minutes, data from measurements, and appointments. Most engineers used one or two diaries at the same time, one for factual knowledge and the other for more process-related knowledge. The study also remarks the prominent use of different colours and carefully added table of contents. The artefacts in the notebooks are either text snippets representing one of the types listed above, or drawings or tables.

[14] `http://nepomuk.semanticdesktop.org` (accessed 06.01.2010)
[15] `http://aperture.sourceforge.net/` (accessed 06.01.2010)

Time and notes

A relevant feature of all personal notes is their durability. An interview conducted by Khan (1994) with 28 participants revealed that 64 % of interviewees kept their notes for years before throwing them away. 20 % of people stored their notes for months and only 8 % regularly threw away their recently made notes. Hence, notes must be retrievable, understandable and usable after years.

Software-based approaches There are many software-based approaches for note management; almost all of them allow full-text search and a virtually unlimited amount of personal notes. Unfortunately, search is not enough for PKM. As Barreau and Nardi (1995) point out, there is also a need to organize notes so that a note is even found if the user is only querying for a related note or browsing to a certain folder or category. Personal notes often have internal structure and relations to other notes. These relations are hard to manage in plain text files and the file system.

People remember personal notes after 8 days

Bernstein, Kleek, monica mc schraefel, and Karger (2008, page 4) reports on an evaluation of a note capturing tool. They ran an eight-day long study with 14 participants who used the tool in their everyday life. The features of the tool were rated "... unbeneficial to our participants over such a short period of time, due to both the small number of notes they accumulated and our participants' still-intact memory of notes' contents." To evaluate better, they plan "participants must gather notes over a long enough period of time that these mechanisms may become useful, or instead seed the application with existing notes."

People forget fictitious stories after 7 days

Kalnikaitė and Whittaker (2008) report on a study in which the ability to remember facts from fictitious stories was measured after one day, seven days and thirty days. The 25 participants used either no tool, pen and paper, or a digital note-taking tool. The study found a significant decay of memory for participants using no tools between one day and seven days, but not much further decay after thirty days. This is somewhat contrary to the study of Bernstein, Kleek, monica mc schraefel, and Karger (2008), where people could still remember after eight days. The difference might be the test content: real personal notes vs. facts from fictitious stories.

In the study of Kalnikaitė and Whittaker (2008), participants which used a tool had a drop in accuracy after a month. After that period of time, the authors speculate, the participants could no longer make sense of their own notes. They conclude that pen and paper notes are often usable by its creator to trigger memory recall after a period of 30 days. The study also examined the differences of usage in pen and paper versus digital tools and found "Digital and analogue note-takers tend to exploit space in similar ways, to use equivalent numbers of bullet points and to take similar volume and quality of notes."

Digital vs. analogue tools

Examples for personal note taking tools are *Evernote*[16], *PersonalBrain*[17] or the *Notes* function in smart phones and in *Microsoft Outlook*.

[16] `http://www.evernote.com/` (accessed 06.01.2010)
[17] `http://www.thebrain.com/` (accessed 06.01.2010)

2.8. Personal Information Management

PIM is an acronym with two closely related meanings: Personal Information *Manager* and Personal Information *Management*. In this thesis, PIM is used to denote the latter.

Personal Information Manager A Personal Information Manager is an application that can manage personal structured data such as address book entries, appointments, tasks, and notes. Popular PIM tools – or PIMs – are *Microsoft Outlook*, *Apple iCal*, *Kontact*, and *Lotus Notes*. Smartphones usually have built-in PIM software. The more advanced semantic desktop systems (cf. 2.6) are also personal information managers, e. g., the semantic task manager from Grebner (2009).

Existing PIM tools either focus on specific structured data such as appointments, to-dos or contact data – or tackle only free-form note taking. E. g., management of CD collections, cocktails recipes, text fragments, ideas, the personal social network, structured argumentation, bibliographic data, or web site logins is usually poorly supported. Plain text notes are in most PIM tools merely an unstructured, unrelated set of memo items.

Personal Information Management PIM is a research field that investigates how and why people manage personal information. The first publications explicitly mentioning the field come from 2001 (Jones, Bruce, and Dumais, 2001), since 2004 international workshops discuss the topic. The field of academic Personal Information Management (PIM) research is much broader than the study of PIM tools.

Personal Space of Information (PSI)

The report of the second PIM workshop (Jones and Bruce, 2005) defines the term Personal Space Of Information (PSI) as the space that "includes all the information items that are, at least nominally, under that person's control (but not necessarily exclusively so)", no matter if the information is analog or digital. This includes, e. g., all personal notes and all emails a person received or sent. The field of Personal Information Management (PIM) aims to help individuals to manage all artefacts in the PSI.

Keeping found things found

Jones et al. (2001) introduces the problem of "keeping found things found" which reports on the tension between *knowing* something and merely *storing* something. Academic PIM research analyses problems such as storing, finding and re-finding files, web pages and emails.

From / To	Myself	Others
Me	Personal notes (PKM and PIM)	Email sent folder (PIM)
Others	Email inbox, news feeds (PIM)	Web, mailing lists

Table 2.1.: Comparing Personal Information Management (PIM) and PKM

Comparing PIM and PKM PKM can be seen as a sub-field of PIM research, focusing on self-authored, personal (semi-)structured, (semi-)formal

notes, which do not fit in to the existing structures. As depicted in Tab. 2.1, PIM deals with artefacts sent to and from a person, whereas PKM focuses on artefacts created and consumed by the same person.

PKM could also be seen as a super-field of PIM, the goal of PKM is to help the individual to digest, collect, understand, use and re-use *personal knowledge* more effective and more efficiently, whereas the goal of PIM is "merely" managing *information*.

Without the notion of *knowledge cues* (cf. Sec. 1.2.3) such debates over knowledge vs. information are endless. In this thesis, the term PKM is mostly used for managing personal knowledge cues (PKM as a subfield of PIM).

2.9. Wikis

This section introduces wikis and its semantic extension, semantic wikis.

Wikis Ward Cunningham developed Wiki concepts and implementations in 1995. A wiki is a web-based system for collaborative creation and usage of content. Wikis have a simple conceptual model: Each page has a name, which is a short string that can be typed on a keyboard and be remembered by the users. Attached to each page title is the wiki page content. Page content consist of a longer string of characters which are interpreted by the wiki render engine to produce HTML. Special syntax in the page content is interpreted as links to other pages. Links are established by referring to other wiki page titles. Empty wiki pages represent concepts with no description attached.

Maybe by using simple titles for pages instead of long, error-prone URLs and by integrating the knowledge articulation tightly to the knowledge retrieval process, wikis became the first widely deployed tool for authoring emerging, networked note structures. They are very popular in intranets, as lab diaries, as public encyclopedia (Wikipedia) and as personal notebooks.

Wiki Syntax Wiki syntax is the way textual input is interpreted as HTML with document structures and hyperlinks.

Typing Structured Text

In emails most people carefully use ASCII-formatting in order to make reading easier. They encode logical structures of the text in visual ASCII formatting. For example, list items are very often stated with leading stars or minus signs. The human reader hereby can easily conclude that all list items share some kind of common characteristics. Text structures often carry a kind of implicit semantics, e. g., all items in a list are persons invited to a meeting. As soon as the semantics of the list would be known to the computer, a user could query for it.

Wiki-Syntax

The result of parsing wiki-syntax is usually an HTML document. Wiki syntax can be learned in a try-and-see manner, because no wiki syntax leads to error messages. As the worst result, an undesired document structure can appear.

After the success of wikis, the concept of lightweight markup languages became more popular and other generic text-to-HTML languages emerged outside of wikis, e. g., Textile[18], Markdown[19], or reStructuredText[20]. Note that both wiki-syntax as well as the other lightweight markup languages produce HTML.

Alternatives to wiki syntax

Another way to process textual input has been explored by Miller (2002). The paper describes a text editor with a special treatment of text selections. If a user selects a span of text, the editor tries to select similar spans of text in the same document, e. g., text spans that are preceded and terminated with the same string. The LAPIS tool uses a set of heuristics for this. Based on these ideas, Gerke (2005) presents an approach where a user can select a number of positive and negative examples and additionally state nesting rules between different sets of text spans. This allows a user to create a document grammar by virtue of machine learning.

Semantic wikis In a semantic wiki, the user can state the *type* of a link. Instead of merely linking *Berlin* to *Germany* the user can type the link with *is capital of*. Semantic wikis allow representing informal knowledge cues (text), semi-formal knowledge cues (structured text) and formal knowledge cues (typed links).

One of the first published semantic wiki was *Platypus* from Tazzoli, Castagna, and Campanini (2004). The first academic meeting was the workshop on semantic wikis by Völkel and Schaffert (2006).

Personal semantic wikis

Semantic wikis are designed and used not only for collaborative use, but also for personal knowledge management (PKM, Oren, 2005; Oren, Völkel, Breslin, and Decker, 2006). Semantic wikis allow stepwise formalisation of content: First a page is created, then filled with text, spell-corrected, structured, re-structured, and linked to other pages. Then links are typed and pages linked to categories.

Ironically, just like with paper-based approaches, *changing* things is not that easy in semantic wikis. Tasks such as moving content from one page to another or renaming a relation require typically an administrator to run scripts over the database. Second, a common use-case of PKM tools is the need to import knowledge from external sources. In most semantic wikis, the import of semantic data needs to be represented by artificially generated wiki syntax inserted into pages.

Semantic MediaWiki (SMW) One of the most stress-tested and used wiki engines is the open-source *MediaWiki* engine, which is used in all *Wikimedia Foundation*[21] projects such as *Wikipedia*, *Wikibooks*, *Wictionary*, *Wikinews*, *Wikibooks*, and *Wikiquote*. The open-source extension *Semantic MediaWiki* (SMW, Völkel, Krötzsch, Vrandecic, Haller, and Studer, 2006;

[18] `http://textile.thresholdstate.com/` (accessed 06.01.2010)
[19] `http://daringfireball.net/projects/markdown/` (accessed 06.01.2010)
[20] `http://en.wikipedia.org/wiki/ReStructuredText` (accessed 06.01.2010)
[21] `http://wikimedia.org/` (accessed 06.01.2010)

Krötzsch, Vrandečić, Völkel, Haller, and Studer, 2007) turns MediaWiki into a semantic wiki engine.

Semantic MediaWiki extends the MediaWiki link syntax "... `[[Germany]]` ..." to "... `[[is capital of::Germany]]` ...". Used on a page like "Berlin", this encodes a triple (Berlin, is capital of, Germany). Additionally, a user can write "... `[[population::3,426,354]]` ..." to denote a triple (Berlin, population, "3426354") with a literal value. However, it is not possible to link to such content snippets ("3426354") with another link. These content snippets in SMW are not first-class entities.

SMW uses the MediaWiki namespace mechanism[22] to provide a wiki page for each property, e. g., `Property:Has_capital`. This allows users to document the intended semantics of the property. Additionally, the range of the property can be stated using another semantic link, which is interpreted by SMW. SMW provides the ability to download an RDF file containing all the semantic information which has been created by the user via wiki syntax.

HALO extension

SMW itself has been extended in numerous ways[23]. The *HALO extension*[24], also known as *SMW+*, extends SMW with easy-to-use annotation and refactoring tools.

2.10. Mind- and Concept Maps

This section introduces *mind maps* and *concept maps* briefly. They are tools for structuring, organising and refining knowledge. The conceptual models of both are analysed in Sec. 3.5.5.

Mind Maps Besides standard office document formats – like text document, spreadsheet, presentation slides, and diagrams – mind maps are becoming a popular format, too. Mind maps were invented by Buzan (1991) as a paper-based method to help people learn new material faster and better. As a computer application, mind maps are mostly used to (re-)structure items or to help capturing ideas in discussions.

Popular mind-map software tools are the commercial tool *MindManager* and the open-source tool *Freemind*[25] Examples for mind-maps created with *Freemind* can be found in Figures A.1–A.15 in Appendix A.5.5.

Concept Maps Concept maps have also first been used on paper. A concept map is a set of labelled boxed which can be connected by labelled lines or arrows. The concept map approach is described in detail by Novak and Canas (2006). A line or arrow may have several start or end-points, although such multi-lines occur seldom.

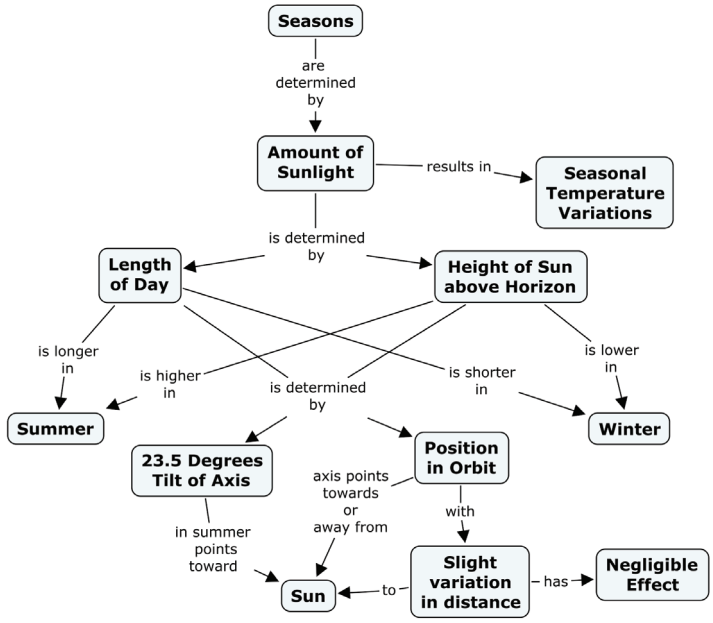
[22] `http://www.mediawiki.org/wiki/Help:Namespaces` (accessed 06.01.2010)

[23] `http://semantic-mediawiki.org/wiki/SMW_extensions` (accessed 06.01.2010)

[24] `http://sourceforge.net/projects/halo-extension/` (accessed 06.01.2010)

[25] `http://freemind.sourceforge.net/.` (accessed 06.01.2010)

Digital concepts maps are used for learning (Jüngst, 1992) and PKM (Tergan, 2005). An example that shows typical Concept Map usage is given in Fig. 2.6.



Source: Originally published by Novak and Canas (2006, Fig. 5, p. 10) with the title: One representation of the knowledge structure required for understanding why we have seasons

Figure 2.6.: Example for a Concept-Map

Comparison to Conceptual Graphs

Concept maps are similar to *conceptual graphs* (Sowa, 1976) in their attempt to map natural language sentences rather directly to a graphical representation. However, different from conceptual graphs, concept maps have no direct formal interpretation. Conceptual graphs on the other hand, have no means to express informal knowledge.

2.11. Tagging and Web 2.0

Tagging is the new name for an old concept. The method of assigning keywords (descriptors) to documents (books and journals) dates at least back to the first publishing of "Rules for a dictionary catalogue" by Cutters (Löffler and Fischer, 1956, p. 124).

Tagging is the process of assigning freely selected keywords to web pages, images or other content collections. Often tagging is used in a collaborative fashion, i. e., multiple users are assigning their keywords to a (mostly) shared content collection.

Popular web-sites using tagging are *delicious*[26] (bookmarks) and *BibSonomy*[27] (bookmarks and publications) or *SOBOLEO*[28] from Braun, Schmidt, and Zacharias (2009).

Tagging systems are also used privately, e. g., in the popular note-taking tool *Evernote* or *Apples iPhoto*.

Folksonomy

Clary Shirky coined the term *folksonomy* in his post "Ontology is Overrated: Categories, Links, and Tags"[29]. A folksonomy is described as the set of tag assignments (user, tag, resource). Even for a single user, the tuples (tag, resource) imply a number of interesting sets such as (Notation: ?=query variable, $*$=any):

- $(t, ?)$: The set of all resources tagged with tag t. This is the most common retrieval query.
- $(?, r)$: The set of tags used for a certain resource r. This is a kind of summary about the resource.
- $(?, *)$: The set of tags. This is the vocabulary of a user.

In a collaborative system, these queries can be extended to all users. This results in resources or tags occurring multiple times in the implied sets. This set can easily be ranked by the number of occurrences. Alternatively, all resources occurring less than k times can be left out. This improves precision and degrades recall. Additional queries possible in collaborative systems are:

$(*, t, ?)$: The set of all resources tagged with tag t by any user, implies a topically grouped resource collection.

$(?, *, r)$: The set of users that have tagged a resource r. This implies a set of users with similar interests, but maybe very different perspectives and viewpoints.

$(?, t, *)$: The set of users that have used tag t. This is a set of potentially related users.

By taking into account groups of tag assignments, further interesting sets can be computed. For example, by looking at all tags a user A used, users with similar tags can be identified; then resource from related users can be suggested to user A. This approach is illustrated well by Jäschke, Hotho, Schmitz, Ganter, and Stumme (2008).

Categories

The classification system used in Wikipedia (categories) is essentially the same as tagging with the additional ability to nest categories. Nesting of tags is also available in *delicious*, although to a lesser degree and only one nesting step is allowed there.

The Semantic File System (Bloehdorn et al., 2006) also uses tags as a core organisation principle.

[26] `http://delicious.com/` (accessed 06.01.2010)
[27] `http://www.bibsonomy.org/` (accessed 06.01.2010)
[28] `http://www.soboleo.com/` (accessed 06.01.2010)
[29] The main article can be found at `http://www.shirky.com/writings/ontology_overrated.html` (accessed 6.11.2009)

2.12. Knowledge acquisition

This section looks into knowledge acquisition aspects.

Knowledge elicitation methodologies Schreiber (1993) published a book on the KADS methodology for *knowledge base development*, which is a methodology with the goal of creating a formal model of a domain. KADS

CommonKADS (Schreiber, Akkermans, Anjewierden, Dehoog, Shadbolt, Vandevelde, and Wielinga, 1999, Ch. 6) is a refined model, emphasising formalised task-specific knowledge. It explains a methodology for knowledge elicitation from non-experts by experts. CommonKADS

Another methodology based on KADS is the "MIKE"-approach (Angele, Fensel, Landes, and Neubert, 1995). It emphasizes incremental modelling and the notion of semi-formal models. MIKE

The described methodologies are rather heavyweight and not suitable for PKM for two main reasons: (1) In PKM, there is no difference between expert and non-expert, as there is only one person, and (2) the person rarely has the explicit goal of creating any kind of formal knowledge, it is rather an emerging by-product. Therefore, knowledge acquisition processes cannot be re-used directly to PKM.

Mediating Representations A mediating representation (Johnson, 1989) is an interactive representation of a part of knowledge model to a user. In the same way as programming languages abstract away, layer by layer, the bit string that is running inside a computer's processor, a mediating representation abstracts away how data is modelled. A mediating representation works in both ways; it can show data to a user and interpret user interface commands as changes to the data. A mediating representation is the user interface equivalent of a conceptual model.

> "In fact, a spreadsheet could be classified as an effective mediating representation. A spreadsheet provides users with an efficient representation language for building business models. Previously such models were programmed directly in languages such as Basic or COBOL. Many more users can build business models on computers because spreadsheets are available. The emphasis is on business modelling skill, not programming skill. [...]. It can be argued that spreadsheets help improve the quality of business models because users (1) are not burdened with computer programming, (2) concentrate on important aspects of the model directly, (3) can easily change the model, and (4) can more easily maintain the model." – Boose (1990)

According to Lethbridge (1994), "a mediating representation is designed to highlight certain aspects of the knowledge, or to allow the user to perform a certain task with the knowledge."

2.13. Human-computer interaction

Conceptual Model An important concept in human-computer interaction (HCI) is the *conceptual model*. A conceptual model focuses on information abstractions rather than device controls (Jones et al., 2006, p. 66).

A conceptual model is the conceptual representation a human works with. Consider a file-system: From the point of view of a user working with a file system explorer, the conceptual model is a tree in which each tree node (folder) contains files and sub-trees (sub-folders). But from a technical viewpoint, a file system consists of nodes, blocks, node tables and file allocation tables. The conceptual model can be quite different from the technical model.

Input devices Personal computers have been equipped with a keyboard since their first appearance in 1973 (The *Alto* described in Friedewald, 2000, p. 261). Other text entry methods did not become mainstream. Since its invention in 1964 (Friedewald, 2000, p. 181), the mouse became popular as a second input device for graphical user interfaces. Other input devices are joysticks, graphic tablets, touch-sensitive displays, scanners, digital cameras or more advanced controls such as motion-tracking gloves as used in virtual reality environments. Today, out of all these input devices, only keyboard and mouse are present on almost all personal computers. Portable computers often use so called track-points or a touch-pad as a mobile version of the mouse.

Input efficiency for knowledge articulation User interfaces for knowledge articulation often use keyboard and mouse in combination. For many mouse commands an equivalent keyboard shortcut exists. For writing a document or program source code, they keyboard is more efficient than the mouse.

A standard computer keyboard contains over 100 keys, so with each key press a user can transmit willingly over 6 bits, which are bits that carry human-intended semantics. Mouse movement generates technically more bits, i. e., all changes of position and button movement, but far fewer meaningful bits, i. e., selecting a menu command with the mouse takes much more effort than with a keyboard. Thus externalising knowledge via keyboard is faster than via mouse. For externalising ideas, text has a second advantage. Language is the basis of thinking. Language is expressed in words which can be written down in texts.

A mouse is useful for browsing or refactoring structures in a visual way.

3. Analysis and Requirements

This chapter gathers requirements for knowledge models. First typical PKM use cases are described which have been collected from a survey. In Sec. 3.2, existing models for processes in PKM are presented. A new knowledge cue life-cycle is presented which describes the interaction of a user with her PKM tool more detailed. Based on this life-cycle model, Sec. 3.3 conducts an economic analysis for the different steps. Finally Sec. 3.4 presents known requirements from literature. At this stage, it becomes clear that knowledge models should be able to represent generic structured models. Sec. 3.5 analyses popular existing conceptual models for working with external knowledge representation on a computer, i. e., documents, file system, wikis, hypertext, tagging, and semantic web languages, are analysed. From this analysis, the most relevant relations for knowledge models are distilled. Sec. 3.7 gives an overview over all requirements gathered from the other sections in this chapter.

3.1. Use cases in PKM

This section analyses the core used cases in PKM, as collected in an online survey.

Survey To find out about the requirements for PKM tools, 50 participants were interviewed in an online survey. Test subjects were recruited form a number of sources: (a) All 16 participants from case study I (see Sec. 6.2), (b) 5 random visitors of the English PKM page on Wikipedia, and (c) 29 researchers and students at FZI and AIFB which haven been recruited directly. The complete survey can be found in Appendix A.2.

PKM tasks

40 participants out of 50 answered the question "*For what kind of projects/tasks do you wish you had better PKM support?*". Each participant could answer in free text and list any number of tasks. The following tasks were mentioned at least two times:

- Note-taking (21 times).
- Todo-lists and task management (11 times). Linking tasks with other artefacts has been mentioned 3 times.
- Writing, creating a (scientific) document (6 times).
- Project planning and project management (4 times).
- Creating or preparing a presentation (3 times).

- Learning, i. e., any tasks related to studying (3 times).
- Publishing, sharing and exchanging personal knowledge (3 times).
- Collaborative projects or team work (2 times).

Other tasks mentioned are address book management, calendar, documentation of software and processes, organisation of social life, planning of personal future, programming, requirements engineering, shopping, and translation.

Note-taking This task has been mentioned by over 50% of participants. Therefore this task is considered a core-task of PKM. In order not to forget things, people make notes about their ideas (i. e., lab book), their life (i. e., diary), their friends (e. g., address book) and complex subject matters (e. g., learning notes).

Note-taking is also a very general task. Many participants indicated also their most problematic sub-task in note-taking:

- Organisation of notes (6 times)
- Annotation of existing artefacts such as web pages or desktop objects (6 times)
- Maturing notes and later exporting them in a more structured format to other applications (5 times)
- Collecting ideas (3 times)

When collecting ideas and filing notes away, this relieves the mind from keeping track of that idea all the time.

The survey (cf. Sec. A.2) asked also "In one week, how many personal items do you write down?" In average, people create 46 "personal items" per week. The answer field was free-text and some people wrote down not only a number but also explained briefly what they considered a personal item in note-taking: An A4 sheet of paper, a page typeset with LaTeX, a longer email, an EMACS database entry, a line of keywords in a tiny notebook, en entry in todo-list application, a sheet of paper, a short note, or an entry.

Related use cases

A common use case involving note-taking is learning.

Learning Learners often take personal notes to phrase text books into their own language and create more compact, personalised representations. Therefore, learning is considered an application of note-taking.

Similar to PKM systems, there is the notion of *Personal Learning Environments* (PLE) (van Harmelen, 2006). PLEs are often focused on creating a *personal portfolio* which is used to track and plan personal learning progress. A personal portfolio can be shown to supervisors and prospective employers to demonstrate acquired skills. The research communities around personal learning and personal knowledge management study similar phenomena from different viewpoints.

A typical task in learning is "Getting Overview" about a new topic. E. g., if a person wants to learn all about French cheese.

Idea Management This task is also very closely related to note-taking. Many people take notes about their ideas in order to first capture and later filter and refine them. This process of "idea maturing" is also advocated by Schnetzler (2004). A similar, more generic process of "knowledge maturing" is followed in the MATURE project[1].

Sample task

A frequent form of taking personal notes is copying a part form an existing digital source and pasting it into another tool. E. g., copying a part of a web-site or email into a personal wiki. In fact, the most often used command in Microsoft Word 2003 is *paste*[2], which amounts to 11% of all commands used.

Document creation The task "document creation" was mentioned 6 times. One user summarized this task as "[. . .] locate the information by keyword, date, other metadata or by tracing a path of discovery, then attributing the source correctly, and communicating in a universally readable format [. . .]". Esselborn-Krumbiegel (2002) defines the typical document creation process as

- orientation – collecting ideas, defining the topic;
- investigation – evaluating and analyzing material from external sources;
- structuring – creating an outline;
- creating a first draft – containing all parts in linear order but low quality;
- refining – content, language and style.

Creating or preparing a presentation can be considered a related task to document creation. Similarly, publishing, sharing and exchanging personal knowledge requires some kind of codified knowledge. Usually informal personal notes are not enough and contextually richer documents need to be created.

Our society is heavily based on communication by documents, ranging from books to emails. "Almost all current documents have existed in electronic form at one stage in their life" (Pédauque, 2003). Few documents are written directly in a text editor in a linear one-pass fashion. During document creation, the final order is most likely not yet defined. Ordering a collection of ideas or text snippets into a coherent flow is one of the main tasks of authoring (Esselborn-Krumbiegel, 2002) (→ Req. 19 $^{\text{order}}$)[3].

[1] `http://mature-ip.eu/` (accessed 06.01.2010)

[2] `http://blogs.msdn.com/jensenh/archive/2006/04/07/570798.aspx` (accessed 06.01.2010)

[3] The notation "(→ Req. 19 $^{\text{order}}$)" is used to indicate that this is an argument for requirement 19. Each requirement is referenced by number plus and additional short verbalisation.

A user should be able to create order gradually, e. g., by stating order between some sections, but not requiring a total ordering (→ Req. 9 stepwise). Different tools are employed for different degrees and kinds of structures (e. g., notes about the idea, outline, argumentative structure, list of references, quotations from other sources). Knowledge often goes a long way, traversing different media and tools while subsequently approaching its final structure. A typical sample task from this area is the export of a set of personal notes into the format of a specialised outline or mind mapping tool.

Sample Task

Personal knowledge models could speed up the document creation process, by allowing an author to manage her knowledge digitally and refine it step-by-step into a document-like artefact. Then, a part of the personal knowledge model could be exported as a document.

Argumentation Another use-case for PKM, although rarely mentioned in the survey, is structured argumentation.

Sometimes and individual has to take difficult decisions with far-reaching consequences, e. g., whether to accept a job offer in another city or not. Structured argumentation helps also to capture the rationale for a decision, e. g., in software engineering. The foundations for structured argumentation have been published as IBIS by Kunz and Rittel (1970), with the core concepts issue, idea, example, pro- and contra-argument. Later works created software tools based on IBIS, such as gIBIS (Conklin and Begeman, 1988) and Issue Based Argumentation In A Wiki[4] (IBAW).

PIM: tasks, appointments, email, contacts Although PIM (cf. Sec. 2.8) is much more than just managing tasks, appointments, email, and contacts, for most people these are the core information types.

Tasks and Appointments Technically, task management is mostly about managing a set of highly structured objects (the tasks) which can be viewed and manipulated in a set of specialised views (e. g., a calendar, a todo-list, a task-dependency-tree, ...). The same is true of project planning and project management. Taken together, the tasks *managing todo-lists*, *task management* and *project management* have been mentioned by 27% of the survey-participants who answered the question. An introduction to task-management and semantic-based task management is given by Grebner (2009).

Emails Again, Emails are highly structured artefacts with a number of standards, e. g., about header data and content formats. Inside this structured envelope users send plain text, HTML text and files around.

Contacts In most PIM applications, contacts are managed as structured data objects with a number of standard attributes such as street address, work phone, private phone, fax number and email address.

[4] `http://xam.de/2006/02-ibaw.html` (accessed 06.01.2010)

These standard attributes can usually be synchronised to, e. g., mobile phone and other PIM tools.

All three kinds of information objects are highly structured and a number of standards for data exchange exist. Many PIM applications provide sophisticated user interfaces for manipulation of these structured objects but rather poor options for management of personal notes. Often personal notes are just a free-text field attached to the structured information items with no option to add structure or links.

This thesis explores personal notes that can also be used to represent PIM objects. It is expected that prototypical tools for managing personal notes, even if they can be structured and formalised well, do not initially meet the requirements of experienced PIM tool users. They are likely very used to their existing, specialised views. Behind this fact is an economic conflict: Either a tool is generic, with the consequence of the user interface being ill-suited for structured objects. Or a tool is domain-specific, with the consequence of a restricting data-model customized for one kind of structured object. This prevents the tool from being used in a number of situations. For managing personal knowledge cues, flexibility was considered more important than specialised user interfaces.

Personal Social Network Management (PSNM) Social networks are popular on the web and within enterprises. Older yellow pages in enterprises emphasised more the personal competence profile and less the connections to other employees or clients. Existing consumer social networks such as Facebook or MySpace emphasise the social relations a lot but offer still few options to maintain privacy. Few people have only social contacts that are entirely present on a particular web platform. In the end, users are left with managing their social network personally outside of the web platform, e. g., whom they know, where they live and when their birthday is. PSNM remains therefore a PKM task. Data-wise, PSNM needs to manage a mix of highly structured and unstructured data. Structured data concerns the standard attributes of a contact. Less precise knowledge about relations to other people, companies, and places is not covered well by existing standards for contact data. Here more free-form like note-taking comes into play.

In this thesis, PSNM is considered a prime use case motivating the need for a unified management of formalised, structured and less formalised knowledge (→ Req. 8 formality levels). An example for a task involving structured and less structured information is a person that plans a trip to Berlin and would like to meet his friends there. So any person that is sufficiently well known and lives in Berlin or works for a company that is located in Berlin is relevant. Of all these persons, the email-address is required to contact them.

Sample task

Summary One participant summarised this situation as “a PKM tool should help me aggregate, collect and view all the small bits of informa-

tion, which are either needed for long term reference, or in the short term for completing a task."

Note-taking can be seen as a general PKM activity, which is required by other use cases such as learning and idea-management. Personal Social Network Management requires more graph-like structures whereas most parts of PIM can be seen as a combination of note-taking and PSNM. Argumentation can be seen as a prerequisite to document creation. Therefore the most representative use cases are note-taking, document creation and PSNM.

3.2. Processes in PKM

To analyse clearly where a user, doing PKM in general and note-taking in particular, needs what kind of support, this section analyses the PKM processes.

3.2.1. Existing process models

Jones - PIM processes

A very generic, note-taking centric process model is given by Jones and Bruce (2005). See Sec. 2.8 for an explanation about the *Personal Space of Information* (PSI). The basic processes in PIM are:

1. *Keeping*: input of information into a PSI;
2. *Finding or re-finding*: output of information from a PSI, and
3. *Meta-activities*: e. g., mapping between information and need, maintenance and organisation.

Avery - PKM skills

A more detailed model for PKM from Avery, Brooks, Brown, Dorsey, and O'Conner (2001) describes the following seven PKM *skills*:

1. Retrieving information;
2. Evaluating information; (whether it is relevant or not)
3. Organizing information;
4. Collaborating around information;
5. Analyzing information; (what is encoded in the information)
6. Presenting information; and
7. Securing information (maintain privacy, integrity, and availability).

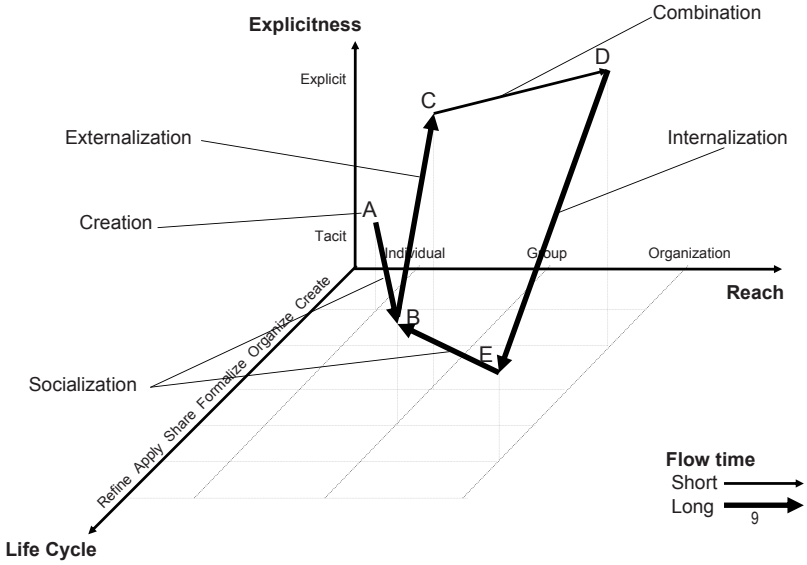
These skills can also be interpreted as processes. Avery et al. (2001) gives an interesting definition of the *knowledge organisation* process:

> Organizing information is a central part of the inquiry process focused on making the connections necessary to link pieces of information. Techniques for organizing information help the inquirer to overcome some of the limitations of the human information processing system. In some ways the key challenge in organizing information is for the inquirer to make the information his or her own through the use of ordering and connecting principles that relate new information to old information. Elementary skills of synthesis and analysis are central to this process. Technological skills in organizing information have become ever more important as electronic tools such as directories and folders, databases, web pages, and web portals provide the inquirer with ever more powerful tools to make connections.

This means that incoming information from others needs to be processed in a way to turn them into personal knowledge cues. He also points out the importance of interaction with tools in this process.

Nissen - knowledge life cycle

Nissen (2005) describes in his *knowledge flow model* (cf. Fig. 3.1) transformations between different forms of knowledge. The model has three axes: *explicitness*, *reach* and *life cycle*. *Explicitness* is either explicit or tacit. *Reach* ranges from individual over group to organisation, nation and global.



A model showing the *flow of knowledge* between groups (axis: reach), different states of explicitness (axis: explicitness), and life cycle steps (axis: life-cycle). This diagram can show transformations of knowledge. The vectors A-E shows Nonaka's SECI model as an example for the expressivity of the model. E. g., vector E represents explicit knowledge within an organisation that has been organized which gets internalized by people in the organisation as it is applied to a task.

Figure 3.1.: Knowledge Flow Model by Marc E. Nissen

The knowledge *life cycle* has the steps

1. Create,
2. Organize,
3. Formalise,
4. Share,
5. Apply, and
6. Refine.

Additionally, knowledge can flow slow or fast between any two points in this parameter space. The figure shows the SECI-model represented within the knowledge flow model.

Barth - PKM processes and tools

A more concrete, very similar process model by Barth (2004) is centred around ideas and includes mentions of tools used:

1. Accessing information and ideas (e. g., desktop search),
2. Evaluating information and ideas (e. g., collaborative filtering),
3. Organizing information and ideas (e. g., diaries, portals),
4. Analyzing information and ideas (e. g., spreadsheets, visualization tools),
5. Conveying information and ideas (e. g., presentations, web sites),
6. Collaborating around information and ideas (e. g., messaging, meeting at the water cooler), and
7. Securing information and ideas (e. g., virus scanner).

North - value creation components in knowledge work

According to North (2007, p. 27–28), core value creation components in knowledge work are:

1. Planning (including organising, strategy development) – Note: Here organising is *not* organising the knowledge, but organising the work.
2. Analyzing (including searching, structuring, and reflecting)
3. Synthesizing (including combination, reconfiguration, and designing)
4. Communication and documentation
5. Learning

Comparison of process models Table 3.1 shows a comparison of the models from Jones, Avery, North and Nissen. Although they are not all PKM process models, they can be compared to a certain degree. The models from North and Nissen are both more focused on the mental processes, whereas Jones, Avery and Barth look more at the interaction of a user with her tools. Therefore the steps *planning* and *knowledge creation* have no correspondence in the tool-centric processes. The same applies to *learning* and the step to *apply* knowledge. On the other hand, only the Jones model has an explicit step representing the codification process to represent knowledge in digital form (*keeping*). There are many *processes* about analysing existing codified knowledge (Avery, Barth, North) and about organising existing codified knowledge (Jones, Avery, Barth, Nissen). However, there are today very few generic, integrated *tools* for analyzing or organizing codified knowledge.

Observations

Jones	Avery/Barth	North	Nissen
Perspective: user & storage	Perspective: user & tools	Perspective: mental process	Perspective: knowledge
—	—	planning	—
—	—	ana., syn.	create
finding	retrieving/accessing	analyzing	—
finding	evaluating	analyzing	—
finding	analyzing	analyzing	—
finding	presenting/conveying	comm.+doc.	share
keeping	—	comm.+doc.	—
keeping, finding	collaborating	*all*	—
meta	organising	ana., syn.	organize
meta	(organising)	ana., syn.	formalise
meta	(organising)	ana., syn., learn	refine
meta	securing	—	—
meta	—	synthesizing	—
—	—	learning	—
—	—	—	apply

How to read this table: Processes are mapped between models. E. g., the process *finding* from Jones maps to four different steps in the model of Avery/Barth, which again maps to the two steps *analyzing* and *communication and documentation* of North. In Nissen's model, not all these steps have a correspondence, only the *comm.+doc.* step from North maps to *share* in Nissen's model.

Table 3.1.: Comparison of PKM process models

3.2.2. Knowledge cue life-cycle

For an analysis of the requirements for a good PKM system based on the notion of knowledge cues, none of the models above is fitting particularly well. A model from the perspective of the individual, focusing on the interactions between people and tools is still missing. Therefore, a knowledge

cue life-cycle model was created (also called Völkel model). Fig. 3.2 shows

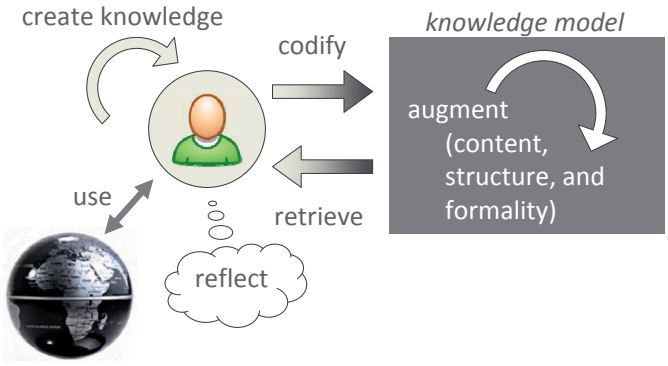


Figure 3.2.: Knowledge cue life-cycle in isolation

the basic processes that happen when a person is doing PKM in isolation. The main processes are (in temporal order):

Create knowledge This is a purely mental process where new knowledge is created by thinking.

Codify Here some experienced mental state is associated with a pattern that is just created in digital form. This process represents the initial creation of knowledge cues.

Augment This is a complex process that involves augmenting an existing knowledge cue by adding more content, adding more structure or adding more formality. Of course, removing content, structure or formality is also possible. This process is described in greater detail further below.

Retrieve In this process the user interacts with her PKM tool to find previously created knowledge cues. Typical ways to retrieve cues is by (full-text) searching, associative browsing and following links – or combinations thereof. Executing semantic queries is another form of retrieval.

Use This process represents the usage of knowledge in a real world situation. The classical process of using knowledge is to act on it, this means to take an informed decision.

Note: The *use*-step is the only step that gives value to a user. All processes that interact with a knowledge model are investments for future cost savings. Exception: Some people *do* take notes solely because the act of writing helps them to remember the content better (Kidd, 1994).

Reflect This process represents any kind of reflection and management of the other knowledge cue processes.

Augment processes There are many ways in which a knowledge cue can stepwise be augmented. One can distinguish to change a single knowledge cue, or to change the structure between multiple knowledge cues. Special operations like splitting and merging knowledge cues change at the same time inner and outer structure of knowledge cues.

Stepwise formalisation and other augmentation processes

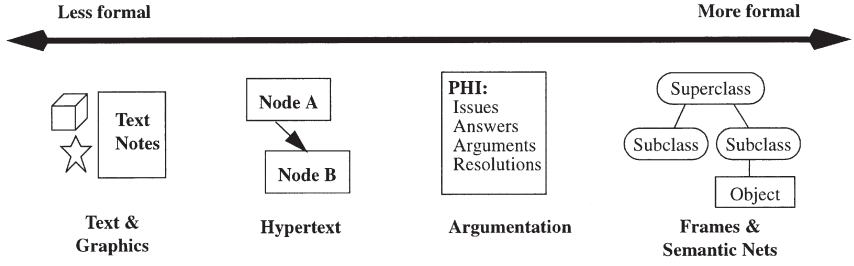


Figure 3.3.: Range of formalisms in domain-oriented systems by Shipman and McCall (1999)

The notion of *incremental formalisation* is used by Shipman and McCall (1999). Fig. 3.3 shows the basic idea of a continuum of formality.

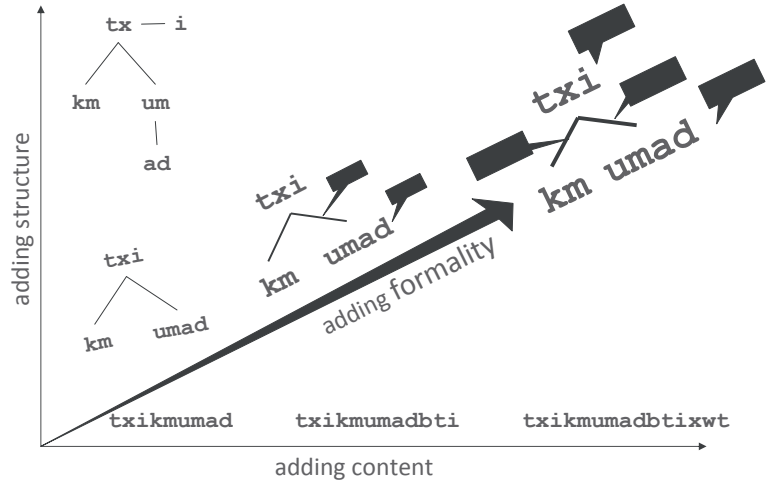


Figure 3.4.: Knowledge cue augmentation processes: A sequence of random characters represents content that is augmented along several dimensions. More content is added, structure is added, or content is formalised – which requires a certain amount of structure. Only addressable items can be typed.

Fig. 3.4 depicts the details of the knowledge cue *augment* process. A random character sequence `txikmumad` serves as an example for an initial knowledge cue. The augment process can now:

Add content This is depicted to the right by extending the characters sequence succinctly with more characters. Deletion or changes of characters are also represented in this process.

Add structure This process keeps the size of the knowledge cue constant, but adds more structure to the content. This process extends the number of addressable entities.

In the example, the character sequence is split up into parts, which are then structured in a tree, just like, e. g., in XML or HTML. In text processors the same effect is achieved by, e. g., by formatting the text as headlines, sections, paragraphs, bold text and italics. Another way to change the structure is to add links between knowledge cues. For a user such processes are, e. g., hyper-linking, tagging, and classifying.

An everyday example of adding structure is to sort digital images into albums. Albums can represent events, locations or other similarities. The semantics of the grouping are not made explicit by merely filing the pictures.

Add formality Only addressable entities can be typed with types from a meta-model. Therefore, this process requires a certain amount of structure to be present. It adds meta-data to structure-data in order to explain the semantics of structures. If structures indicate how items are structured, then formality explains in which way the structure connects items. The meta-data is represented by call-out rectangles. The example shows meta-data added to relations and parts of the knowledge cues. These meta-data annotation can be though of as formal types. Adding formality requires a certain type-system, which is often defined in a meta-model. Meta-model issues are not represented in the picture for clarity reasons. They are discussed in Sec. 2.1. This process covers also the change from a formally typed item or relation to one with another formal type, where the latter is described by more or less axioms. E. g., changing from “has sub-structure” to “has employee”.

A typical examples for adding formality is the classification of mp3 music files with meta-data to describe the genre of songs. If this genre is stored in the genre-field of the standardised ID3 header, other software can, e. g., automatically create a playlist of all classical jazz songs.

Fig. 3.5 shows the extension of the basic processes with collaboration. This is a more realistic case, but looking at the processes in isolation was easier to understand. Knowledge workers not only create and transform knowledge, they also have to communicate knowledge to other parties. For this, they have, e. g., to write documents and reports (Rockley, 2002). This is modelled as *export* as *share*. In the figure, there are four more processes:

Export This process takes a sub-set of the knowledge model and transforms it into a desired target format. Typical target formats are word processor document, email, mind-map or web page. Technically, creating these desired target structures is expensive to implement, since each target format needs a kind of converter. However, each converter is rather easy to create, if the variation in knowledge cues is not too

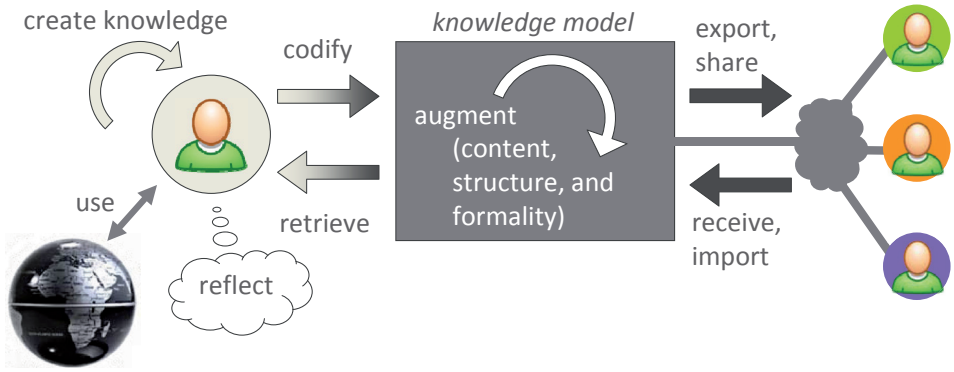


Figure 3.5.: Knowledge cue life-cycle with collaboration

high. Not all features of the target format need to be used; instead, it is enough to generate a simple file via some kind of template-approach.

Share After the knowledge cues have been exported to a desired target format, they can be shared with other users. In a perfect world, the other users would directly use the knowledge cues; hence no transformation would have to be performed. However, a way to control which cues are shared when and with whom is still required.

In addition to use knowledge in the *use* process, knowledge cues can be used directly, if they are sufficiently rich in content, structure and formality. A typical example is the creation of a presentation from an outline. Sometimes one merely has to add a corporate design template to convert personal notes into a publishable presentation.

Receive Some kind of input format is received. Conceptually, the most important step is to decide which parts should be imported and which not. Knowledge cues from other users that have not been reviewed cannot be considered knowledge cues for the receiving user.

Import Here existing formats are converted into knowledge cues. Conceptually, this is easy, if the knowledge cues are sufficiently expressive. Again, one converter is required for each kind of input format. However, different from export, in order to import a source format *every* feature of the source format needs to be understood and converted. Therefore import converters are generally much more costly to implement.

Summary To summarise, the knowledge cue life-cycle has the processes:

1. *Create knowledge* mentally;
2. *Codify* by creating initial knowledge cues;
3. *Augment* knowledge cues by adding more content, structure, or formality;

4. *Export* knowledge cues into other formats;
5. *Share* knowledge cues with others;
6. *Retrieve*[5] knowledge cues and evoke a previously experienced mind-state;
7. *Receive* knowledge cues into other formats and evaluate and filter them;
8. *Import* relevant and reviewed parts as knowledge cues;
9. *Use* knowledge in a real world situation;
10. *Reflect* on all knowledge cue processes.

Example

As a simple example, how these steps can be instantiated, imagine Dirk working for SAP. One day, he has an idea how to rewrite a database query so that it returns the result twice as fast, although it works only for the first ten elements. Dirk has just *created knowledge*. Unfortunately, it does not solve his task at hand, because he needs to return many more results. So he decides to *codify* his knowledge and writes in a mind map. His mind map collection is already very big, so he decides to *augment* his newly created *knowledge cue* (here: the mind map) and stores it in a well-named folder hierarchy. He also formats his thought nicely and writes full sentences, so that he can understand his note much later as well. ... Months later, Dirk has to work on a new project and needs to return just the best three query results from a data base really fast. He scratches his head and searches in his mind map application, trying to remember the tags he might have used. In fact, he is in the middle of a *retrieve* process step. He finds four matching mind maps and reads them. The second mind map tells him how he can rewrite the query! Dirk is just *using* his knowledge cue. After happily implementing that database function, he takes a step back and thinks about what just happened. Now he is in the *reflect* step. It occurs to him that other colleagues might benefit from this knowledge as well. So he *exports* his mind-map as formatted text and pastes it into the SAP wiki. He just *shared* a knowledge cue. Now his colleagues can *receive* this wiki page, read it, and add it to their own personal knowledge model. Depending on the tools they use, they might be able to *import* the existing knowledge cue, so that they don't have to re-create all the structure in their tool.

Comparing the knowledge cue life-cycle with other process models To compare the different models without confusing the terms, each term in the comparison is suffixed with the first author of the model, i. e., "J" for Jones, "A" for Avery, "B" for Barth, "No" for North", "Ni" for Nissen and "V" for Völkel. A graphical representation of the comparison is depicted in Figures 3.6 and 3.7. Dotted lines represent weaker mappings.

Jones

The knowledge cue life-cycle emphasises the processes between user and

[5] This process can also search other people's knowledge models, if their knowledge cues are shared.

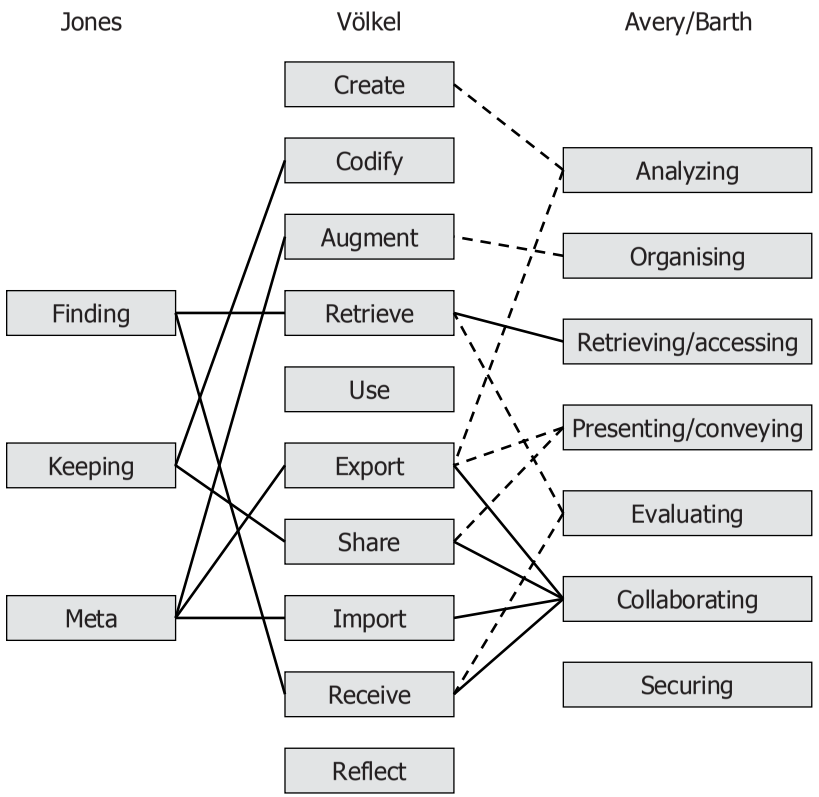


Figure 3.6.: Comparing the knowledge cue life-cycle with models from Jones and Avery/Barth

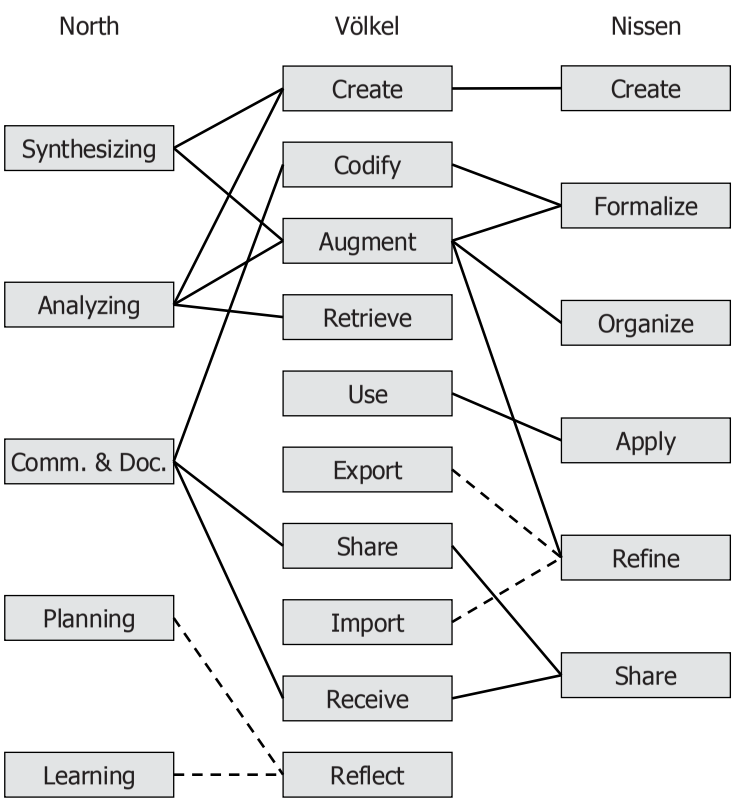


Figure 3.7.: Comparing the knowledge cue life-cycle with models from North and Nissen

tool, therefore it maps well to the process model of Jones and Bruce (2005). $Keeping_J$ is present as two processes, $codify_V$ and $share_V$. From a PIM-point of view, any knowledge that has been received is already in the *personal space of information* (PSI). Therefore, $import_V$ maps best to $meta_J$. Analogous, the process $finding_J$ is represented as $retrieve_V$ and $receive_V$. Again, the $export_V$ process maps better to the $meta_J$ process. The process $augment_V$ maps directly to $meta_J$. Mental processes such as *create knowledge*$_V$, use_V and $reflect_V$ are not present in the Jones model.

Avery

Compared to the PKM skills identified by Avery et al. (2001), $retrieving_A$ maps to $retrieve_V$, $evaluating_A$ to $receive_V$ and $retrieve_V$. $Analyzing_A$ is performed by doing $export_V$ to another tool and *create knowledge*$_V$ by looking at the results. $Presenting_A$ maps partially to $export_V$. $Collaborating_A$ maps to all four collaborative processes in the Völkel model: $export_V$, $share_V$, $receive_V$, and $import_V$. $Organising_A$ maps to $augment_V$, but note that $augment_V$ is much more than just organising knowledge. $Securing_A$ is a cross-cutting concern across the life-cycle, not present in the model of Völkel. Note that Avery has no processes corresponding to *create knowledge*$_V$, use_V and $reflect_V$. Interestingly, not even the basic process $codify_V$ has any correspondence in Avery's model. How information enters the system remains therefore unclear.

Barth

This model is very similar to the one from Avery and the mapping remains the same. The processes from Avery are mapped to those from Barth in Table 3.1.

North

Here $planning_{No}$ can be seen as a part of $reflect_V$. $Analyzing_{No}$ seems to map to many processes, at least to $retrieve_V$, $augment_V$ and *create knowledge*$_V$. The corresponding process $synthesizing_{No}$ maps at least to $augment_V$ and *create knowledge*$_V$. $Codify_V$ maps to *communication and documentation*$_{No}$. The communication aspect maps also to $share_V$ and $receive_V$. $Learning_{No}$ is not really present in the Völkel model, it vaguely maps to $reflect_V$.

Nissen

The model of Nissen has an explicit $apply_{Ni}$ step which maps to use_V. The phases $organise_{Ni}$, $formalise_{Ni}$ and $refine_{Ni}$ all map to $augment_V$. $Create_{Ni}$ maps best to *create knowledge*$_V$, as Nissen is talking about knowledge, not knowledge representations. $Share_{Ni}$ maps easily to $share_V$. The $codify_V$ process is probably also covered by $formalisation_{Ni}$. Note that the Nissen model has no reflection processes. Transformation processes such as $export_V$ and $import_V$ map vaguely to $refine_{Ni}$.

Probst - core processes of OKM

As a contrast to these PKM related models, consider an OKM model: Probst, Raub, and Romhardt (2006) defines OKM with six interrelated core and two accompanying control processes:

1. knowledge identification (reflection about knowledge and knowledge processes in an organisation),
2. knowledge acquisition (by hiring or buying other organisations),
3. knowledge development (developing new capabilities within the organisation),

4. knowledge sharing and distribution (between the members of the organisation),
5. knowledge usage, and
6. knowledge keeping (experiences, information and documents)

The control processes are *knowledge goal definition* and *evaluation of goal achievement*. These eight processes do not map well to the PKM processes. Only knowledge sharing, knowledge usage and knowledge keeping map to PKM processes.

Nonaka

The four processes of Nonaka (1994)'s spiral model can be mapped well to the knowledge life-cycle. Externalisation is $codify_V$, internalisation is $retrieve_V$, combination can be found in $augment_V$, and socialisation does not occur in the knowledge cue life-cycle.

Summary

To sum up, there is no direct mapping from any of the existing processes to the Völkel model. None of the models details the interaction between user and tool and represents purely mental processes as well. Especially the *use* process is important in the next section where an economic analysis of PKM processes is carried out.

PKM vs. OKM in light of the process model

The general difference to OKM approaches is an emphasis of cognitive processes and human-computer interaction, as both are potential cost drivers. High costs often result in tools not being used at all. In PKM, sharing of information is a subordinate goal to encoding and re-using knowledge cues.

Mapping the knowledge cue life-cycle to use cases The three core use cases explained in the last section, namely note taking, document creation and PSNM, map well to the knowledge cue life-cycle.

Note taking. Taking a note is represented as *create knowledge* and *codify*. If the note is later annotated, filed or linked, this happens in the *augment* process. Later, the personal note is *retrieved* and sometimes the knowledge is then *used* in practice.

Document creation. Document creation is a use case where the final goal is the production of a well structured information artefact that codifies knowledge so that the target audience can hopefully understand it. In addition to the processes for note taking, the processes *augment*, *export* and *share* become much more important. In the *augment* process the user combines many snippets, structures them, defines a linear order, creates sections, adds references to literature (linking to other cues) and finally exports the result into a common format, e. g., PDF.

Personal Social Network Management (PSNM). Knowledge creation in PSNM is rather spontaneous, e. g., if one gets to know somebody. Some facts about a person can be *codified* and this external model of the personal social network can subsequently be *augmented*. A practi-

cal example for possible PSN models are sites like Xing[6] or LinkedIn[7]. On these sites, the user can manage a collection of contacts together with a short description and tags. Contacts can be *exported*, e. g., to PIM software. Furthermore, contacts can be *shared* with other users – usually this is a global setting for all contacts of a user. Future PSNMs might offer better ways to *augment* the information about contacts and import such data, especially relations among contacts, from a number of sources.

3.3. Economic analysis

This section analyses costs and benefits of an individual doing PKM from the perspective of a neutral observer. *The results are based on a paper (Völkel and Abecker, 2008) published at ICEIS 2008.*

Look at the complete cycle

Much work has been done in information retrieval (IR) to improve search precision for classical and hypertext documents, only by looking at the data as "given". However, much less research has been done on the whole cycle of authoring *and* retrieval. One of the most important cost drivers is the question how well the costs spent on externalisation and query formulation can improve the precision of the retrieved items.

> "We need to think in terms of investment, allocation of costs and benefits between the organizer and retriever"
> Glushko (2006, p. 24-31).

In PKM, organizer and retriever is the same person. Different from an organisational context, there is thus a personal motivation to organize knowledge. The user can freely trade efforts of authoring with efforts of retrieval.

Related work

Few works address ways to quantify costs and benefits in PKM or at least analyse the determining factors. A related work done by Bontas, Tempich, and Sure (2006) in the area of ontology engineering does not fit, as the use of a personal knowledge model is not a linear, planned process with the goal of creating a formal representation. Rather the contrary is true: An individual is usually reluctant to formalise anything, because it is unclear if the extra effort will ever pay off.

Knowledge work today Today, most knowledge work is assisted with computers. Fig. 3.8 shows a simplified view on knowledge work: People (in circles) interact heavily with their digital information spaces (gray boxes). Their personal spaces are connected, e. g., via internet. There is also a smaller blue arrow indicating face-to-face meetings, but they become more the exception than the norm, compared to the amount of communication carried out over digital media, e. g., as emails, meeting requests, shared documents, wiki pages, PDFs, social networking tools, ...

[6] `http://www.xing.com` (accessed 06.01.2010)
[7] `http://www.linkedin.com` (accessed 06.01.2010)

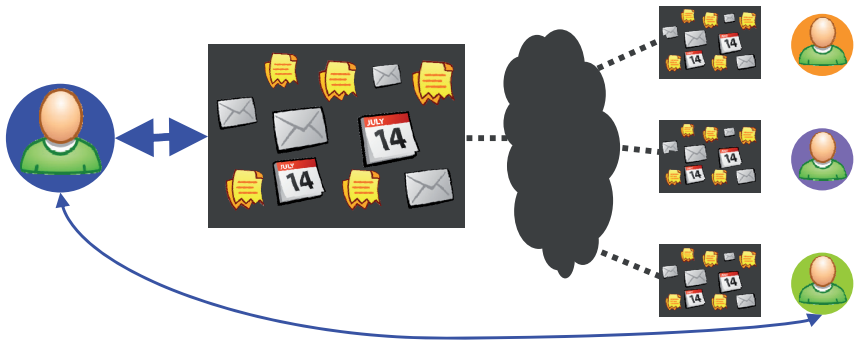


Figure 3.8.: Knowledge work today

Pollard (2005) sees a trend from central content management to personal content management, shared in a peer-to-peer fashion. Instead of publishing one document for a broad audience, a knowledge worker today has to publish many documents and reports, targeted for small groups or even single persons. Thus re-use and delivering of knowledge to different audiences becomes important. Thus, a knowledge model export should be cheap.

Communication costs

PKM can be seen as the process of sending oneself messages to the future. As a metaphor, each note is like a message in a bottle which is thrown into the flow of time. At any moment in time, our live is flooded by such bottles arriving – paper piles everywhere and a hard disk full of reminders. We forgot the content of the messages – otherwise we wouldn't need to open the bottles. At every moment, we have to decide which bottles to open and which to ignore. We can imagine the bottles having different colours or labels – this is the filing system we used when we created the message. Unfortunately, our filing system changes over time.

Message in a bottle

To apply Nonaka's model to PKM, a person in time can be seen as different entities. Intuitively, a person five years ago (P_{-5}) is not the same person as today (P_0) and will be different from that person in five years (P_{+5}). The processes externalisation, internalisation and combination stay unchanged. Socialisation becomes something completely different. The entities P_{-5}, P_0, and P_{+5} can never meet. But each entity P_x can *remember* knowledge created or internalised from earlier entities P_{x-t}.

Interaction bottleneck Fig. 3.9 shows the main PKM situation: On the left is a person adding digital content to a personal space of information. Later, depicted on the right, the same person is retrieving information again. In collaborative knowledge work (Fig. 3.8) and even more so in personal knowledge management (Fig. 3.9), the interface between human and tool is a strong cost influence factor.

Technological developments, like written language, the printing press and finally the internet, have lowered the costs and accelerated the process of information distribution many orders of magnitude.

Bottleneck authoring

Full-text retrieval on global scale (i. e., web search engine) and local scale (i. e., desktop search engine) takes less than a second. Reformatting a doc-

Figure 3.9.: Interaction bottleneck

ument or presentation according to pre-defined styles takes seconds. Recalculating a complex spreadsheet with many formulas is carried out quickly as well. What takes time is to write, read and file documents; to create and arrange documents and presentations; to create, debug and comprehend spreadsheets. As information systems have become really fast, the human-computer interaction became a bottleneck of efficient knowledge work.

Keyboard vs. mouse

On a standard PC, text is much faster to create than diagrams. By means of information theory pressing a single key on a standard computer keyboard with 102 keys sends $\sim 6,6$ bits of information, whereas moving the mouse to one of 102 defined locations on a screen and clicking there requires more time and haptical effort.

Visual representations

Visual representations play an important role in architecture, engineering, and research as "artefacts of knowing" (Ewenstein and Whyte, 2007). However, the most common representation type used in these disciplines are not purely pictures, notation or writing but a *hybrid* between pictorial, geometric and scripted elements (Elkins, 1999). They can be seen as semi-formal models of a domain, like, e. g., UML diagrams.

A study on 28 people in meetings using pen-computers (Khan, 1994) found out that 50% only copied diagrams from white-boards, 17% used diagrams very occasionally, and 33% did not draw diagrams at all. Hence, even if people have the hardware to create diagrams, they seem not to be the dominant form of expressing ones own thoughts. Therefore this thesis focuses on (hyper-)textual representations. A visual tool based on CDS, iMapping[8], is described in Sec. 5.3.1.

3.3.1. Costs and benefits without tools

First costs and benefits are analysed for the case that *no tools* are used at all. The next section (3.3.2) analyses costs and benefits *with tools*.

If no external tools are used, the PKM process consists of

- *Knowledge creation* at a cost C_C and
- *Knowledge use* with a benefit B.

[8] iMapping is one of the main parts of a thesis-in-progress by Heiko Haller at the FZI Forschungszentrum Informatik Karlsruhe.

The **cost of knowledge creation** C_C is proportional to the amount of time and personal (e. g., cognitive and motivational) effort an individual spends on thinking, researching, experimenting, and learning. These internal processes for knowledge creation can be measured by the amount of time spent. But how to measure the value?

Value of knowledge

Maybe the time needed to create the knowledge? But if it takes long to create some knowledge that has a low benefit, then one would intuitively not assign a high value to it. Also knowledge might be cheap to create and store now, but much more costly to re-create later, e. g., *Where have I been on a given day in 1999?* The opposite might also be true: Some knowledge is difficult to create today, but common knowledge later. In any case, the *value* of knowledge is certainly unrelated to the *cost* of creating it.

The **value of knowledge** does not exist as such (Iske and Boekhoff, 2002); it depends highly on the task. The value of some knowledge can be defined as "increment in expected utility resulting from an improved choice" made possible by this knowledge (Varian, 1999). This is the value of decisions taken on the basis of the knowledge re-evoked by the knowledge cue, i. e., one estimates the value of the state of the world that would result from actions performed in absence of the knowledge (V_1) and compares it with the value of the state of the world resulting from actions taking in presence of the knowledge (V_2). Then the benefit B of having this knowledge for this task is the difference in value, i. e., $B = V_2 - V_1$. B then represents the additional created value or saved costs. B can (in theory) be measured in money, saved time, improved quality or better emotions. In practice, measuring the value of the state of the world is often hard to quantify, i. e., consider long-term effects and the difficulty to measure, e. g., happiness. In the end, value of knowledge is often a subjective quantity, because the value of "the state of the world" is often subjective. If the task was the creation of an information artefact, the price paid by others can be an indicator of its value. Most approaches resort to measure only costs (Feldman, Duhl, Marobella, and Crawford, 2005). Economists use an abstract "utility value" without defining a unit of measurement.

3.3.2. Costs and benefits with tools

If tools are used, there are in practice often costs for the availability or usage of the tools themselves as well, e. g., license costs for software tools. However, theses costs usually play a minor role in the total costs of PKM and are hence ignored.

For a long time in history, knowledge cues were only stored in *documents*, first analog then digital. Digital document retrieval is the prime application domain of information retrieval (IR) techniques. The field of IR has a history of quantitative research, mostly focusing on *precision* and *recall*. These measures were first defined by Cleverdon, Mills, and Keen (1966). However, Cleverdon et al. (1966) proposed to measure not only these two factors but also "the extent to which the system includes relevant matter", and "the effort involved on the part of the user in obtaining answers to

his search requests". Maybe because these two factors cannot so easily be measured automatically, they were mostly ignored by IR research.

The TREC conferences have heavily influenced IR research. In the last ten years, it ran the novelty track, in which each sentence of a document was regarded as an information item on its own. The question answering track goes in a similar direction: here the user is not querying for a set of documents but for a concise, factual answer to his question. Both tracks show a tendency towards smaller content granularity. A fact given within a document requires on average to read half of the document until the fact is found. The *granularity* of information thus influences its retrieval costs. The coarse granularity of documents leads to high costs for locating relevant parts of documents, e. g., for re-use, aggregate queries, or question answering. *Data bases* and ontologies allow for more structured access, i. e., browsing and searching. In contrast to documents, data bases and ontologies allow retrieving sets of items together with relevant properties. Ontologies (and some database systems as well) allow answering queries to which the answer has only implicitly been entered. It requires effort to structure and formalise knowledge to make it fit into a database or ontology, but retrieval abilities are also higher – ontologies are built for re-using knowledge.

Cost-benefit ratio changes user behaviour

Granularity, degree of structure and degree of formality all influence costs of codification and retrieval. Bederson (2004) reports on users of his NoteLens tool: "Perhaps the most interesting observation is that it [the NoteLens tool] has changed the cost structure of information access. It used to be that finding notes was relatively slow, but editing notes was relatively fast, and so I would create fewer longer notes. Now, finding notes has become relatively fast, and I have started creating more, shorter notes. In fact, I even edited several long notes and broke them up into shorter pieces so I could access each individually".

The individual processes of the knowledge cue life-cycle (Sec. 3.2.2), have the following costs and benefits:

1. *Knowledge creation* at a cost C_C.

2. *Codify:* Some parts of the implicit knowledge become external. Knowledge cues are created. These user has externalisation costs C_E.

3. *Augment:* The user invest more time and augments the knowledge cues so that they encode knowledge more explicit. This leads to further externalisation costs C_E.

4. Time passes by and the author might start to forget some or all details of the articulated knowledge. Sometimes even the knowledge to know something is forgotten as well.

5. *Retrieve:* At a certain moment, while performing a certain task, the user initiates a retrieval process in his PKM system.

   Note: As information retrieval systems have become faster, the classic information retrieval measure of "time to execute query" becomes less

relevant to determine the costs perceived by the user. The human-computer interaction becomes more often the bottleneck and cost-driver of efficient knowledge work.

After having executed a query or performed a browsing step, the user reads the search results, and refines the search query. After some steps, the user either found one or several matching knowledge cues or cancels the search with no result. The process of reading through a list of search results takes time and therefore adds to the search costs.

If a knowledge cue is long in size, it takes longer to read through it. If the desired knowledge is codified by a part of a cue, reading through the complete knowledge cue is thus additional search cost. All these costs are subsumed under retrieval costs C_R.

6. *Knowledge use:* If results were found, the user has some benefit B from remembering knowledge from the knowledge cue. If the task was to create some kind of information artefact, the codified knowledge might be used as the basis for this artefact.

People estimate costs and benefit

Esser (1998) analyses factors that determine when and which external memory humans use. Three variables were observed: Expected likelihood of successful remembering a piece of knowledge when stored in an external store, cost of storing it there and most importantly: perceived value of the knowledge to be stored. The higher the perceived importance of remembering the knowledge, the higher costs for storage were accepted.

The basic hopes of any sane person doing PKM are:

- The benefit of re-gaining the knowledge is greater than the management costs for its knowledge cues, i. e., $C_E + C_R < B$
- The management costs are smaller than the costs to re-create the knowledge from scratch, i. e., $C_E + C_R < C_C$.

A person not believing in these two assumptions is wilfully wasting effort. Estimating the benefit of storing an item for later use ($C_E + C_R$) compared with the expected costs to re-generate the contained knowledge later (C_C) is certainly hard.

Comparing PKM systems

For comparing different PKM systems one needs a pragmatic way to take the benefit into account. The value of a certain piece of knowledge, with respect to a task, can hopefully be estimated by individuals on a simple Likert scale (Likert, 1932), e. g., a rating from one to five.

The added value of using a knowledge management system for a given period of time (t) is the overall cost-benefit gain G which can be approximated by summing up all benefits B and subtracting the sum of all costs C ($G = B - C$). For a given amount of knowledge one can compare the costs of different PKM systems.

Counting example

Assuming a user has to solve a certain problem 10 times. There are different scenarios of distributing costs:

Remember The lucky case: On the first task instance, the user creates the required knowledge and for the next nine occurrences she simply remembers the knowledge. Total costs are $1C_C$.

Re-think A common case: For each of the 10 task instances, the user has to re-create the required knowledge. This can happen if the knowledge is complex, hard to remember or the time-span between the task instances is long. Total costs: $10C_C$.

Re-use This scenario assumes a PKM tool. On the first task instance the user creates the required knowledge and codifies it (C_E). For the next nine task instances the user searches, finds the knowledge cue and gets back the knowledge in memory (retrieval costs: C_R). Total costs: $C_C + C_E + 9C_R$.

Not finding The same scenario as *re-use*, but this time assuming the user cannot find, what she is looking for. Total costs are: $10C_C + C_E + 9C_R$.

Not searching The same scenario as *re-use*, but this time assuming the user even forgot that a note has been taken. The total costs are: $10C_C + C_E$.

Over-investing The same scenario as *re-use*, but this time assuming the user spends ten times more effort into codifying the knowledge. Assuming the knowledge in question is very simple and in this case the retrieval costs do not go down, the total costs are $1C_C + 10C_E + 10C_R$.

These examples show already that the strategies and memory abilities of a PKM tool user heavily influence the costs.

3.3.3. Detailed analysis

Given the following basic factors for costs and benefits:

- Each knowledge cue x that is externalised has externalisation costs $C_E(x)$. These costs are detailed in the next section.
- For each task $t \in T$ (with T being the set of all task encountered in the measurement) the user has the option to search for knowledge cues. This has retrieval costs $C_R(t)$. Note that a user might retrieve an item several times or not at all.
- Retrieved items have a benefit $B(t)$ for the given task t.

The overall process has thus the following gain:

$$\begin{aligned} G = & \quad \textstyle\sum_t B(t) - (\sum_x C_E(x) + \sum_t C_R(t)) \\ = & \quad \textstyle\sum_t (B(t) - C_R(t)) - \sum_x C_E(x) \end{aligned}$$

Simplified knowledge model

For the purpose of the economic analysis, a knowledge cue is assumed to contain a single term up to a single sentence. The smallest units of content that make sense to a human are single or multiple words (encoding concepts) or sentences (encoding facts or questions). To be able to count the amount of structure and formality in a knowledge model, no further structure is allowed *within* the knowledge cue. Given a set of symbols, i. e., words, each knowledge cue c can be seen as a simple linear stream of symbols devoid further machine processable structure. Aspects of natural language processing are ignored, to measure only *non-ambiguously explicitly* structured knowledge. Structure and formal statements in the knowledge model are represented as relations between knowledge cues. A formal statement is a triple (c_1, c_2, c_3). The second knowledge cue denotes the semantics of the relation. E. g., a knowledge cue can represent an "is a hyperlink to"-relation, or a "has super-concept"-relation.

Note that this restricted and simple model is only used for the economic analysis. The knowledge model detailed in Sec. 4.1 explicitly models structure within knowledge cues, too. Mappings to documents and ontologies are also covered in that section.

Future PKM systems are expected to allow modelling textual and semantic content in the same environment, as described by Bettoni, Ottiger, Todesco, and Zwimpfer (1998); Ludwig (2005); Oren, Völkel, Breslin, and Decker (2006).

Externalisation costs

C_E can be divided into cost of initially creating the knowledge cue (C_C) and costs of augmenting existing knowledge cues, e. g., classifying new or existing cues or linking between cues (C_A). Linking cues can also be an act of formalisation if the relation is specified with a relation that has a formal semantics. Hence $C_E = C_C + C_A$.

Granularity

Let N be the set of all knowledge cues in the system. Cost of content externalisation is correlated to the size of externalised artefacts. E. g., writing more words takes more time. Let $|n_j|$ be the size of the jth cue, measured in the number of symbols it contains. Note: The smallest symbol size in a text-based environment is a single word or term. Smaller entities are usually not acting as knowledge cues on their own. Articulating a single symbol costs c_s. Articulating the jth cue costs $|n_j|c_s$. Then $C_C = \sum_j |n_j|c_s$.

Augmentation costs

In general, augmentation is the process of adding content, structure, and formality in and between knowledge cues. For an economic analysis, augmentation is restricted to operations between knowledge cues, e. g., linking, tagging, typing and categorising cues. The costs of augmenting are independent of the cues size. The more cues a knowledge base contains, the more effort it might take to find the right cue to link another cue to. Structuring is expected to make more of the knowledge accessible to the computer, which should enable to answer queries with better (more) results. Note: The structure of a knowledge models contains itself knowledge. It is not possible to specify the structuring costs per se, but the degree of formality of a knowledge model can be expected to correlate with the spent structur-

ing costs. The degree of formality d_f can be measured by the amount of formal statements ($|A|$) compared to the total number of knowledge cues, similar to the definitions given by Lethbridge (1998) as $d_f = \frac{|A|}{|N|}$. Assuming a fixed cost c_f for articulating a formal statement[9], one can estimate $C_A = |A|c_f$. This leads to the total externalisation costs

$$C_E = C_C + C_A = \sum_j |n_j|c_s + |A|c_f$$

This equation assumes that no content and no formal statement is ever deleted or changed. But in reality, the cues need to be maintained to keep or improve their value over time. E. g., some knowledge is no longer applicable or needs to be updated to reflect changes in the world. Informal ideas undergo several transitions until some of them might become text books (Maier and Schmidt, 2007).

Some structuring operations could as a side effect increase (split) or decrease (merge) the number of cues. Changes to the number of cues are ignored for sake of simplicity. Deleting outdated or erroneous knowledge could improve the value of using the knowledge model quite a lot, but some costs do occur for these maintenance tasks, too. Therefore instead of measuring the knowledge model as such, one needs to measure the costs of the operations that lead to the current state, i. e., all operations performed.

Costs of operations

Let c be a function that assigns to each operation some costs. Basic operations on a model are:

add content **(**$content_a$**)** Adding m symbols to a knowledge cue costs $m \times c(c_{sa})$ with c_{sa} being the costs of adding one symbol.

delete content **(**$content_d$**)** Deleting m symbols from a knowledge cue costs $m \times c_{ds}$ with c_{ds} being the cost of deleting a symbol. Deleting has often lower cost than adding, e. g., when deleting a cue in a user interface usually causes deletion of many contained symbols and some connected cues.

Cost of updating can be modelled as the sum of deletion costs and addition costs.

add formal statement **(**$stmt_a$**)** Adding a formal statement. The cognitive costs are expected to vary according to the impact of the formal statement. E. g., it should take less time for a person to create a *hasPart* or *knows* statement than a *hasSubclass* statement, because a *hasSubclass* statement has many more implications that the creator needs to check mentally. However, these differences in costs are ignored for sake of simplicity.

delete formal statement **(**$stmt_d$**)** Deleting a single statement could have dramatic effects, depending on the used inference rules.

[9] This assumption allows to evaluate a semantic modelling tool, but not to state anything about the knowledge domain.

To take the restructuring operations into account, $n(content_a)$ is defined as the total number of added symbols – and respectively $n(content_d)$ as the number of deleted symbols.

For each kind of operation (op), the costs can be calculated by multiplying the number of times this operation is performed (n_{op}) with the costs of this operation (c_{op}). The complete externalisation costs are

$$C_E = n_{content_a} c_{content_a} + n_{content_d} c_{content_d} + n_{stmt_a} c_{stmt_a} + n_{stmt_d} c_{stmt_d}$$

The next section analyses the retrieval costs, before the complete cost model can be stated.

Retrieval costs

In order to precise the relation of structures in the knowledge base and search costs one first needs to develop a unified model for the search process in knowledge models. A first work in this direction is the "information foraging process" (Pirolli and Card, 1995).

There are three basic ways to retrieve information when interacting with an information system (Bates, 2002):

Following links should not be confused with browsing. A common practice in large search spaces for which neither suitable collections nor query terms are known is to explore links, e. g., citation links. Following links is thus a kind of associative retrieval.

Browsing a collection of items related to the information need[10]. In principle, two kinds of collections are possible: explicit, i. e., created by a user, and implicit, i. e., the members in the set are determined by a (semantic) query. Toms (2000) and Teevan, Alvarado, Ackerman, and Karger (2004) emphasise the importance of finding information "by accident", e. g., when searching for something else. Such (re-)findings are important for creative processes and knowledge creation.

Formally, browsing is the act of scanning a list of items and evaluating each of them for relevance. Evaluating a single item has the costs C_{ev}. The user is free to stop evaluating items from the list at any time.

Searching denotes the process of executing a query (e. g., keywords) and refining it until the top results are relevant to the information need. Semantic queries, utilising knowledge indirectly for inferencing, also fit into this category.

Formulating a query has the costs q. Systems that allow several kinds of queries need different values for q to model the difference in cost. After each search-step the user is confronted with a list of search results which need to be evaluated, similar to browsing.

The search costs depend on the complexity of the query and the structure of the knowledge base. A complex query has a higher cost to be

[10] Note that this is the definition of browsing from Bates (2002) as one of the three search strategies, not browsing of web pages.

formulated, but has the ability to return exactly the required cue. Simpler queries return usually too many results and need refinement. From a user's perspective, starting with simpler queries that are gradually refined is more economic than asking directly a complex query. The interactive refinement process gives earlier feedback about how many results are returned, which guides query refinement until the query is complex enough to filter out the desired cues. This way, queries do not become more complex than needed.

A complete search process involves typically all three kinds of operations. Instead of measuring each step (query, follow a link, browse, follow another link, query again, ...), the complete retrieval process is modelled as a process that involves some costs $C_R(t)$. These costs can be broken down to $C_{ql}(t)$ for formulating queries and following links and costs for evaluating items C_{ev}.

$k(t)$, p_t

Assuming the user evaluates $k(t)$ items in the retrieval task t. As for any retrieval task, the measures precision and recall can be used (Van Rijsbergen, 1979). Each of the t retrieval tasks has its own, task-specific precision p_t and recall r_t, since different retrieval strategies might be used. As a refinement of precision and recall, a relevant item can be said to have a certain *benefit* for the given task. This value is expressed in a normalised way ranging from zero (no benefit) to one (knowledge highly relevant for the task). The benefit of an item j for a given task t is defined to be $v_j(t)$. For irrelevant items, $v_j(t)$ is zero.

$v_j(t)$

Let $k(t)$ be the total number of all retrieved items in the search process. There is a certain cost C_{ev} to evaluate each item in order to be able to decide if the knowledge represented in the item is relevant for the task. The effort of evaluating a single item is assumed to be independent of precision and recall of the complete process. The complete costs of the retrieval process for task t are then $C_R(t) = C_{ql}(t) + k(t)C_{ev}$.

Only retrieved knowledge cues can bring benefit for the user. Knowledge that is never used is of zero value. The complete benefit $B(t)$ of the $k(t)$ retrieved items is $B(t) = \sum_j v_j(t)$. Both p_t or $k(t)$ might also be zero. Assuming all retrieved items in a task are either relevant (value = 1) or not (value = 0), the formula can be simplified as $B(t) = p_t k(t)$. For each retrieval task t:

$$\begin{aligned} G(t) = & \quad B(t) - C_R(t) \\ = & \quad p_t k(t) - C_{ql}(t) + k(t)C_{ev} \\ = & \quad k(t)(p_t - C_{ev}) - C_{ql}(t) \end{aligned}$$

Thus the query formulation costs C_{ql} are a kind of fixed costs: Regardless of retrieved knowledge cues and regardless of their value, query formulation costs have been spent. The relation between precision and evaluation costs decides if the whole retrieval process was worth the hassle. Precision is typically heavily dependent on the degree of structuredness and formality of the data.

Analysing the equation highlights three factors than can negatively influence the gain of using a PKM system:

1. If the user does not try to retrieve knowledge for a task t;
2. if no or too few relevant items are retrieved, i. e., if recall or precision are too low;
3. if the value of the successfully retrieved items is too low, i. e., results fit, but of too low value.

Addressing factor (1) of the factors than can negatively influence the gain of using a PKM system: If users do not consult their knowledge model, they cannot gain any value from knowledge cues stored therein. Therefore, a PKM system should run queries automatically ($\rightarrow$ Req. 1 $^{\text{auto-query}}$). This is stated as a requirement in a structured format:[11]

Requirement 1: System should run queries automatically Requirement 1

Motivation:Cutrell, Dumais, and Teevan (2006) propose to automatically start a search when certain triggers are encountered. Knowledge cues relevant to the current task context should be delivered pro-actively (Schmidt, 2009). Furthermore, a good system should bring up knowledge cues relevant to the current business process (Abecker, Bernardi, Ntioudis, Mentzas, Herterich, Houy, Müller, and Legal, 2001).

This requirement depends on requirements: 26 $^{\text{queries}}$
Effect on knowledge cue life-cycle: Exploit value of knowledge model more

Continuing with the factors that can negatively influence the gain of using a PKM system: Factor (2) is addressed by works in information retrieval and improved search algorithms.[12] Note that knowledge articulation and modelling have no annual competitions.

Factor (3) can perhaps only be addressed by personal experience or training. Note that if the value of the best-matching content objects is lower than the cost to find and use them, it would be better to tell the user that there are no results instead of bothering him with too irrelevant results. Unfortunately, a PKM tool cannot know how valuable a knowledge cue is for a specific task. The reason for results with too low value can either be that not enough effort went into creation of knowledge cues and only very few

[11] This is the first of a long list of requirements and it is time to introduce the structured format in which requirements are stated in this thesis.

The title *is* the requirement. The following text introduces, explains and motivates the requirement from analysis and known literature. Requirements are referenced with a number together with a short name. Each requirement appears also as a margin note to allow locating them easier. A requirement descriptions ends with a table containing a list of requirements on which this requirement depends, and a short description that explains which process in the knowledge cue life-cycle is relevant and how this process changes.

Section 3.7 gives a summary on all requirements.

[12] See, e. g., proceedings of Text REtrieval Conference (TREC), `http://trec.nist.gov/` (accessed 06.01.2010). TREC started in 1992 and is today *the* annual conference and competition on text retrieval. Recent conferences included tracks on question answering.

of the formerly present mental state of knowing is retrieved. It can also be the case that the task requires only rather trivial knowledge and the user would have a better economic result if she would not have taken notes in the first place. A PKM tool could actively ask the user for feedback on search results, collect data about knowledge cue usage (time spent for creation and augmentation, number of times appearing in search results, number of times selected), and help to train better economic behaviour.

3.3.4. Summary and conclusions

The complete cost function

Stitching the parts for externalisation and retrieval together yields to:

$$\begin{aligned} G = & \quad \sum_t (B(t) - C_R(t)) - \sum_x C_E(x) \\ = & \quad \sum_t k(t)(p_t - C_{ev}) - C_{ql}(t) \\ - & \quad n_{content_a} c_{content_a} - n_{content_d} c_{content_d} - n_{stmt_a} c_{stmt_a} - n_{stmt_d} c_{stmt_d} \end{aligned}$$

The overall benefit of using a PKM system can be characterised by summarizing over the successfully retrieved knowledge items (content or formal statements) for each task. Costs can be characterised as the sum of the costs of all authoring and structuring efforts.

These formulas can serve as a conceptual framework for tool-specific measurements. The economic analysis of the different phases in the knowledge cue life cycle (authoring, retrieval) can also serve as a guideline for evaluating PKM tools.

Cost gap

Users have currently the choice to formalise too little (e. g., only text), resulting in too high retrieval costs. Simple keyword search has a number of drawbacks: One can only retrieve document snippets when remembering words occurring in the text. If many snippets contain the keyword, one has to read long results list, guess which snippets might be the right one and then read all relevant snippets in order to find the right knowledge cue. Smarter keyword search can also support synonyms, but in these cases the additional costs of maintaining synonym lists have to be taken into account, too. Special terms are most likely not to be found in general language resources such as Wordnet (Fellbaum, 1998).

Or they have to formalise too much (e. g., form-driven database applications), leading to unacceptably high articulation costs. The sweet spot can often be in this *cost gap*. No tool can solve this "decision under uncertainty" of estimating costs and value. In classical KM one would perform a feasibility study to evaluate what the right degree of formalisation is – in PKM such decisions have to be taken on a minute-by-minute basis. To reduce the cost gap, a PKM tool should allow modelling on all levels of formality (→ Req. 8 formality levels).

Sweet spot

[bh] As explained above, the total costs for managing knowledge cues are influenced strongly by costs for externalisation (codify, augment) and internalisation (retrieval).

Assuming a PKM tool that allows managing knowledge cues in different degrees of structure and formalisation. The more effort a user puts into

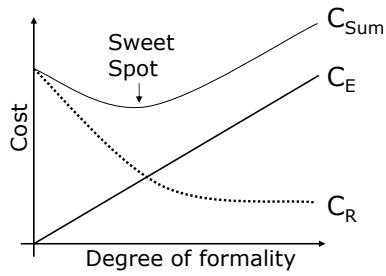


Figure 3.10.: Sweet spot of lowest total costs

creating knowledge cues, the more structure and more formal statements will be present in the knowledge model. Hence, the overall externalisation costs are proportional to the degree of structuredness and formality in the knowledge base.

More structure and formality lead to better, cheaper retrieval. The more structured such information, the easier it can be accessed. A highly structured information collection is easier to navigate, yields more accurate search results, and has more export options to other formats. However, there is always a minimal cost for retrieval, namely query formulation and result set evaluation – regardless if the query was a full-text query, a formal query or a set of browsing steps. Hence, the degree of formality is inverse proportional to retrieval costs, but with diminishing returns.

Optimal degree of formality

Taking these arguments together, there must be a *sweet spot*, i. e., a local minimum for the total costs of managing a particular knowledge cue ($C = C_E + C_R$). In other words, depending on the frequency of later usage, there is an *optimal degree of formality* for a knowledge cue. Unfortunately, the optimal degree can only be *estimated* by a user, because the future usage of the knowledge cue is not determined yet. Of course, the closer a future task is, the easier it is to estimate the knowledge required for that task.

To let users decide on the optimal balance of effort and benefit, PKM requires the uniform management of unstructured, structured and formal knowledge (→ Req. 8 [formality levels]).

Practical cost factors hindering knowledge re-use One obvious way to make knowledge workers more productive is by *re-using as much knowledge they have created as possible*. However, there are several practical cost factors that hinder re-use of knowledge:

Fragmented applications instead of unified management of all knowledge cues. To unify the diverse range of existing artefacts, a knowledge model should deal with a mix of text, structured text, binary artefacts and relations of varying degree of formality (→ Req. 8 [formality levels]). Semantic desktop concepts tackle the fragmentation by re-uniting data on the meta-data level (cf. Sec. 2.6).

Loss of structure and formality in communication. Bradshaw, Light, and Eichmann (2006) argue that researchers would understand documents much quicker if they had access to annotations made by others. Today, often simple documents are exchanged. If instead well-structured knowledge models including the annotations and the content would be exchanged, there would be less loss in the conversion processes.

Knowledge maturing in a tool zoo. A mind-map might be a good way to capture unstructured thoughts. A tagging approach can be used to group a number of, e. g., bibliographic references together. These two kinds of tools usually do not work well together. Often the items tagged in one system cannot be viewed in another system. The physical PDF files of the research papers would probably be stored in the file system, requiring the user to give, e. g., each file a unique name and a single location in the folder hierarchy. A hypertext system, e. g., a wiki, allows creating arbitrary links between pages, but allows no structured queries.

Each of the presented tools or formalism is helpful in other steps of the knowledge maturing process (Maier and Schmidt, 2007). For each step, the user has to switch tools and is confronted with a new conceptual model, offering new capabilities but taking away others.

Loss of structure and formality in transformations. Knowledge workers often loose information when they convert knowledge cues modelled in one tool to another one. E. g., when converting a mind map to a presentation the colour of the mind map items is lost. A similar loss of structure occurs for most transformations, as the data models of the applications differ. This loss of structure degrades abilities for retrieval.

An example for loss of formality is the conversion of a bulleted list to a string of characters with bullet symbols and line breaks. The visual result is very similar, but the semantics of which item is a bullet point is lost. E. g., the list item indentation level can no longer be manipulated automatically. To minimise this loss, the knowledge model should be able to represent most used PKM formats (→ Req. 2 $^{\text{super-set}}$).

Cost of interoperability. Specialised tools have lower costs for data entry and manipulation. Hence a PKM tool could lower the total costs of codification and augmentation if export and import are easy to use and loose no or few of the codified knowledge. E. g., the costs of transforming explicit knowledge encoded in one kind of formalism into another one, such as from a spreadsheet to a presentation, have to be taken into account. It should be possible to re-use structures appearing in structured text, i. e., by later stating the implied semantics of the structures.

The economic analysis clearly points out the need for a unified management of knowledge cues, which can differ in size, degree of structuredness and

degree for formality. Given further the need to be able to import structures from existing PKM tools (or any tool used for PKM tasks), the second requirement can be stated:

Requirement 2

Requirement 2: Knowledge model should be a super-set of existing conceptual models

Transformation costs

Motivation: To re-use content residing in one kind of representation in another tool, it needs to be transformed. Transformations between data-models come not for free. A naive approach to convert between n formalisms would require writing n^2 transformations. However, if a common intermediate formalism can be used, the costs come down to $2n$. To save costs in content transformation, the conceptual knowledge model should therefore be a *super-set of the conceptual models of all other relevant PKM tools*. Sec. 3.5 analyses existing formalisms and collects requirements for a common super-set.

This requirement depends on requirements: –
Effect on knowledge cue life-cycle: Import: Lower costs

3.4. Requirements for Knowledge Models from literature

This section lists requirements from literature and those mentioned in the survey, sorted by the processes of the knowledge cue life-cycle. The requirements have been gathered from a variety of sources analysing PKM (listed for each requirement), representation formalisms (see Sec. 3.5) and user behaviour (via the knowledge cue life-cycle, see previous section). Of course, no such list can be considered "complete" – there might be even more requirements.

Sources used

For this thesis, many sources have been used (papers emailed from colleagues, followed forward- and backward references, search engines, ...). To achieve a certain degree of completeness, the following additional sources have been used: (a) Proceedings of the Annual ACM Conference on Research and Development in Information Retrieval from the years 2005, 2006, and 2007. (b) Proceedings of SIGCHI. All years up to 2008 have been queried for "personal", "knowledge", and "management". (c) Proceedings of workshops on personal information management from 2006 until 2009. For each requirement, a summary table lists other requirements this requirement depends on and which phases of the knowledge cue life-cycle are related.

3.4.1. Interaction for codify and augment process

Authoring in the knowledge model should have low cognitive overhead, i. e., the amount of cognitive activity absorbed by dealing with the system additional to the actual task. Therefore the user should be able to start with informal articulation (→ Req. 4 $^{\text{informal articulation}}$) and any desired level of granularity (→ Req. 6 $^{\text{granularity}}$). From there on, it needs to be possible to add formality stepwise (→ Req. 9 $^{\text{stepwise}}$).

Requirement 3 **Requirement 3: Fast entry**

Motivation:The survey revealed a desire for fast entry of new items and extension of existing items (mentioned by six participants in the survey).

This requirement depends on requirements: –
Effect on knowledge cue life-cycle: Codify, Augment

Requirement 4 **Requirement 4: Informal Articulation**

Motivation: Users needs a simple way to express content in an informal way, e. g., as plain text, formatted text or box-and-arrow diagrams (Oren, 2006) or "this is *nested within* that, but I can't say why".

Abecker and van Elst (2004) advocate a "Minimalization of upfront knowledge engineering. Since KM is considered an additional organizational activity orthogonal to the 'productive' work, expensive start-up activities may be a big barrier for a successful KM introduction."

Blandford and Green (2001) studied the use of short personal notes for task work and found that informal tools like paper and unstructured text files were sometimes preferred over traditional PIM applications because they supported more freeform input.

Informal articulation requires also adding existing binary files to a knowledge model without any processing. The notion of an *attachment* is common for emails and wiki systems.

This requirement depends on requirements: –
Effect on knowledge cue life-cycle: Codify and augment at low costs

Requirement 5 **Requirement 5: Formal Articulation**

Motivation: Formal reasoning can help to reduce retrieval costs, when from a set of explicitly stated formal statements further formal statements can be inferred *automatically*. This deduced knowledge has not to be constructed by hand but is already available for browsing and queries. Another kind of formal knowledge is the assignment of formal types to knowledge cues. These formal types influence how knowledge cues are handled in the system. E. g., a file type is an example for a formal type: A `.doc` file is treated different from the operating system (scanned for viruses, opened in a word processer) than, e. g., a `.jpg` file.

Knowledge that is already formalised, e. g., in domain ontologies, should be re-used in a personal knowledge model.

A good PKM tool should allow representing formal data, let the user create such formal data, and make sure that all imported formal data from other source is also easy to browse and manipulate by the user. This requires, e. g., that all items in a knowledge model have a render-able representation.

This requirement depends on requirements: 7 addressability
Effect on knowledge cue life-cycle: Structured queries and inferencing for retrieval reduce retrieval costs

Requirement 6 **Requirement 6: Let the user decide on granularity of modelling**

Motivation: Content varies greatly in size and type. Polanyi (1998, p. 81): "...linguistic symbols must be of reasonable size, or more generally

that they must consist of easily manageable objects. [...] Language can assist thought only to the extent to which its symbols can be reproduced, stored up, transported, re-arranged, and thus more easily pondered, than the things which they denote."

Documents are authored word by word, outline functions let users work on the level of sections and subsections. The whole document is stored in a single file. Desktop search engines help to find complete documents. Using a document means reading it. Sometimes reading small bits is already sufficient for a given task. For re-use, one can send the whole document or copy and paste parts of it. To sum up, documents are managed most of the time as complete entities.

More structured applications have different notions of granularity. In an address book, typically content is managed at the level of contacts, i. e., each contact is one item that can be send to somebody else, changed, created or deleted. Individual mail addresses are usually not considered to be objects on their own. E. g., if two persons happen to live in the same house, the street address has often to be maintained twice.

The granularity of items affects re-use and question answer performance: It is more costly to create a set of smaller and related items instead of one large item. To allow efficient re-use, a knowledge model should allow small content such as single words, sentences or short notes, up to full-blown formatted documents.

The web began with small personal home pages and grew up with huge search and shopping portals. Since a few years there is a tendency for smaller content granularity, especially on collaborative websites. The term *micro-content* emerged for this set of addressable content consisting of tags (single terms), comments (often not more than a single paragraph), blog posts (often about half a page), images (including meta-data and a title) or video snippets.

Shneiderman (1989) reports on a comparative study in which two groups of people had to locate answers to a series of questions in a *Hyperties* database. The group with more (46 instead of 5), shorter (4-83 lines instead of 104-150 lines) articles answered more questions correctly and took less time to answer the questions.

This requirement depends on requirements: 7 addressability
Effect on knowledge cue life-cycle: Let the user decide on trade-off between costs and benefits

Requirement 7: Entities need to be addressable Requirement 7

Motivation: To be able to link two entities, they must be addressable. To be able to model different versions of an entity, each version needs a kind of address. In a personal knowledge model, each entity that is shown to the user should be addressable.

This requirement depends on requirements: –
Effect on knowledge cue life-cycle: This affects all steps in the knowledge cue life-cycle, especially the augment and sharing steps.

Requirement 8

Requirement 8: Simultaneous use of Different Levels of Formality

Motivation: Often research notes and references are already managed digitally. Works in PIM (Jones and Bruce, 2005) and Semantic Desktop research have further stressed the need for unified search and organisation of a user's personal items.

People need to be able to work at any level of formality (or informality), and to freely mix such levels (Lethbridge, 1991b). For textual content this means exploiting syntax, structures and semantics.

E. g., in semantic wikis (cf. 2.9) all three levels can be used. While typing text, there is *syntax* for formatting (bold, italic), *structures* (headlines, lists) and *semantics* (typed links).

Another example is "SmartArt" introduced in Microsoft Excel 12, which lets the user represent the same data as a pie chart or a bullet list (Rutledge and Bajaj, 2006)[page 205pp]. Hence the semantics have been stated by the user and can be represented as different structures.

Yet another example is the Eclipse IDE for software development, which shows at runtime a *structure* tree (called the *Outline view*), derived from *syntax* constructs (the typed source code). Different from wikis, this syntax tree can also be used for re-structuring, which in turn is reflected as syntactical changes. Behind the scenes, Eclipse compiles the source code after each change to determine the semantics of it. If an error is found, it is shown as a little red marker in the textual source code.

One can assume that a typical user has the majority of the content in unstructured form, some content will be at least structured and only little content will be fully formalised. It would be too costly to structure or formalise all content. On the other hand, formalising *some* content lets the user profit, e. g., from inferred content types. So when content is typed with "Researcher", a search for "Person" would also return it – if the relation between researcher and person has been formally defined as a *sub-type*-relation. For many use cases, however, a weak structure is sufficient (e. g.,: "This issue needs to be solved before that one.", "These topics are somehow related to those.", "When I look for this address I should not forget this note").

The data will usually be in a state where some parts are not formalised at all and other parts are axiomatised. Such a mix of data could be called "semi-semantic", analogous to "semi-structured" data. Semi-structured data such as XML and HTML contains some structured and some unstructured parts.

This requirement depends on requirements: 4 informal articulation, 5 formal articulation
Effect on knowledge cue life-cycle: Allows user to adapt effort to estimated value, unified retrieval

Requirement 9

Requirement 9: Stepwise changes from informal to more formal structures

Motivation: The user should be able to migrate the knowledge into more formal structures, if desired (cf. Völkel and Haller (2006)). A very important characteristic of formal knowledge engineering in general is the *modelling process*. During modelling, a knowledge model might be in an inconsis-

Allow inconsistency

tent state. A tool should not simply prevent such inconsistent states, but rather inform the user about the consequences. The migration from one consistent formal state to another consistent formal state can be a complex operation which cannot or should not need to be carried out completely in the mind of the user. Instead, a sequence of modelling operations should allow producing inconsistent states, and then, step-by-step, move towards the desired target state. Inconsistent states should also be share-able, so that a resolution can be found collaboratively, if desired.

A typical session of gradually adding structure and semantics is given by Wiil (2005):

> ... if a knowledge worker is given the task of reviewing (understanding, organizing, and presenting) a specific body of knowledge, she may use a spatial ordering tool in the first phases of her work where no formal understanding of the knowledge units exists. In the next phases she may add metadata to customize the knowledge units. She may also add links to associate related knowledge units. Finally, she may use a taxonomic tool to classify the knowledge units to reflect her obtained understanding of the body of knowledge.

Oren (2006) advises to focus on simply capturing and representing the things that the user wants to store, before doing any reasoning with it. One participant in the survey wrote "I want to be able to massage unstructured collections of notes into a more regular form, adding structure as I go and as required."

In the early stages of a creative project (e. g., a research project, development of a documentary film, or investigation into a new application area for a medical product) it is common to rapidly accumulate a lot of references and facts which need capturing. The challenge is that at this stage in the project, the ideal structure for indexing and organizing these facts, documents, and media objects is unclear. Even the needed annotation types are unclear (Reynolds, Cayzer, Dickinson, and Shabajee, 2001).

This requirement depends on requirements: 8 formality levels
Effect on knowledge cue life-cycle: Augment: lowers cognitive costs of creating more formality

Requirement 10: Knowledge model refactoring Requirement 10

Motivation: With stepwise formalisation a user can gradually add more structure and formality to knowledge cues. Externalised personal knowledge artefacts are usually organised in a systematic manner, e. g., files are sorted in folders and sub-folders.

Unfortunately, a good structure today is not a good structure tomorrow, therefore personal organisation schemes change. Weeks, months or years later, an existing folder structure sometimes "does not make sense" and cannot help well in locating files. The knowledge model as a whole needs also to be restructured and changed to adapt to new insights and mental conceptions.

Stuart K. Card in (Jones et al., 2006) sees this not only as a tedious maintenance task, but says "re-representation of information is a key to interpreting it".

Borrowing a term from software engineering this process can be called *knowledge refactoring*. Efficient knowledge refactoring should let a user perform re-filing, re-categorisation and annotation changing operations on single or multiple knowledge cues efficiently. All kinds of structures and formal statements within and among knowledge cues should be easy to change.

Schreiber and Harbo (2004) emphasise flexibility of knowledge models and the need for reorganisation:

> A central question is how to understand a structure of the collected information. [...] But, as known, the problem is that the process of learning will continue and all the time the individual will be inspired to look after new subjects or new elements of the subject. After a while, the individual will experience that the structure of the information is not the right one anymore. It is necessary to choose a new way to structure the information. [...] The old one will not be used and, further, it will make a barrier of the learning process. Thus, a flexibility concerning how to structure the information is necessary.

One of the more often requested features (survey: fifteen participants) was support for (re-)structuring existing structures: A PKM tool should "help to structure and sort items, be easy to restructure, help to move from unstructured to more structured, organize pieces of larger text, and help to categorize items according to existing filing schemes such as taxonomies, tags, vocabularies and ontologies" (survey).

This requirement depends on requirements: – Effect on knowledge cue life-cycle: Augment: Lowers costs of restructuring

Requirement 11

Requirement 11: Versioning

Motivation: The cost of creating and manipulating knowledge cues is lower, if people have an easy way to undo their operations and revert to previous versions of a knowledge model. This concept of "make it easy to fix errors instead of trying to prevent them" is one of the defining characteristics of a wiki (Leuf and Cunningham, 2001).

This requirement depends on requirements: 7 addressability Effect on knowledge cue life-cycle: Codify and augment at lower costs

Requirement 12

Requirement 12: Capture the context for knowledge cue creation and import

Motivation: Here context is understood as the work context in a technical sense, i. e., currently active software tools, kind of loaded files, system time, etc.

Codify, augment

Understand the notion of context, capture it together with the information and use it to enhance recall and understanding (Oren, 2006). E. g., the

creation date of a knowledge cue can easily be captured at creation time. In most collaborative systems such as wikis, blogs, and social networking sites, the author and the time of creation or last change are automatically logged and used for searching and browsing. Automatic context tracking should relive the user from maintaining bookkeeping data such as creation data of items or linking (→ Req. 20 [(hyper)-links]) two items that are commonly used together (Graça Pimentel, Abowd, and Ishiguro, 2000). Bettoni et al. (1998) suggest supporting PKM by tracking communication, tasks and contexts, used files, and concepts.

Import

Users wish it should be clear "which data is from my personal information sphere and which is coming from outside" (survey). At import time, the import source and time of creation are useful to track the provenance of knowledge cues.

This requirement depends on requirements: 7 [addressability]
Effect on knowledge cue life-cycle: Codify, augment, import

Requirement 13

Requirement 13: Active assistance in maintenance tasks

Motivation: Knowledge organisation schemes change, so the knowledge model needs to be refactored (Req. 10 [refactor]). However, even with good refactoring support available, it would be too costly to re-organise all personal artefacts frequently. Knowledge cues get out-dated. Old knowledge cues appear in search results and while browsing, hence the retrieval costs become higher than necessary. Metadata about the usage of the knowledge cues by the system is required: How often did the knowledge cue appear in search results? How often has it been changed? When has the most recent statement been made about this knowledge cue? When was the last time this knowledge cue was used for inferencing? Such metadata can be used by the system to ask a user specifically and actively about the status of certain knowledge cues.

This requirement depends on requirements: 12 [context]
Effect on knowledge cue life-cycle: Augment (maintenance is mostly a form of augmentation), reflect

Requirement 14

Requirement 14: Easy to learn

Motivation: Each new tool has a learning curve that depends on the complexity of the underlying concepts and the user interface. A good user interface cannot compensate for an ill-designed underlying data model. Therefore this thesis strives to create a data-model, which is similar to existing data models in use in PKM tools.

This requirement depends on requirements: –
Effect on knowledge cue life-cycle: A tool that is easier to learn has lower initial costs of using

Ways of adding structure and formality

Requirement 15

Requirement 15: Grouping of items

Motivation: The tool should be able to let me group seemingly unrelated content (survey). Users need composition for navigation (Frank, 1988). This

allows, e. g., browsing and thereby narrowing down their view and allows discovering related, yet unexpected items.

It is important for a user to be able to group seemingly unrelated content together, so that retrieval of one item triggers retrieving of the others, too (Jones, Phuwanartnurak, Gill, and Bruce, 2005).

Grouping knowledge cues is also a pre-requisite for any kind of batch operations such as exporting, sharing, refactoring, delete and copy operations.

This requirement depends on requirements: 7 addressability Effect on knowledge cue life-cycle: Augment, import, retrieve, export

Requirement 16

Requirement 16: Containment relationship

Motivation: A containment relationship is a stronger form of grouping with additional semantics for operations. Delete and copy commands on containers trigger recursively the same command for all *contained* elements. Such containment semantics are present, e. g., in file explorers (deleting a folder deletes all files and sub-folders), presentation software (deleting a slide deletes the content on the slide), programming (deleting a package deletes all classes therein), ... As this interpretation is so common, it should exist in a similar way in a knowledge model.

This requirement depends on requirements: 15 grouping Effect on knowledge cue life-cycle: Augment, import, retrieve, export

Requirement 17

Requirement 17: Optional naming of knowledge cues

Motivation: The data model should allow giving things human-usable names. A name is understood as a *unique name*. Names make linking much easier, as the link target name can simply be typed and one has not to select from a complex GUI. Names also allow direct navigation deep into a knowledge model.

The success of wikis and their page naming scheme shows the importance of naming.

Users also demanded that it should be easy to place new items into a named container (survey).

But as users often have difficulties to find names (Boardman, 2004)[p. 105], Frank (1988) advises to not *require* a user to name all items. Consider several contacts in an address book to link to the same postal address (e. g., all 20 people working for a non-governmental underwater-life-protection organisation). In this case it would be too much work to assign a dedicated name to *the address of the office*. Yet it would also be cumbersome to have to change the entries of all these people in case the postal address of the NGO changes. Therefore it should be allowed, but not required, to give entities a name.

Names allow a user to fetch a unit of information in $O(1)$[13]. This is similar to know, e. g., the URL of a certain web page or the file name and path of an office file. Human-usable naming is probably an overlooked area

[13] That is, in a single step. These costs remains small, independent of the growth of the knowledge model.

of content management. E. g., wikis allow users to use easy-to-remember unique names to quickly navigate or link to known pages. The semantic web is fundamentally built on URIs, which are unique names for resources. Unfortunately, URIs are not designed for human usage: they are cryptic strings, hard to read and remember.

This requirement depends on requirements: 7 addressability
Effect on knowledge cue life-cycle: Lower retrieval costs

Requirement 18: Alternative names Requirement 18

Motivation: Many systems with unique names have also means to create additional *alias* names, which redirect to another unique name. E. g., in MediaWiki there is a redirect-concept, in HTTP there are several kinds of redirect status codes to instruct a browser to load another URL instead, and in the file system there are shortcuts (windows) or links (Linux) to other files.

This requirement depends on requirements: 17 naming
Effect on knowledge cue life-cycle: Lower retrieval costs

Requirement 19: Order knowledge cues Requirement 19

Motivation: Ordering a collection of ideas or text snippets into a coherent flow is one of the main tasks of authoring (Esselborn-Krumbiegel, 2002). A user should be able to create order gradually and partially. E. g., by stating the reading sequence between some sections. Note how different this is from providing a list data-structure: A list can only represent a total order.

This requirement depends on requirements: 7 addressability
Effect on knowledge cue life-cycle: Augment, import, retrieve, export

Requirement 20: Linking Requirement 20

Motivation: Seven survey participants required links between items, e. g., a link between the tasks "buy food for dog" and "bring dog to veterinary". Participants mentioned links between notes and from notes to knowledge sources.

Oren (2006) finds "an under-utilisation of the interlinked nature of the information". Knowledge models should allow for precise and effective linking - and browsing (→ Req. 27 follow links and browse). While browsing associatively, it should be easy for a user to "bookmark" found things, that is, create a link from a known starting position (→ Req. 17 naming) to the newly found knowledge cue.

This requirement depends on requirements: 7 addressability
Effect on knowledge cue life-cycle: Augment, import, retrieve, export

Requirement 21: Hierarchy Requirement 21

Motivation: Shneiderman (1996, p. 336) emphasizes the need to get "Overview first, zoom and filter, then details-on-demand." Users in the survey required being able to hide level of details to get an overview of the content. Others wished a graphical overview that represents connections and interactions between notes.

Hierarchies of all kind are commonly used in user interfaces to let the user narrow down his interests step-by-step.

For Noirhomme-Fraiture and Serpe (1998) hierarchical links constitute one half of the category of *structural links*, cross references are the second half.

This requirement depends on requirements: 7 [addressability]
Effect on knowledge cue life-cycle: Augment, import, retrieve, export

Requirement 22

Requirement 22: Simultaneous use of multiple levels of detail

Motivation: Users need ways to see multiple levels of detail at once (Frank, 1988):

> Another problem with this arrangement is that the user can see the entire document at only one level. [...] there is no way to zoom in and out of the document structure, examining its contents at different levels of detail. This capability is commonly found in outline processors and is a critical component in many writing and information organization tasks.

This requirement depends on requirements: 21 [hierarchy]
Effect on knowledge cue life-cycle: Augment, retrieve

Requirement 23

Requirement 23: Annotating content

Motivation: When using documents, a field study of O'Hara and Sellen (1997) concludes that annotating documents is frequently a part of the document reading and understanding process.

Personal annotations are changed radically before being shared with others (Marshall and Brush, 2004).

Peter et al. (2006) argue for making structures within documents explicit via annotations.

This requirement depends on requirements: 7 [addressability]
Effect on knowledge cue life-cycle: Augment, import, retrieve, export

Requirement 24

Requirement 24: Tagging

Motivation: Tagging, which can be seen as a specific kind of annotating, is the basic assignment of easy-to-type keywords to information artefacts. Tag names contain usually no whitespace and tend to be really short. A common representation is a *tag cloud*, showing all tag names at once, with a font size proportional to their usage frequency.

People have problems in using strict hierarchies (Oren, 2006). Therefore less strict methods such as tagging (→ Req. 24 [tagging]) and categories (→ Req. 25 [categories]) are required.

This requirement depends on requirements: 7 [addressability]
Effect on knowledge cue life-cycle: Augment, import, retrieve, export

Requirement 25

Requirement 25: Classifying into categories

Motivation: Categories are usually used slightly different than tags: Categories tend to have longer, encyclopedia-like names. E. g., *elephant* would be a typical tag name (lowercase, no spaces, rather short), whereas *Indian*

Elephant would be a typical category name (title casing, contains spaces, more specific).

In most category-systems, there is a weak hierarchy, i. e., categories can often be nested into (several) other categories. In practice, the boundaries between tags (short, easy-to-type, not nested) and categories (nice to read, nested) are blurred sometimes.

In the survey, users prefer categories over strict hierarchies (mentioned three times). One participant described this as being able to put the same content to different locations simultaneously.

This requirement depends on requirements: 7 addressability
Effect on knowledge cue life-cycle: Augment, import, retrieve, export

3.4.2. Interaction for retrieval process

Requirement 26: Queries Requirement 26

Motivation: Besides browsing a user also needs the ability to search and query the data (Frank, 1988).

A number of different types of queries are required:

Full-text queries lower the costs of retrieval if textual parts of knowledge cues are remembered. Query formulation costs are low and full-text query capabilities are well known.

Aggregate queries operate on structured data and can return, e. g., a result table which is composed of many small knowledge cues scattered in a knowledge model.

Structured queries can, e. g., create result tables and aggregate knowledge cues. Aggregation is, e. g., summing up individual numbers. They can thus save a lot of manual work and lower the costs even further.

Metadata queries allow retrieving items based on semantic properties. Thanks to the higher precision, compared to full-text queries, metadata queries can lower the retrieval costs.

Formal queries can exploit formal semantics to return more precise results, including inferred facts. Hence retrieval costs are lowered again.

This requirement depends on requirements: 5 formal articulation
Effect on knowledge cue life-cycle: Lower costs of retrieval

Requirement 27: Following links and browsing collections Requirement 27

Motivation: Following links is one of the three core strategies of information retrieval described by (Bates, 2002), cf. Sec. 3.3.3. It becomes necessary if only vague associations are remembered, but the desired knowledge can neither be retrieved by full-text search nor by complex queries. There is a fundamental difference between search (where you know what you are looking for) and browsing (where you find things that you placed there)

(Jones, Phuwanartnurak, Gill, and Bruce, 2005). Oren (2006) summarizes: exploit the interlinked nature, do not rely only on search, and allow people to associate freely.

This requirement depends on requirements: – Effect on knowledge cue life-cycle: Lower the costs of retrieval

Requirement 28
Inverse links

Requirement 28: Inverse Relations

Motivation: Many wikis allow traversing hyperlinks not only forward, but also in backward mode. For each page they list all pages linking to the page. Similar feature are provided by major web search engines and blog systems ("trackback"). The *tabulator* semantic web browser[14], uses inverse relation labels to enhance the browsing experience. In most semantic GUIs incoming links are rendered different from outgoing links. Therefore it makes a difference for browsing whether a user stated ([SAP], [employs], [Dirk]) or ([Dirk], [works for], [SAP]). For the user, this is often an artificial distinction.

In order to allow browsing *semantic links* in a knowledge model, links must be traversable in both directions. Therefore, it is desirable that link types have labels for both directions, e. g., "works for" and "employs".

Note: In OWL and NRL (cf. Sec. 2.6), inverse relations are *allowed* but not *mandatory*. Therefore a tool cannot rely on inverse relations and need to provide (sometimes cumbersome) work-arounds.

This requirement depends on requirements: 20 $^{\text{(hyper)-links}}$; relation instances are typed links Effect on knowledge cue life-cycle: Retrieve

3.4.3. Expressivity

Requirement 29

Requirement 29: Flexible schema

Motivation: The survey paper of Oren (2006) states a requirement for flexible schemas: Leave users their freedom and do not constrain them into rigid schemas. This is not only relevant for authoring but also for importing from other data models to be able to represent as much of the given structure and formality as possible.

This requirement depends on requirements: Req. 5 $^{\text{formal articulation}}$ Effect on knowledge cue life-cycle: Augment: Higher expressivity → better retrieval possible. Import: Can represent more of the existing structures → better retrieval at lower costs

Requirement 30

Requirement 30: Transclusion

Motivation: Users often loose structure of knowledge cues when transforming from one tool to another. E. g., text snippets from a hypertext context loose their identity when pasted into a document. Instead of copying the value of an item it is more elegant to copy a reference to the item. If the content item is changed, the change is reflected in all parts where it is embedded. This ultimately lowers the cost of augmentation, as changes in one knowledge cue have not to be propagated manually. Embedding a reference and rendering the content is called *transclusion*. The need for transclusion

[14] `http://www.w3.org/2005/ajar/tab` (accessed 06.01.2010)

is further explained by Ludwig (2005), Nelson (1995) and in the evaluation of Popcorn described by Davies, Allen, Raphaelson, Meng, Engleman, King, and Lewis (2006, p. 155).

This requirement depends on requirements: 7 addressability Effect on knowledge cue life-cycle: Augment: Lowers the cost of maintenance

Requirement 31: Meta-Modelling Requirement 31

Motivation: If knowledge cues become old, but not outdated, they become just harder to understand. The meaning of terms shifts. It is therefore required to let the user describe and annotate all aspects of knowledge cues. Even annotation on annotations, statements and relations are sometime required, e. g., annotating a statement with the justification for stating it. This allows a user to create a more self-describing knowledge model.

The data model must allow annotating (and therefore addressing) all of its elements, in order not to limit expressivity.

This requirement depends on requirements: 7 addressability Effect on knowledge cue life-cycle: Augment: Higher expressivity → better retrieval

This concludes the general requirements list, which was gathered from literature studies. As with every requirements list, there are certainly even more requirements that could be added. However, if the conceptual model of a PKM tool and the PKM tool itself would fulfill all the requirements stated here, its users would likely be quite happy already. The next section analyses the conceptual models of existing tools used for PKM tasks, in order to extract the most commonly used relation types.

3.5. Analysis of relations from conceptual models of tools used for PKM

As required by requirement 2 super-set, the knowledge model should be a super-set of existing conceptual models. This section analyses the conceptual models of a number of existing tools used for PKM tasks. Methodology

Learning a new tool requires a user to understand the conceptual model of the tool. It is easier to learn a tool with a familiar conceptual model. Hence, the formalism of a good PKM tool should be similar to those of other tools in use (→ Req. 2 super-set). Cost of learning

This section looks only in domain-free (cf. Req. 29 domain-free) PKM tools. Additionally, one can assume that the conceptual model of those tools that *are* used most for PKM tasks, are those that are suited rather well. It is possible that new, different conceptual models will once be developed, which will be adapted much better to the human psychology and PKM tasks in particular. However, those imaginary models are likely to suffer from (a) a longer learning curve (it takes longer to learn something that does not resemble known things) and (b) an incompatibility to existing tools and content stored in the models of them. A knowledge model should be expressive enough to represent existing application data models to enable re-use of external structures.

The approach is instead to consolidate existing structures under the assumption that those have been "proven" to work. The conceptual models of user interfaces, data structures, and document models have been analysed. From this analysis a number of requirements for a common knowledge model are distilled. In detail, the following steps have been performed:

- Relevant categories – Select relevant formalism and tool categories used in PKM contexts. Relevant is defined in this context as being used by a large number of knowledge workers to manage knowledge cues. For details on knowledge cues, refer to Sec. 1.2.3.
- Popular instances – Select a popular formalism or tool instance from the category.
- Analyse the conceptual model (cf. Sec. 2.13) and extract common relations (Sec. - Sec. 3.5.6).
- Create a subsumption hierarchy of relations. This hierarchy is presented in Sec. 4.2.

The remainder of this section details each step that has been performed.

Relevant categories What approaches are used in practice to organise personal information and knowledge? What approaches deal with the representation of vague knowledge, allow stepwise formalisation or expose a modelling language to and end-user? A comprehensive list is hard to define precisely, yet the following category list is broad enough, to cover all relevant (vague knowledge, stepwise formalisation, exposing modelling language) approaches used in practice to organise information and knowledge:

- Personal notes. Popular examples of notes on paper are to-do lists, shopping lists, diaries, and lab books.
- Documents. An interesting aspect of documents is the way structure is created in word processors in an integrated fashion with typing the main text. By exploiting these structures and codifying the explicitly in a knowledge model, the costs for creating structures are lowered. Such a mapping is detailed in Sec. 4.3.
- Hypertext.
- Desktop information management tools.
- Data structures: Data structures in programming languages are likely to contain re-usable building blocks to represent many kinds of basic semantics.
- Tools for Structuring Knowledge and Creativity Tools, such as mind mapping and concept mapping.
- Collaborative information organisation tools.

Popular instances This paragraph identifies a popular formalism or tool within each category, in order to distil the essentials of their data model concepts.

Personal Notes: Although they are used by virtually every knowledge worker, the relation types used on paper are highly individual and hard to categorise and describe. They have therefore to be omitted from this analysis of commonly used relation types. A background on personal notes is given in Sec. 2.7.

Documents: A background on documents is given in Sec. 2.2. Documents exist on paper as well as in electronic form. The conceptual model behind them is the same in both cases. Which text features are commonly used? This section models the most important structural aspects of documents, namely sequential navigation, logical structure, argumentative structure and content semantics.

Hypertext: A background on hypertext is given in Sec. 2.4. By far the most prominent example of hypertext in use is the *World Wide Web* with its main representation format HTML. Among the most popular tools to *create* interlinked HTML documents are probably wikis and to some extent blogs. Both tools offer a rich subset of HTML features. Of course, other tools allow to create more beautiful HTML pages, looking like printed magazines. However, such design-oriented tools are rarely used for managing personal content.

Desktop information management tools: Most popular operating systems today (Windows, Mac OS, Linux/KDE, Linux/Gnome) still follow the WIMP-metaphor (cf. Sec. 2.3). All of them contain a desktop with icons and a file browser (Windows: Explorer, Mac OS: Finder, Linux/KDE: e. g., Konqueror).

The Microsoft Windows Explorer is likely the most used tool, as it is present in all Windows versions which have a market share of $\sim 90\%$[15]

Data structures: A background on models and modelling is given in Sec. 2.1 and on software engineering in Sec. 2.5.

Structuring and Creativity: Popular creativity tools are mind-maps and concept maps. Both are popular graphical approaches for structuring information and helping the user to get an overview. Mind maps can also be used to present information to others. A background on mind-maps is given in Sec. 2.10. Mind-maps are mostly used for (re-)structuring ($\rightarrow$ Req. 10 $^{\text{refactor}}$) information atoms for rather short term-use. Not many people use mind-maps to take notes to be read later weeks, months or years later.

[15] 89.6% on 1.12.2008 according to Net Applications

A non-scientific poll[16] of five popular mind-mapping tools voted *Freemind*[17] as the most popular tool.

Collaborative Information Organisation Tools With the advent of "Web 2.0", some web sites dedicated to collaborative information management became popular. Among these, *Delicious*[18], for organising bookmarks and *flickr*[19], for organising pictures have been chosen for further analysis.

Both systems allow users to "tag" their content and to browse the implicitly created sets of items which share a common tag. Many other popular web 2.0 sites for collaborative information organisation also offer tagging as a lightweight means to structure information. Some systems even allow adding structure to the tags themselves (e. g., Soboleo[20] and Bibsonomy[21]).

Notational convention In the remainder of this chapter, the concepts and relations of tools are summarised in a standardised way. Tab. 3.2 shows an example for the typographic conventions used.

Concepts	
List of concepts used in the tool, e. g., A, B, ...	
Relations	
From concept/to concept	*Semantics*
A/B	Describes the relation from concepts of type A to concepts of type B.
C/C	Describes a recursive nesting of concepts of type C to themselves.
...	...

Table 3.2.: Typographic conventions used in this thesis for summarising concepts and relations

[16] Poll conducted in a blog post at `http://lifehacker.com/5188833/hive-five-five-best-mind-mapping-applications` (accessed 09.04.2009) from Jason Fitzpatrick on March 2009. Based on answers from 3510 readers, the results are: MindMeister 11% (401 votes), MindManager 25% (878 votes), XMind 20% (715 votes), FreeMind 27% (938 votes), iMindMap 4% (136 votes), Other 13% (442 votes).

[17] `freemind.sourceforge.net` (accessed 06.01.2010)

[18] `www.delicious.com` (accessed 06.01.2010)

[19] `www.flickr.com` (accessed 06.01.2010)

[20] `www.soboleo.com` (accessed 06.01.2010)

[21] `www.bibsonomy.org` (accessed 06.01.2010)

3.5.1. Documents

The core concepts and relations for structuring documents are shown in Tab. 3.3. A detailed introduction to documents can be found in Sec. 2.2.

There are many popular document formats. In the digital world, variants of HTML[22] are likely to be among the most popular ones. HTML is analysed in the next section.

Concepts	
document, section, paragraph, sentence, word, style, reference, title	
Relations	
From concept/to concept	*Semantics*
document/title, section/title	Some elements can have a title.
word/word, sentence/sentence, paragraph/paragraph, section/section, ...	All parts of a document can be traversed in sequential order (→ Req. 19 $^{\text{order}}$).
document/section, section/paragraph, paragraph/sentence, sentence/word	The *logical structure* can be modelled (cf. Groza et al. (2007)) as a hierarchical tree (→ Req. 21 $^{\text{hierarchy}}$).
.../style	Any element in a document can have a different style, which defines, e. g., bold, italic, underline typesetting, the font family and font size.
sentence/sentence	Some formats, e. g., SALT (Groza et al., 2007), allow encoding the argumentative structure as well (→ Req. 5 $^{\text{formal articulation}}$).

Table 3.3.: Core concepts and relations of documents

3.5.2. Hypertext

The main concept in HTML and the WWW is the notion of a *hyper-link* which – when activated – moves the focus away from the document the user is looking at towards another document (→ Req. 20 $^{\text{(hyper)-links}}$). Hyperlinks in WWW are directed. They have a single source and a single target.

Before the broad use of WWW there were a number of other hypertext systems that seem to have been harder to deploy on a global scale, but which might also contain a number of interesting concepts for PKM. Therefore this thesis looks also into the *Xanadu* data model. The core concepts and relation types of hypertext are shown in Tab. 3.4.

[22] Even estimating the number of publicly hosted web pages is difficult. By combining number of registered web sites and average number of pages per site, an estimate of 29.7 billion pages seems reasonable. Source: `http://www.boutell.com/newfaq/misc/sizeofweb.html`, accessed 09.04.2009. This number can be contrasted with the number of all book titles published in the world ever, estimated as 65 millions. Source: A study from the University of California in 2000, `http://www2.sims.berkeley.edu/research/projects/how-much-info/print.html`, accessed 09.04.2009.

Concepts	
document, element, hyperlink	
Relations	
From concept/to concept	*Semantics*
document/element	A document contains elements (→ Req. 16 containment).
element/hyperlink	An element can act as a hyperlink (→ Req. 20 (hyper)-links).
hyperlink/element, hyperlink/document	A hyperlink can link to another element in the same document or to another document. A link to an element in another document is also possible in HTML.

Table 3.4.: Core concepts and relations of hypertext

Xanadu, Transclusion

A central idea of *Xanadu* is *transclusion*, as defined by Nelson (1995):

> The central idea has always been what I now call *transclusion*, or *reuse with original contexts available*, through embedded shared instances (rather than duplicate bytes).

Transclusion allows using a piece of content inside many documents. If the content piece is changed, it changes in all documents that use it (→ Req. 30 transclusion). The same feature is requested, e. g., by Ludwig (2005).

NoteCards

NoteCards (Halasz, Moran, and Trigg, 1987) was one of the first and very popular hypertext systems. It was designed to run on an individual's workstation. One of its designers summarizes open issues of the system (Frank, 1988), of which some apply to PKM tools:

1. Users need search and query functionality (→ Req. 26 queries), i. e., associative browsing (→ Req. 27 follow links and browse) is not enough.

2. Users need composition (→ Req. 15 grouping) and ways to see multiple levels of detail at once.

3. Do not require a user to name all items (→ Req. 17 naming). Segmentation of content, titling and filing happen step-by-step (→ Req. 9 stepwise).

An hypertext study by Shneiderman (1989) found out that users are better able to answer questions when a text is modelled as more-fine grained (here: 46 articles) hypertext instead of large chunks (here: 5 articles) (→ Req. 6 granularity).

3.5.3. File explorer

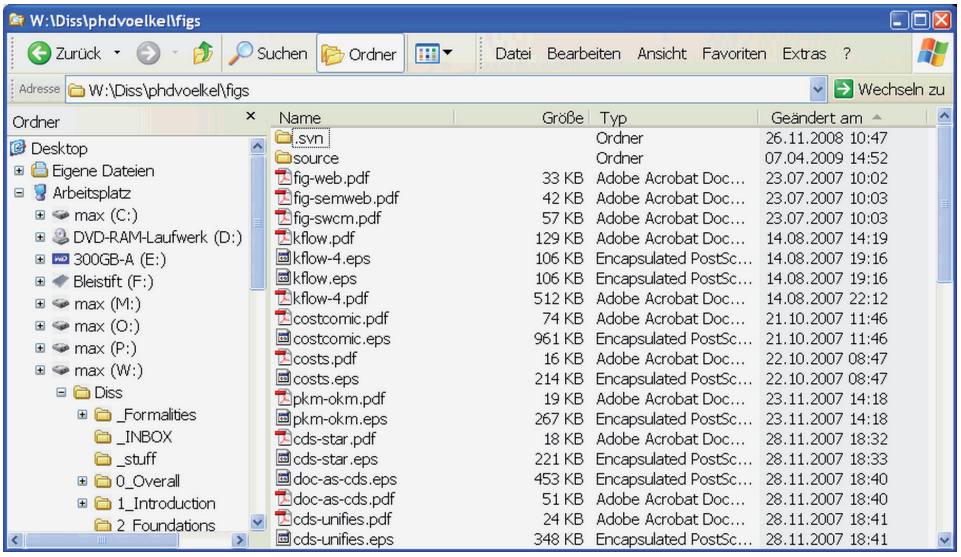


Figure 3.11.: Microsoft Windows XP Explorer

The file browser basically shows a strict hierarchy of directory names (cf. Fig. 3.11, left) and the files contained therein (right side). File and directory properties such as size, file type, and change date are also shown in this tool (right side).

In the explorer, users can browse a tree (→ Req. 19 [order]) (→ Req. 21 [hierarchy])) and thus narrow down their search and discover related, yet unexpected items. As summary of core relations is given in Tab. 3.5

Concepts	
drive, folder, file, shortcut, size, file type, date	
Relations	
From concept/to concept	*Semantics*
file/name, folder/name	human-readable name, can be changed (→ Req. 17 [naming])
drive/folder,folder/folder	nesting, hierarchy (→ Req. 21 [hierarchy])
drive/file, folder/file	container membership (→ Req. 15 [grouping])
shortcut/file	a shortcut is an alternative name for a file (→ Req. 17 [naming])
file/size	computed attribute
file/file type	formal type that determines what happens when users click (→ Req. 5 [formal articulation])
file-change date	computed attribute, last time the file was edited with changes (→ Req. 12 [context])

Table 3.5.: Core concepts and relations of file explorers

3.5.4. Data structures

Structures that are deeply built into modern mainstream programming languages, seem to have worked well in the past to moderate between humans and computers.

- *Java* is one of the most popular all-purpose programming languages. It has native support only for arrays. The included *Java Collections Framework* adds support for sets, lists and maps.
- *Python* is a scripting language with dynamic typing. It is often recommended as a language for beginners, as it has built-in support for many data-structures: arrays (lists), sets and maps (dictionary).

The relation types needed to represent these data structures are:

Unordered collection Does not imply or require an order among items. This is the same as (→ Req. 15 grouping).

Lists Membership in a collection (→ Req. 15 grouping) plus a total order (→ Req. 19 order) among its elements.

Maps A map is a collection of items which has for each item (map key) a defined corresponding item (map value). The corresponding item is usually not a map key itself. The functionality of a map requires grouping the set of assignments (→ Req. 15 grouping) and linking values to keys (→ Req. 20 (hyper)-links).

The core concepts and relations are summarised in Tab. 3.6.

Concepts	
element, collection, list, map, set	
Relations	
From concept/to concept	*Semantics*
collection/element	A collection in general is a multi-set. An element is contained in a collection zero, one or several times. Does not imply or require an order among items. This is the same as (→ Req. 15 grouping).
element/element in a list	In a list, there is a total order among elements (→ Req. 19 order).
element/element in a map	A map is a collection of items which has for each item a defined corresponding item (key-value-pairs). The functionality of a map requires grouping (→ Req. 15 grouping) and linking (→ Req. 20 (hyper)-links).

Table 3.6.: Core concepts and relations of data structures in programming languages

3.5.5. Mind- and Concept Maps

Mind Maps in Freemind

Conceptually, a **Mind Map** (cf. Sec. 2.10) is a tree (→ Req. 21 $^{\text{hierarchy}}$) of nodes, centred around a central root node. In many popular tools such as *FreeMind* and *Mind Manager* only the nodes can carry labels, the arcs cannot. The data-model of *FreeMind* is summarised in Tab. 3.7.

Concept Maps in CMap Tools

Concept Maps can be edited, e. g., in *CMap Tools*. In this tool the user can label nodes and links. There is not a distinguished central node. The data model of *CMap Tools* can be decomposed as shown in Tab. 3.8.

Concepts	
map, node, icon, description	
Relations	
From concept/to concept	*Semantics*
map/root-node	every map has a root-node
node/node	nodes are connected either hierarchical (→ Req. 21 $^{\text{hierarchy}}$) or by cross-cutting links (→ Req. 20 $^{\text{(hyper)-links}}$)
node/node	within a parent node the list of nodes is ordered (→ Req. 19 $^{\text{order}}$)
node/icon	each node can have a number of icons (→ Req. 23 $^{\text{annotation}}$)
node/text	a node can have a full-text description

Table 3.7.: Core concepts and relations of Mind Maps

Concepts	
node, line, label	
Relations	
From concept/to concept	*Semantics*
node/node	two nodes can be connected via a line (→ Req. 4 $^{\text{informal articulation}}$)
node/label, line/label	nodes and lines can be labelled (→ Req. 17 $^{\text{naming}}$)

Table 3.8.: Core concepts and relations of Concept Maps

3.5.6. Collaborative information organisation tools

The data model of tagging sites consists of items which can have zero, one or more tags. Core concepts and relations of tagging systems are shown in Tab. 3.9.

A knowledge model should allow tagging items (→ Req. 24 [tagging]). Similar to Bibsonomy (cf. Sec. 2.11), users should be able to create structure between tags (→ Req. 5 [formal articulation]).

Concepts	
resource (URL or digital image), tag, user	
Relations	
From concept/to concept	*Semantics*
resource/tag	annotation of resource with tag (→ Req. 24 [tagging])
tagging/user	which user tagged which resource with which tag? (→ Req. 31 [meta-modelling]), (→ Req. 12 [context])
tag-bundle/tag	nesting of tags, in *Delicious* used as conjunctive queries and for hierarchical browsing of tags (→ Req. 21 [hierarchy])

Table 3.9.: Core concepts and relations of tagging systems

3.5.7. Summary of common relations

The main relations used in the different conceptual models are summarised in Tab. 3.10.

Relation	Documents	Hypertext	File Explorer	Data structures	Mind- and Concept Maps	Collaborative
Generic relatedness					+	
(Hyper-)links	+	+	+	+	+	
Order	+	+		+	+	
Generic hierarchy	+	+	+		+	
Formal typing			+	+		
Type hierarchy				+		+
Annotation					+	
Tagging						+

Table 3.10.: Summary of common relations in different conceptual models

3.6. Knowledge representation

As the knowledge model designed in this thesis should be able to represent knowledge cues and formalised knowledge in a domain-independent way, this section looks into existing knowledge representation formalisms to gather further requirements.

Ontologies for PKM?

Existing knowledge representation languages are very general (RDF, RDF Schema, OWL, SKOS) and feature few semantic relations suited to directly model and structure personal knowledge. Furthermore, dealing with them requires quite a technical mind-set, e. g., thinking about the distinction between literals, blank nodes and URIs (RDF, OWL).

Note that this is not a matter of just a good ontology editor missing! If an ontology editor really allows a person to model any kind of RDF data-structure, then the concepts URI, blank node and language-tagged literal need to appear somewhere in the user interface, which means that a user has to learn and understand these concepts. If, on the other hand, the user is only able to model a *subset* of an ontology language, then this is a loss of expressivity. And in fact, such a tool is no longer an, e. g., RDF-editor but an X-editor, where X is some language, based on a subset of RDF. This thesis aims to identify a language X – not necessarily a subset of RDF – that is easy enough to be used directly by end-users for their PKM tasks, as well as expressive enough to automate some of the PKM tasks, e. g., by powerful queries.

This thesis looks into XML and RDF as basic data model representations. Then, ontology languages are analysed, namely RDF schema and OWL.

3.6.1. Data exchange formats

The *structures* used in data exchange languages must work for all use cases of that data format. Many parts in a data format vary, but some structures cannot be changed. This section analyses these built-in structures. In particular, XML and RDF are analysed.

XML For data exchange, XML (introduced in 2.5) is probably one of the most popular languages today. The data model used in XML is the so-called XML info-set. This is a strict tree of elements. Elements may contain text and a number of attribute-value pairs. Basically XML encodes a labelled tree with ordered children.

Representing XML requires hierarchical (→ Req. 21 $^{\text{hierarchy}}$) nesting of elements, unique names for the elements and attributes (→ Req. 17 $^{\text{naming}}$), and modelling of attributes. An attribute is basically a key-value-pair connected to an element. As an example, the image element `IMG` of XHTML has an attribute `SRC` to state the source URL of an image. The attribute is a *part-of* the element, a case of hierarchical nesting (→ Req. 21 $^{\text{hierarchy}}$). Furthermore, the key of the attribute denotes the semantics of the attribute. In fact, not the complete key-value-pair is part of the image, only the URL value is. The key is only stating the role or type the value plays.

More formally, the semantics of

```
<aaa bbb="ccc"> ... </aaa>
```

can be modelled as three formal statements: (aaa, hasPart, bbb), (bbb, has Value, “ccc”), (bbb, hasType, key). As requirements we get the need for representing part-whole-hierarchies (→ Req. 21 hierarchy) and for assigning types to items (→ Req. 5 formal articulation).

Concepts	
document, element, attribute, key, value, text	
Relations	
From concept/to concept	*Semantics*
document/element	Each document has a root element
element/element	Elements can be nested into each other, forming a tree (→ Req. 21 hierarchy)
element/attribute	Each element can contain any number of attributes. Within one element, each attribute-name (→ Req. 17 naming) may be used only once.
element/text	An element may contain text of any length (→ Req. 6 granularity).

Table 3.11.: Core concepts and relations of XML

RDF A background on RDF is given in Sec. 2.6. As RDF (i. e., the core triple model) is itself a generic model to represent typed relations, there are no particular relations *types* present (see also RDFS).

RDF allows several kinds of meta-modelling, i. e., besides annotating reified statements, one can assign types to types (→ Req. 31 meta-modelling).

3.6.2. Ontology and schema languages

A background on semantic technologies is given in Sec. 2.6. This section analyses RDF Schema, OWL and SKOS.

RDF Schema and OWL RDF Schema (Dan Brickley, 2004) does not distinguish between class and instance. Every resource can be used as a type of another resource. OWL is stricter, and mandates a clear separation of modelling layers. A knowledge model should be able to assign types to items (→ Req. 5 formal articulation). A knowledge model should allow meta-modelling, i. e., assigning types to types (→ Req. 31 meta-modelling).

Both RDF Schema and OWL have inheritance hierarchies of classes and properties. Both have the notion of domains and ranges for properties; they are used in a reasoner to infer types of instances on which theses properties are used. OWL offers a construct to state equality between concepts, RDF Schema lacks such a construct.

Concepts
Resource, Property, Class, Literal, List, Statement

Relations *From concept/to concept*	*Semantics*
Resource/Class	Each resource can have any number of classes (`type`) (→ Req. 5 [formal articulation]).
Class/Class, Property/Property	Classes and properties can be arranged in formal type hierarchies (`subClassOf`, `subPropertyOf`) (→ Req. 21 [hierarchy]).
Property/Class	Properties allow inferring the type of a resource used as a subject (`domain`) and object (`range`) of a triple.
Resource/Literal	One or several human-readable `labels` (→ Req. 17 [naming]) and `comments` (→ Req. 23 [annotation]) can be attached to any resource.
List/Resource, List/List	Lists allows for ordered (→ Req. 19 [order]) collections of resources.
Resource/Resource	A resource can be a `member` of a non-ordered collection (→ Req. 15 [grouping]).
Resource/Resource	A resource can also be linked to another one in a non-semantic way (`value`), merely providing and addressable set of values (→ Req. 4 [informal articulation]).
Resource/Resource	A kind of unspecified hyperlink is modelled as `seeAlso` (→ Req. 20 [(hyper)-links]).
Resource/Resource	A resource can be linked to formal definitions via `isDefinedBy`. This is used to model, e. g., ontologies (→ Req. 5 [formal articulation]).
Statement/Resource	To allow meta-modelling (→ Req. 31 [meta-modelling]), a resource of type `Statement` can link to its parts via `subject`, `predicate`, and `object`.

Table 3.12.: Core concepts and relations of RDF

Simple Knowledge Organisation System (SKOS) SKOS is a W3C recommendation (Miles and Bechhofer, 2008) for a *common data model for sharing and linking knowledge organization systems via the Web*. SKOS is meant to represent domain-free thesauri, term hierarchies and vocabularies. From a usage point of view, SKOS acts as a rather generic representation formalism itself. In PKM systems, a clear modelling of terms and their relations to other terms, might be required to allow long-term usage.

Concepts for identity and relatedness are important for PKM applications, where knowledge models emerge and only later one finds out that two concepts can be considered to denote more or less the same topic.

The concept and relation names in Tab. 3.13 and Tab. 3.14 are taken directly from the official namespace published at `http://www.w3.org/2004/02/skos/core#`. SKOS defines some inverse relations, those names are given in brackets.

Concepts	
RDF literal, Concept, ConceptScheme,Collection, OrderedCollection	
Relations	
From concept/to concept	*Semantics*
Concept/literal	A concept can have zero or one preferred labels "`prefLabel`" (→ Req. 17 [naming])
Concept/literal	SKOS allows for an unlimited amount of alternative names (→ Req. 18 [alias names]), which can be visible "`altLabel`" or hidden "`hiddenLabel`".
Concept/literal	One or several unique identifier "`notation`" per SKOS concept. Those identifiers are considered to be technical notations.
Concept/literal	Seven ways to annotate (→ Req. 23 [annotation]) a SKOS concept with an informal description, namely `changeNote`, `definition`, `editorialNote`, `example`, `historyNote`, `note`, and `scopeNote`.
Concept/ConceptScheme	`inScheme` — groups concepts to administrative units (→ Req. 15 [grouping])
ConceptScheme/Concept	`hasTopConcept` (inverse: `topConceptOf`) relates a ConceptScheme to a concept at the root of a concept hierarchy
Concept/Concept	`semanticRelation` models the existence of any kind of semantic relation between two concepts.
Concept/Concept	The generic relation `related` (symmetric) indicates that two concepts are related semantically, but *not* in a hierarchical fashion.
Concept/Concept	`broader` (inverse: `narrower`) allows arranging concepts in a non-strict hierarchy (→ Req. 21 [hierarchy]). Such a hierarchy may contain cycles. [23]

Table 3.13.: Core concepts and relations of SKOS (part 1)

Concepts	
RDF literal, Concept, ConceptScheme,Collection, OrderedCollection	
Relations	
From concept/to concept	*Semantics*
Collection/Collection, Collection/Concept	A collection is a grouping (→ Req. 15 grouping) within a ConceptScheme that can contain (`member`) any number of concepts or other collections. Such a nesting of collections imposes another kind of hierarchy (→ Req. 21 hierarchy).
Ordered-Collection/Concept	An ordered collection has a total order among concepts (→ Req. 19 order), which is represented in an RDF list (`memberList`).
Concept/Concept in another ConceptScheme	`broadMatch`, `narrowMatch` and `relatedMatch` model the same `semantics` across concept schemes as `broader`, `narrower` and `related` within a concept scheme.
Concept/Concept in another ConceptScheme	There are two constructs for similarity *between* concept schemes: The relation `exactMatch` denotes two concepts to be completely identical; this relation is transitive. For weaker similarity, `closeMatch` can relate two concepts. This is a non-transitive property to avoid a conceptual drift when propagated over many concept schemes.

Table 3.14.: Core concepts and relations of SKOS (part 2)

3.6.3. Common relations

This section briefly analyses common relations created by end-users in tools that allow creation of arbitrarily labelled relations.

Concept Map usage study

About 1400 concept maps created with *CMap tools* by 114 students have been analyzed by Marshall, Zhang, Chen, Lally, Shen, Fox, and Cassel (2003). Although the concept map system was pre-configured with a set of relations, the users chose to create their own link-types for over two thirds of the links. A total of ca. 40,000 links was created, of which Marshall et al. (2003, p. 9) found only 5,300 distinct link names. So on average, each link type has been used slightly over seven times. By manually categorising the links into 120 types, Marshall et al. (2003) found that over 50% of the links are hierarchical. Interestingly, the concept maps based on class lectures had a higher number of hierarchical links than those concepts maps created by students from content acquired from text books. This might indicate that more refined material has a more hierarchical structure. A knowledge model should be able to represent hierarchical links (→ Req. 21 hierarchy).

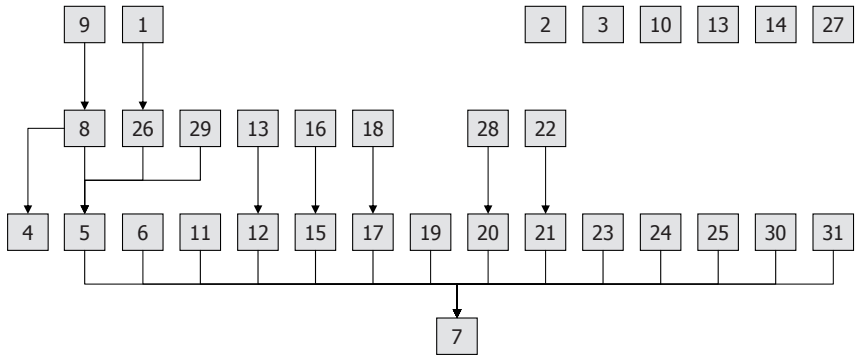


Figure 3.12.: Requirements dependency graph

3.7. Requirements summary

This section just briefly lists all requirements stated in the previous sections. As the requirement summary table shows (cf. Tab. 3.15), ultimately, a PKM tool must be a general purpose modelling tool. Fig. 3.12 shows the requirements dependency graph. Requirement 7 [addressability] is the pre-requisite for many other requirements. Requirement 5 [formal articulation] is necessary to achieve a certain level of automation in knowledge model tasks. Some requirements are completely orthogonal to others: 2 [super-set], 3 [fast entry], 10 [refactor], 13 [maintenance], 14 [easy to learn] and 27 [follow links and browse].

3.8. Conclusions

This chapter has presented a survey on PKM tools. Then a number of existing process models for PKM has been presented and a new model for describing the knowledge cue life-cycle has been developed. Based on the new model, a novel economic analysis of the different steps has been conducted. Survey, process model and economic analysis have been used to gather requirements for knowledge models. Additional known requirements from literature for knowledge models have been presented. Then a detailed analysis of the conceptual models embedded in popular tools used for PKM tasks led to requirements for relation semantics.

Requirement	**Description** (this is a requirement for . . .)
Economic Analysis	
1 auto-query	System should run queries automatically (tool)
2 super-set	Super-set of PKM data models (model)
Interaction for codify and augment process	
3 fast entry	Fast entry (tool)
4 informal articulation	Informal articulation (model and tool)
5 formal articulation	Formal articulation (model and tool)
6 granularity	User decides on modelling granularity (model and tool)
7 addressability	Entities need to be addressable (model and tool)
8 formality levels	Sim. use of different levels of formality (model and tool)
9 stepwise	Stepwise formalisation (model and tool)
10 refactor	Knowledge model refactoring (tool)
11 versioning	Versioning (model and tool)
12 context	Capture the context for cue creation and import (model and tool)
13 maintenance	Active assistance in maintenance tasks (tool)
14 easy to learn	Easy to learn (model and tool)
Ways of adding structure and formality	
15 grouping	Grouping of items (model and tool)
16 containment	Containment relationship (model and tool)
17 naming	Optional naming of knowledge cues (model and tool)
18 alias names	Alternative names (model and tool)
19 order	Order knowledge cues (model and tool)
20 (hyper)-links	Linking (model and tool)
21 hierarchy	Hierarchy (model and tool)
22 levels of detail	Sim. use of multiple levels of detail (tool)
23 annotation	Annotating content (model and tool)
24 tagging	Tagging (model and tool)
25 categories	Classifying items into categories (model and tool)
Interaction for retrieval process	
26 queries	Queries (model and tool)
27 follow links and browse	Browsing (model and tool)
28 inverse relations	Inverse relations (model and tool)
Expressivity	
29 domain-free	Flexible schema (model and tool)
30 transclusion	Transclusion (model and tool)
31 meta-modelling	Meta-modelling (model and tool)

Table 3.15.: Requirements summary

4. Conceptual Data Structures

This chapter presents the building blocks of the solution, called the *Conceptual Data Structures* (CDS). The logical structure of the chapter is depicted in Fig. 4.1.

The core ideas of CDS have been presented by Völkel and Haller (2009); Völkel, Haller, and Abecker (2007); Völkel and Haller (2006). The CDS data-model is originally based on the combination of the RDF data-model (cf. Sec. 2.6) with the way content is handled in the World Wide Web (cf. REST, Sec. 2.4). This has been published as Semantic Web Content Model (SWCM) at the I-Semantics conference (Völkel, 2007b). The ideas and proofs for representing RDF in a model where each literal has a URI are described in detail by Heitmann, Oren, and Völkel (2006).

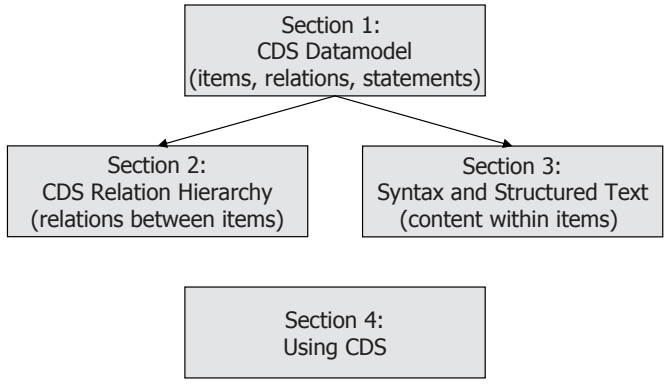


Figure 4.1.: Overview of Chapter 4

The main idea is to create a super-set (→ Req. 2 super-set) of conceptual models of common PKM tools via a two-layered strategy: A generic, expressive yet simple data-model (CDS Data Model ($\mathfrak{D}$), see Sec. 4.1) for the *representation* of knowledge models is presented. The data-model allows representing pieces of *content* connected by typed *relations*.

- The main concern addressed by the data-models is *expressivity*. It serves as a representational layer for any kind of content and structure in a knowledge model. It can be used without the other parts of CDS.

  As a result of the overall design, the data-model must address the requirements 17 naming, 6 granularity, 12 context, 26 queries, 31 meta-modelling, and 11 versioning.

- The Relation ontology (CDS Relation Ontology ($\mathfrak{R}$), see Sec. 4.2) is represented as a set of Relations in the data-model. This layer defines the semantics of relations. The Relation ontology addresses concerns of *scalability* as it allows handling a large number of relations in a systematic and simple way.

  $\mathfrak{R}$ has been designed to be compatible yet not depending on $\mathfrak{D}$. E: g. $\mathfrak{R}$ can also be used in RDFS or OWL.

  The Relation ontology is mainly responsible for requirements such as 19 order, 20 (hyper)-links, 21 hierarchy, 23 annotation, 24 tagging, and 25 categories.

- The inner structure of content is defined in a *Structured Text Interchange Format* (STIF). STIF (Sec. 4.3.1)) is a formal model of structured text. It is needed to define conversions from textual syntax to structured text. An actual textual syntax is presented as well.

  The main concern here is *re-use* of structures in text and of structured content from other sources.

- Finally, transformations from structured text *within one knowledge cue* to structures *between several smaller knowledge cues* are shown. The main goal of this is *lowering articulation costs*, especially for structures between knowledge cues.

CDS is a conceptual model (cf. 2.13) designed for end-users. It is described in terms of its entities and their semantics *for a user*.

Implementation aspects are discussed in Chapter 5. Since CDS is designed for knowledge modelling tools, some requirements on user interfaces are discussed as well. Prototypical user interfaces built on CDS can be found in Sec. 5.2, Sec. 5.3.1, and Sec. 5.3.2.

4.1. CDS data model

This section describes $\mathfrak{D}$ (the CDS data model) for representing personal knowledge models. The role of $\mathfrak{D}$ is to represent all kinds of knowledge cues. Sec. 4.2 describes a second layer on top of $\mathfrak{D}$: an ontology of relations ($\mathfrak{R}$). $\mathfrak{D}$ is described in terms of its entities and their semantics.

The data model allows representing knowledge cues and statements about them. All user-authored entities are called ITEMS and have an author, a creation date and a last change date. STATEMENTS are special ITEMS which additionally inherit the properties of triples. The model further supports the use of an inference engine. Inferred[1] or imported triples have no author as they might exist solely during a query, e. g., by backwards-chaining.

4.1.1. Informal description and design rationale

The basic idea of CDS is to represent knowledge cues either as *items* or as connections between such ITEMS. An ITEM is essentially an addressable, human-readable symbol. For teaching CDS, iconic representations have

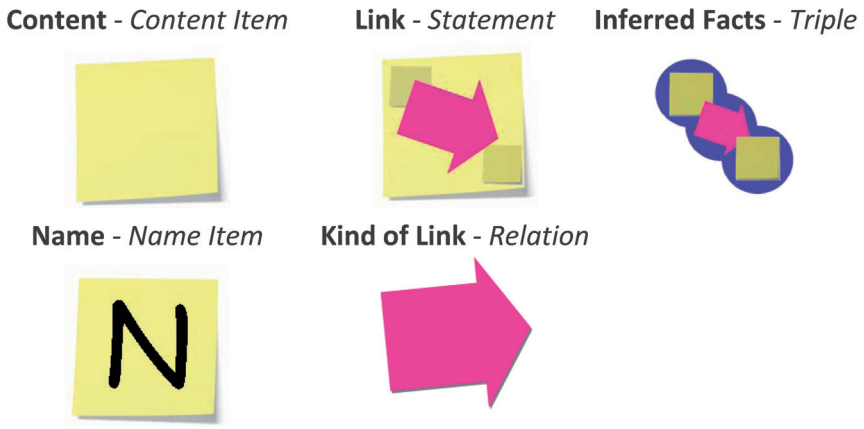


Figure 4.2.: Iconic representations for the five building blocks of the CDS data-model, as they have been used to teach CDS.

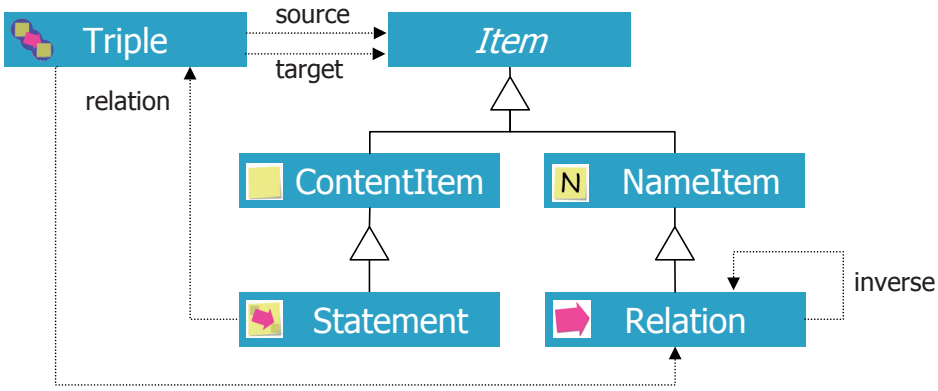
Overview

been developed (c. f. Fig. 4.2). Miniature versions of these images are used in the HKW user interface (cf. Sec. 5.2.1). Pras and Schoenwalder (2003) distinguish *information models* from *data models*. Information models are more abstract and contain no implementation details such as protocols or index structures. In this terminology, $\mathfrak{D}$ is an information model, despite its name.

$\mathfrak{D}$ can be described from different viewpoints. $\mathfrak{D}$ is designed as a model that allows users encoding their knowledge cues. For a user, a spreadsheet can be used as an (almost) infinite table of cells. Each cell can contain numbers, text, formulas and other data types. For using a spreadsheet, it is irrelevant whether cells are internally represented by one value plus a formatting instruction, e. g., to render "0,634" as "63,4%". In an analogue way, $\mathfrak{D}$ can be described on different levels of abstraction.

Fig. 4.3 shows a high-level conceptual view with abstract classes rendered in italics. This is the mental model a user has to learn before being able to use CDS. One immediate axiom from the figure is for example "each instance of a RELATION is also an instance of ITEM".

[1] Produced by deductive rules, see *Reasoning* in Sec. 2.6.



Legend: As usual in UML class diagrams, lines with a triangle denote inheritance, e. g., the type *Relation* inherits from *NameItem*.

Figure 4.3.: The data model in a nutshell: All *Items* are addressable (via a URI), *Triples* are not. *ContentItems* have a snippet of content. So do *Statements*, which additionally have a *Triple*. A *Triple* has a source *Item*, a target *Item* and a relation of type *Relation*. The content of a *NameItem* is a short, unique string (it's name). A *Relation* is a *NameItem* that has always a defined inverse *Relation*.

Fig. 4.4 shows the complete CDS data model ($\mathfrak{D}$). The remainder of this section describes $\mathfrak{D}$ from a user's perspective and explains top-down the design rationales. Sec. 4.1.2 gives a precise bottom-up definition.

Item

Items are the main building block of $\mathfrak{D}$. An Item is an addressable unit of content. Much different from other data models, $\mathfrak{D}$ has no way to represent content which is not addressable. This comes at the cost of managing a larger space of identifiers but has many positive effects for handling changes and synchronising state (cf. Req. 7 [addressability]). Comparing Items for equality within a Model is reduced to comparing their identifiers, which can be implemented much faster.

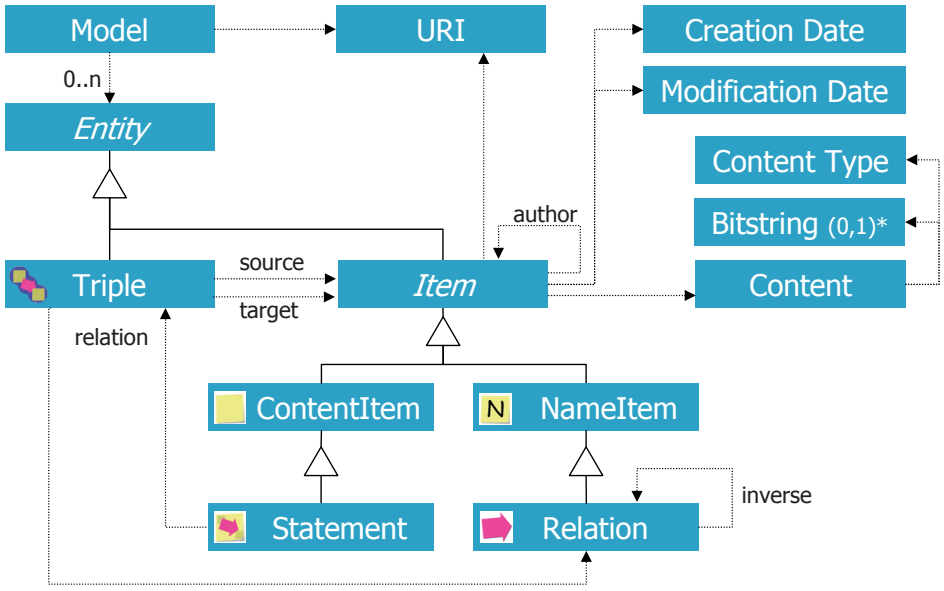
Conceptually, each Item corresponds to the notion of a *Resource* in the REST-sense (Fielding, 2000, p. 88):

> The key abstraction of information in REST is a resource. Any information that can be named can be a resource: a document or image, a temporal service (e. g., "today's weather in Los Angeles"), a collection of other resources, a non-virtual object (e. g., a person), and so on.

Metadata

Authorship

Each Item has a defined *author*, who is represented as another Item. If a user creates an Item x, she is the *author* of that Item. If now another user



Legend: As usual in UML class diagrams, lines with a triangle denote inheritance, e. g., the type *Triple* inherits from *Entity*. A simple arrow denotes association, read "a *Model* has zero to *n* *Entity*s. Lines without explicit cardinality information are 1:1 by default.

Figure 4.4.: The diagram shows a *Model* that contains a set of *Entity* objects. Both *Model* and *Item* have a *URI* by which they can be addressed. An *Item* has content (which in turn has a bitstring plus a content-type) and meta-data (author, creation and modification date).

changes the content of x, who should be the author of x? The initial creator or the one who changed its content? In CDS, the last person who changed an ITEM is considered the author. If all ITEM creation and change events are versioned, it is possible to answer questions like "which people ever changed the ITEM?". This authorship model is inspired from wiki pages, where it has been widely adopted.

Creation Date

Each ITEM has also a *creation date* and *modification data*. The creation date is set, when an ITEM is initially created. The modification date is updated each time the content of an item changes. The modification dates allow producing a list of all ITEMS that have been changed after a certain point in time. Such lists are used widely on the internet for so-called *newsfeeds* (or just *feeds*), e. g., on news sites or as a list of changes in a wiki.

URIs as identifiers

Addressing in $\mathfrak{D}$ is realised with *Universal Resource Identifiers* (URI, cf. Sec. 2.4). Each ITEM has a URI.

URIs are standardised, widely used in the world wide web as *resource locators* and in RDF as *resource identifiers*. Furthermore, there is an established social process for managing globally unique identifiers, i. e., by splitting the namespace hierarchically. For the `http` scheme a *domain name* is

used. Domain names are sold to individuals on established market place and their character sequence is part of the `http`-URIs. This reduces the problem of global uniqueness to uniqueness of identifiers managed by single individuals.

Content and content-types

The content of an ITEM is ultimately represented as a binary string in a computer. To render and manipulate this bitstring in a meaningful way, some metadata denoting the semantics of the bitstring is required. The world wide web (WWW) faced a similar problem in HTTP when a browser requests the content of a resource. In HTTP this is solved by sending a *content type* descriptor before sending the actual byte stream. To be compatible with existing approaches, $\mathfrak{D}$ uses the same approach and the same set of content identifiers as the WWW: MIME-types (Freed and Borenstein, 1996). MIME-types have originally been designed to denote the file type of email attachments and have subsequently been used in HTTP for the same purpose. MIME-types are standardised and there is an existing global registry, the IANA[2]. [3]

In CDS implementations, all textual content is represented in Unicode (Unicode, 2007). The characters must be encoded using the UTF-8 (Yergeau, 1998) encoding from unicode characters to bytes. This is the encoding most often used on the internet.

Naming

A simple collection of addressable content units is neither convenient to use nor expressive. Although a computer can address each ITEM, humans do not remember easily a large set of (sometimes cryptic) URIs. Instead, humans are good at mentally managing a (smaller) set of concept names and following associations between concepts. Therefore $\mathfrak{D}$ has a concept of *names*. To manage the names properly, they are represented as ITEMS, too. Consistent with Fig. 4.4, the super-type of all ITEMS with a content that denotes a name are called NAMEITEMS. RELATIONS are like NAMEITEMS, but have additional properties which are explained in Sec. 4.1.1. All descriptions of NAMEITEMS apply to RELATIONS as well.

The naming concept has been inspired from wikis in general (cf. Sec. 2.9) and from Wikipedia[4] in particular: e. g., a single technical namespace is used to manage collaboratively over one million articles in the English Wikipedia.

A NAMEITEM represents both a symbol for the computer (via its URI) and a term of the user's vocabulary (via its content). Using NAMEITEMS,

[2] `http://www.iana.org/assignments/media-types/` (accessed 06.01.2010)

[3] Note that MIME-types are used for semantics of *textual or binary content*, whereas XML Schema Types are used, e. g., in RDF and OWL to denote the semantics of text-based data values. A W3C Note describing an extension to XML Schema for specifying MIME-encodings is given in `http://www.w3.org/TR/xml-media-types/` (accessed 06.01.2010).

[4] `www.wikipedia.org` (accessed 06.01.2010)

an application can (and should) hide URIs completely from user interfaces. Instead, the unambiguous human-readable names should be used. From a user's perspective, a NameItem is simply a short unambiguous string. More on the usage of NameItems can be found in Sec. 4.4.

ContentItem

All Items where the content is not a name, but any other kind of content, have a common super-type called ContentItem. Statements have all properties of ContentItems, but the opposite is not true. Statements are explained in Sec. 4.1.1.

A ContentItem is the simplest kind of Item, it represents a piece of addressable content. From a user's point of view, a ContentItem represents a distinguishable symbol. The user *may* attach a human-readable representation to it, which can be text or, e. g., an image. This allows a ContentItem to act as a knowledge cue. If a ContentItem has no content, the meaning for a user is encoded solely in the way this ContentItem is used, e. g., connected to other Items.

Examples for content of a ContentItem are: A sentence like "John was the best player in his team in 1996", a single word like "exothermic", a paragraph, the whole thesis that you are reading just now, a figure from this thesis, or even an audio or video sequence. Additionally, any content one can find on the web could be used as the content of a content item.

NameItem

There are a number of constraints for NameItems:

- The content type for all NameItems is always fix (c_{NAME}). Consistent with MIME-types, c_{NAME} should be represented as "`text/plain+name`".
- The content of NameItems can never be empty. This constraint simplifies handling of NameItems in user interfaces. Pragmatically, user interfaces should also discourage names containing, e. g., only whitespace.
- The name should not contain formatting instructions to be easy to type and compact to display.
- As stated in Sec. 4.1.2, the name must be *unique* within a Model. This allows comparing NameItems by URI *and* by name. Most importantly, this allows mapping human-readable names unambiguously to machine-processable URIs.

Creation of a NameItem is cognitively more costly than creating a ContentItem with the same content, because the content of a NameItems must be unique within a model. Therefore a user interface might have to tell a user that a certain name is illegal (e. g., because it contains formatting instructions or line-breaks), or simply that it is in use already. If the name is in use, the user must decide if the existing and to-be-created NameItems

represents the same concept or whether one of the two needs to get a different name to reflect a more precise modelling of the world. If the user first creates a ContentItem and later converts it into a NameItem, then this an operation in the process of stepwise formalisation.

For a user, a NameItem represents a symbol. In contrast, a ContentItem can represent larger text snippets or even non-textual content such as images. Both kinds of Items are knowledge cues.

NameItems resemble more file names, folder names, person names, location names, tag names, keywords, or paper titles. They are so short that they can act as a reference handle for other Items, linked to them via Statements.

ContentItems resemble more file contents, e. g., a picture of a person, the full text of a document, etc... A typical text document has a file name and a file content. In $\mathfrak{D}$, the file name would map to a NameItem, which is connected via some relation to a ContentItem, in which the file contents would be represented. This mapping allows, e. g., to have several different NameItems to be connected to the same ContentItem, which would represent a document with multiple names.

Several alternative definitions for naming have been evaluated. The name spaces for NameItems and Relations can be united, as it is the case in $\mathfrak{D}$, or they could be separated. From a user's point of view, a single namespace is easier to manage. First of all, there is no concept of name spaces to be learned. Second, in user interfaces which offer auto-completion there is no need for further visual clues to distinguish between a proposed Item "A" as an Item or "A" as a Relation. Third, using a single namespace the user can introduce the concept of name spaces himself by using syntactic conventions on the names, e. g., by prefixing Relations with `rel:`. Therefore $\mathfrak{D}$ uses a single name space for NameItems and Relations.

Examples for NameItems are "Heiko Haller", "FZI", "FZI Karlsruhe", "Heiko", "My Pictures", "Evaluation of the central effects of alcohol and caffeine interaction", etc.

Triples as links and formal statements

So far only atomic entities have been introduced. Mappings between machine-readable identifiers (URIs), content units (bit-strings) and human-readable names have been established.

Triples represent both labelled links as well as formal statements. The difference lies in the used Relations. The semantics of Relations are defined in Sec. 4.2. In $\mathfrak{D}$, Triples are just structural elements, connecting Items in a knowledge model. Each Triple consists of

source This is an Item where the link starts. In RDF, this is called the *subject* of a triple.

relation This is an Item that represents the *kind of relation*. One kind of relations is simply *hyperlink with no further semantics*, other Items can represent formal relations such as *sub-type-of*. Not all Items can be used as the Relation. Restrictions for relations are motivated

and explained in Sec. 4.1.1. In formal facts, this ITEM is the *predicate* or *property* of a formal fact.

target This is an ITEM representing where a link points to. In formal facts, this is the *object* of the formal fact.

There is a problem associated with encoding knowledge in TRIPLES: The "direction" in which a TRIPLE was stated does not constitute to its truth, but has implications on queries. As an example, consider a person named *Dirk* working for *SAP*. One way to state this in a TRIPLE is:
([Dirk], [works for], [SAP])
Another way to state exactly the same fact would be:
([SAP], [employs], [Dirk]).
When the user later has the information need to find out where Dirk works, she might verbalise it at "Where does Dirk work?" and consequently try to formulate a query starting with the *Dirk*-ITEM. If she has stated the knowledge as ([SAP], [employs], [Dirk]) she will not find an answer.

The solution offered by $\mathfrak{D}$ is to make the connection between the RELATION (here: "`works for`" and "`employs`") explicit. Details on RELATIONS are explained in the next section.

Inverse RELATIONS and inverse TRIPLES work together. Each TRIPLE has the structure (ITEM, RELATION, ITEM). A TRIPLE `(source, relation, target)` implies a TRIPLE `(target, inverse relation, source)`.

Two TRIPLES are considered equal if their components are equal. Two ITEMS are equal, if their URIs are equal.

Relation

The middle part of a TRIPLE is called the *relation of the triple*. RELATION is also the name for the type of ITEMS that may be used as the relation of a TRIPLE. There are several desirable features for a RELATION:

- It should be possible to annotate RELATIONS in $\mathfrak{D}$– therefore a RELATION should be a kind of ITEM, too.
- It should be easy to select a RELATION by its unique name – therefore a RELATION is a sub-type of NAMEITEM.
- As explained in the last section, each RELATION should have an *inverse relation*.

Inverse relations

Each relation has an inverse relation (via a bijective function r in Sec. 4.1.2). Like all ITEMS, each relation has a URI, so does the inverse relation. The inverse relation of a relation may be itself. This is required to model symmetric relations.

Symmetric relations

Examples for RELATIONS are "works for", "is located in", "knows", "should be explained before", "belongs to", "see also", "has wiki article", "supports", etc.

Statement

STATEMENTS are ENTITIES which connect ITEMS. Each STATEMENT plays a dual role as addressable ITEM and as a TRIPLE, connecting other ITEMS. Two statements with different URIs might state the same triple.

Annotating statements

A statement is itself an ITEM, thus the user can annotate statements as well. This adds a complete level of expressivity to the MODEL. As a result, a user can annotate and link *any* ITEM in the model. E. g., she can link from a STATEMENT to a CONTENTITEM which includes a discussion for the design rationale of the STATEMENT. This is a desired feature for technical models or formal argumentation.

Binary vs. N-ary relations

Why are only binary relations allowed, i. e., why can a statement not have several sources and targets like, e. g., links in XLink (Orchard, Maler, and DeRose, 2001)? First of all, 3-tuples are believed to be expressive enough to encode any order of relations (Dau and Correia, 2006), (Peirce, 1958, 3.483, 1). Second, triples comply better with existing semantic web technologies, i. e., the RDF data model, and hence are better suited for interoperability with existing tools.

More on semantics of RELATIONS is explained in Sec. 4.2.

Examples for STATEMENTS are "Process control block" "has definition" "..." or "Heiko" "knows" "'Max".

Entity

An ENTITY is either an ITEM or a TRIPLE. This super-concept just denotes any kind of object that can be part of a model. All ENTITIES are used to represent knowledge cues. Note how ENTITIES differ in the way they can be addressed:

Triples cannot be addressed in any way, besides completely enumerating their content.

Items can be addressed by their URI.

NameItems and Relations can be addressed both by URIs and by human-type-able names.

Model

A MODEL is a container for ENTITIES. It is important to have a MODEL-concept in order to model different world views or simply different use cases. In this respect a MODEL is much like a file or database. Only the existence of multiple sets of ENTITIES allows, e. g., to compare such sets.

A Model has a URI, which makes it possible to uniquely identify models. Additionally, this simplifies the mapping of MODELS to RDF (cf. Sec. 2.6). RDF has the notion of *data sets* in SPARQL (Prud'Hommeaux et al., 2007). Another name for the same concept in RDF is *Named Graphs* (Carroll et al., 2004).

Summary

An ITEM is the basic building block to represent either a name or another unit of content. TRIPLES, which are the only ENTITIES in a MODEL that are not ITEMS, connect ITEMS. A STATEMENT is an addressable TRIPLE which can additionally have a content unit attached to it. Referring to sketches, ITEMS represent the nodes and STATEMENTS represent arrows between them. The RELATION of a STATEMENT indicates the *kind* of arrow. The content of a STATEMENT can be considered a label for an individual arrow. A TRIPLE is in this context just the fact that two ITEMS are connected via a certain RELATION. Two TRIPLES stating the same are *the same* TRIPLE. A TRIPLE has no content. Two STATEMENTS stating the same, are different STATEMENTS with different identity and possibly different content. An alternative view, emphasising different properties of objects is given in Table 4.1.

Datamodel comparison

4.1.2. Formal definition

The formal definition consists of atomic entities (NAMEITEMS, CONTENTITEMS) on top of which compound entities (RELATIONS, TRIPLES, STATEMENTS) are successively defined.

Atomic entities The building blocks of CDS are addressable content units and unique names.

Content Units

Let B (binary) be the infinite set of all possible words over the alphabet $\{0,1\}$, i. e., $B = \{0,1\}^*$.
Let C_{TYPE} be a finite set of *content types*. A *content type* denotes a description how a given bit-string should be *interpreted*, e. g., as characters of a string, pixels in an image or parts of an executable program. For now it is sufficient to define that a program can interpret bit-strings for some *content types*. Let c_ϵ be the empty content type for content of zero length.

A pair of a bit-string and a content-type is a *unit of content*. The set of all content units is $B \times C_{TYPE}$.

Let F be an arbitrary index set to denote unique identifiers, e. g., the natural numbers. F represents the class CONTENTITEM in Fig. 4.4. Several CONTENTITEMS or STATEMENTS can have the same content. Let f be a function $f : F \to B \times C_{TYPE}$ that assigns to each CONTENTITEM a content unit.

Names

Let Q be a finite set of characters.
Let A be the infinite set of all possible words over the alphabet Q.

The set A represents *names* in the CDS data model ($\mathfrak{D}$). More precisely, the set A represents the class NAMEITEMS in Fig. 4.4, i. e.,

each element in A is a different member of the class NAMEITEMS, due to the uniqueness of names.

Compound entities Here the CDS entities are described. Note that the names refer to CDS names introduced in the beginning of this chapter. E. g., RELATION refers to a CDS concept and not a relation in the mathematical sense.

Relation

Let R be the set of all RELATIONS, which is defined as a subset of names A, i. e., $R \subseteq A$. Let r be a bijective function $r : R \to R$ that maps each RELATION to an *inverse* RELATION, i. e., $\forall x \in R : r(r(x)) = x$. As a simplification of notation, "-x" is also used to denote $r(x)$.

NameItem

The set N of all NAMEITEMS is defined as $N = A \setminus R$.

Statement

Let $S \subset F$ be the set of all STATEMENTS.
Let T be a set of TRIPLES, which are defined shortly. Due to the self-referential nature of $\mathfrak{D}$ [5], the term STATEMENT needs to be defined first.
Let s_{triple} be a function that assigns to each STATEMENT a TRIPLE, $s_{triple} : S \to T$. s_{triple} is not injective, i. e., the same triple can be stated by different STATEMENTS.

ContentItem

Let C be the set of all CONTENTITEMS with $C = F \setminus S$.

Item

Let I be the set of all ITEMS which is defined as the set union of CONTENTITEMS, NAMEITEMS, RELATIONS and STATEMENTS. This is by definition equivalent to the set union of all *names* (A) and all *content units* ($B \times C_{TYPE}$). Hence, $I = A \cup (B \times C_{TYPE})$.

Triple

Let T be the set of all TRIPLES. Formally, $T = I \times R \times I$.
The components of a TRIPLE are named from left to right: *source*, *relation*, and *target*.

For every TRIPLE (s, p, o), an inverse TRIPLE $(o, -p, s)$ is *inferred*. Practical considerations when the inference engine is running and how it can be constructed are discussed in Sec. 5.1.

Formally, given a finite set of TRIPLES $M_{explicit} \subset T$ the set $M_{inferred}$ of inferred TRIPLES is defined as: $\forall (s, p, o) \in M_{explicit} : (o, -p, s) \in M_{inferred}$.

[5] A STATEMENT connects ITEMS, but a STATEMENT is also itself a kind of ITEM. Such a self-referential structure is quite common for meta-models.

$M_{true} = M_{explicit} \cup M_{inferred}$ is the set of all *true* TRIPLES in a MODEL M.

Entity

Let E, the set of all ENTITIES, be the set union of all ITEMS and all TRIPLES, i. e., $E = I \cup T$.

Model

A MODEL M is a *finite* subset of ENTITIES $e \in E$, i. e., $M \subset E$.
The subset of all TRIPLES in M is called $M_{explicit} = M \cap T$ for which a corresponding set $M_{inferred}$ is defined.

The space of possible MODELS M can be summarised as:
$M \subset E = I \cup T = I \cup (I \times R \times I)$
with $I = A \cup (B \times C_{TYPE})$ and $R \subset A$.

A choice of a character set Q, a set of content types C_{TYPE} defines the space of possible models. With Unicode as the character set Q and the MIME-types as C_{TYPE}, the set of possible models is fully specified.

Each MODEL M can thus be seen as a tuple of *finite* sets
$(N_M, R_M, C_M, S_M, T_M)$ with

NAMEITEMS	$N_M \subset N$
RELATIONS	$R_M \subset R$
CONTENTITEMS	$C_M \subset (B \times C_{TYPE})$
STATEMENTS	$S_M \subset S \subset (B \times C_{TYPE})$ with $s_{triple} S_M \rightarrow T_M$
TRIPLES	$T_M \subset T = I \times R \times I$

Identifiers

Let UID be an infinite set of identifiers.
A bijective function *uid* assigns to each MODEL and each ITEM in a MODEL a globally unique ID, i. e., $uid : M \cup E \rightarrow UID$ and $\forall x, y \in (M \cup E) : uid(x) = uid(y) \Rightarrow x = y$.

Helper functions for content and content types Let c be a function that returns the *content* of an ITEM. Formally,

$$c : I \rightarrow A \cup B \text{ and } c(x) = \begin{cases} x, & x \in A \\ b, & x = (b, t) \in (B \times C_{TYPE}) \end{cases}$$

Let c_{type} be a function that returns the *content type* of an ITEM.
Let $c_{NAME} \in C_{TYPE}$ denote content with *name* semantics.
Then $c_{type} : I \rightarrow C_{TYPE}$ is defined as

$$c_{type}(x) = \begin{cases} c_{NAME}, & x \in A \\ t, & x = (b, t) \in (B \times C_{TYPE}) \end{cases}$$

Let $c_{BINARY} \in C_{TYPE}$ represent generic binary content. The content is a bit-string. The semantics are the same as those for the MIME-type `application/octet-stream`, the most generic binary content type defined by Freed and Borenstein (1996).

Metadata Each ITEM $i \in I$ has a *creation date*. Formally:
Let $\mathcal{T}$ be a totally ordered set ($\forall t_1, t_2 \in \mathcal{T} : t_1 < t_2$ or $t_2 < t_1$ or $t_1 = t_2$). Members of $\mathcal{T}$ denote *discrete time points*. There is no metric defined on $\mathcal{T}$.
Let $m_{creation}$ be a function that assigns to each ITEM $i \in I$ a *creation date*, i. e., $m_{creation} : i \mapsto t_k \in \mathcal{T}$.

Each ITEM $i \in I$ has also an *author*. Let m_{author} be a function that assigns to each ITEM an *author*. Authors (a) are represented as ITEMS. Formally: $m_{author} : i \mapsto a \in I$.

Built-in item to represent the "system" author

Axioms As each ITEM has an author, also each ITEM representing an author needs to have an author. If a user u creates an ITEM x, u is the author of x. Who is the author of the ITEM u? To avoid recursion, the MODEL contains a built-in CONTENTITEM x representing the data model $\mathfrak{D}$ itself. $\mathfrak{D}$ is the author of all system-created ITEMS, such as u.

For the ITEM $\mathfrak{D}$, the following properties are defined. The author of $\mathfrak{D}$ is $\mathfrak{D}$ itself. The creation date of $\mathfrak{D}$ is defined as the publishing date of this thesis. Formally, $UID(x)$ =`http://www.semanticdesktop.org/ontologies/2007/09/swcm#author:system`.

Root item

When a user opens a MODEL to start associative navigation, she must have a starting point. Most navigational user interfaces focus always on one certain entity. E. g., a file system explorer shows by default the folder "My Files". For each domain name, a browser tries first to load a file called "index.html". A similar convention of a "root" ITEM also exists in CDS. Each MODEL should contain a *root* NAMEITEM x with the following properties $c(x)$ ="RootItem", $c_{type}(x) = C_{NAME}$, $uid(x)$ =`cds:rootItem`, $m_{autor}(x)$ =$\mathfrak{D}$.

Summary The Table 4.1 shows a comparison of the different $\mathfrak{D}$ entity types. The main distinction to keep in mind is between ITEMS representing basic building blocks (NAMEITEMS, CONTENTITEMS, and RELATIONS) and those representing connections between the basic building blocks (TRIPLES and STATEMENTS).

Formal interpretation of a CDS model

This section defines the formal interpretation of the set of its TRIPLES M_T, which has been defined in Sec. 4.1.1 and formalised in Sec. 4.1.2.

Let V, the *vocabulary* used in M_T, be the set of all ITEMS that occur in M_T either as a source, relation or target. Note that this must be a subset of all ITEMS used in M (called M_I), hence $V \subseteq M_I$. Formally, $V = \{s|(s,r,t) \in M_T\} \cup \{r|(s,r,t) \in M_T\} \cup \{t|(s,r,t) \in M_T\}$.

An interpretation I of the vocabulary V is defined analogous to the definition in RDF by Hayes (2004)). It defines extensional semantics, i. e., a relation and the pairs of elements it relates are considered to be different objects. An interpretation H is defined by:

- A non-empty set U, the universe of the interpretation H

	ContentItem	NameItem	Relation	Triple	Statement	Model
URI UID	yes	yes	yes	–	yes	yes
Content c	any	name only	name only	–	any	–
Content-Type c_{type}	any	fix[a]	fix[a]	–	any	–
Meta-data such as author and creation date	yes	yes	yes	–	yes	–
Association to source, relation, target	–	–	–	yes	yes	–

[a] Content-Type = "`text/plain+name`"

Table 4.1.: Comparing objects in the CDS data model by properties

- A sub-set of U, which is called the set of properties P with $P \subseteq U$
- A mapping $IR_{EXT} : P \to \text{powerset}(U \times U)$,
  i. e., the set of sets of pairs (x, y) with x and y in U.
- A mapping S from Items to the universe, $S : V \to U$

Let V_R be the vocabulary of Relations, i. e., $V_R = V \cap R$. For each Relation x in V_R, the Relation extension IR_{EXT} of $-x$ must contain exactly the same set of pairs as IR_{EXT} of x, but each pair in reverse order. Formally, $\forall x \in V_R : \forall (a, b) \in IR_{EXT}(x) : (b, a) \in IR_{EXT}(-x)$. This definition is analogue to the definition of inverse relations in OWL[6].

For each Item $x \in V : H(x) = S(x)$, i. e., the interpretation of x is defined by the mapping function S.

For each Triple $(s, r, t) \in M_T$:

$$H\left((s, r, t)\right) = \begin{cases} \text{true}, & \text{if} \quad H(r) \in P \wedge (H(s), H(t)) \in IR_{EXT}(H(r)) \\ \text{false}, & \text{otherwise} \end{cases}$$

A whole Model is said to be *true*, if it contains only *true* triples.

This definition of interpretation still leaves much freedom. Only when the semantics of particular vocabulary Items is considered and formalised, too, more restricted interpretations can be defined. In CDS, semantics of the vocabulary is defined in $\mathfrak{R}$, in Sec. 4.2.

Simplifying the state-space

The state-space $E = I \cup T = (A \cup B \times C_{TYPE}) \cup T$ can be mapped to an isomorphic structure with less sets involved. This simplifies implementing $\mathfrak{D}$ in an application programming interface (API). The basic idea is to unify

[6] `http://www.w3.org/TR/owl-ref/#inverseOf-def` (accessed 06.01.2010)

the set of character strings A with the set of binary strings B by means of an *encoding* and *decoding* function.

- Let $enc : Q \to B' \subseteq B$ be an *encoding* function that maps characters to bit strings.

- Let $dec : B' \to Q$ be the inverse *decoding* function of *enc*.
  It holds that $\forall q \in Q : dec(enc(q)) = q$. Note that in most computer systems for most configurations, not all binary strings can be interpreted as characters in a given encoding, so usually $B' \subset B$. This is especially true for Unicode and the encoding UTF-8 (cf. Sec. 4.1.1).

- Let $c_{NAME} \in C_{TYPE}$ denote content with *name* semantics with the additional constraint that only NameItems and Relations use the content-type c_{NAME}, i. e., it must hold that $\{x | c_{type}(x) = c_{NAME}\} = A$.

This mapping allows simplifying the set I from $A \cup (B \times C_{TYPE})$ to $I = B \times C_{TYPE}$. This mapping is exploited later in the API design in Ch. 5.

4.1.3. Queries

This sections explains typical queries, i. e., read-operations, on a Model. Operations that change the state are explained in Sec. 4.1.4.

Content display A typical read operation, which is likely not perceived as a query in the mind of a user, is to retrieve (and likely display) the content of an Item. For an Item x, this can be done by calling $c(x)$ and decoding the obtained bit-sequence according to the semantics defined for the type $c_{type}(x)$.

Full-text search User-perceived queries are either full-text search over the content of Items, possibly restricted by Item type (ContentItem, NameItem, Relation, Statement) or by content type (name, STIF[7], other . . .). Such queries can only be answered if the CDS-based tool using $\mathfrak{D}$ can understand the semantics of the content types and successfully extract full-text (in characters) from the raw bit string. E. g., extracting the textual content from a slide show stored as the content of a ContentItem. Queries for NameItems and Relations are always possible, as the mapping from the bit-string to a character sequence to match against is defined by $\mathfrak{D}$ (see Sec. 4.1.2).

Structural queries The next level of expressivity are *structural queries*, i. e., queries over the graph induced by the Triples. There are eight basic patterns to filter the set of all triples (c. g. Tab. 4.2), where a star denotes a wild-card to match any Item.

[7] Structured Text Interchange Format, introduced in Sec. 4.3.1

source	relation	target
s	r	t
s	r	*
s	*	t
s	*	*
*	r	t
*	r	*
*	*	t
*	*	*

Table 4.2.: All triple query patterns

The pattern `(source, relation, target)` only tests if such a Triple exists or not. It can be used to query the Model for the existence of certain facts. The other seven patterns result in a set of matching Triples. Such a result set can be used in several ways. Triple result sets can be rendered of projected to sets of Items. As an example, a query such as `(Dirk, worksFor, ⋆)` can be projected to a set of employers of Dirk.

Browsing A special case of structural queries is *browsing* a Model. Assume the user starts with a certain Item x, then the following queries can be posed automatically: `(x,⋆,⋆)` and `(⋆,⋆,x)`. If x is a Relation, the additional query `(⋆x⋆)` can be posed, otherwise the result of this query is empty. The queries `(x,⋆,⋆)` and `(⋆,⋆,x)` can be reduced to just `(x,⋆,⋆)` as the other query returns only redundant information because of the inferred inverse triples (see Sec. 4.1.2). The result of the pattern `(x,⋆,⋆)` can be rendered as a segmented list of Items (the targets), sorted by Relations (the relation part of a result tuple). The user can then select on of the Items from the target position of an Item and continue browsing. As Relations are Items, too, they can be browsed in a similar fashion. For Relations the user interface can additionally show a list of all Triples where the Relation x is used.

Semantic queries In systems using $\mathfrak{D}$, the semantics of inference may be extended by applying semantics to Relations. As a result, the set of inferred triples, $M_{inferred}$, can contain many more Triples which are the result of sophisticated inference algorithms. The user should have the option to decide if she wants to query only the explicitly stated Triples, i. e., those in $M_{explicit}$ or the complete set M_{true}.

Complex queries More complex queries can be formulated by combining basic triple query patterns using variables and join operations. The definition and execution of such queries is outside the scope of the definition of $\mathfrak{D}$. In implementations, the constructs offered by the SPARQL query language (Prud'Hommeaux et al., 2007) should be reused. E. g., SPARQL defines conjunctive queries such "`?x :livesIn`

`:Karlsruhe. ?x :worksAt :SAP`" which would return all resources `?x` for which both a triple "`?x :livesIn :Karlsruhe`" and a triple "`?x :worksAt :SAP`" exists. SPARQL also offers constructs to filter results sets according to some criteria or to create the union of two results (disjunctive queries).

4.1.4. Operations

There are several operations a user can perform to change the state of her MODEL. They can be grouped into content operations, add/remove operations for each ITEM type and conversion operations between ITEM types.

Each operation that changes that state of a MODEL has an *author*, which is an ITEM that stands for the real-world entity that is the originator of that change operation.

Adding and removing ContentItems

Adding a ContentItem The simplest possible operation is creating a new CONTENTITEM in a MODEL. The new CONTENTITEM must have a new, unique URI. After this operation, there is a CONTENTITEM x for which $x \in M$. If no content has been specified on creation, the content is empty, i. e., $c_{type} = c_\epsilon$. The creation date is set to the time of ITEM creation. The author is an ITEM representing the creator of the ITEM.

Removing a ContentItem Removing a CONTENTITEM from a MODEL can cause to recursively remove further ITEMS. If a CONTENTITEM is removed, first all STATEMENTS using this CONTENTITEM as source or target need to be removed. See *Adding and Removing Statements* for details on removing statements. Finally, the CONTENTITEM itself is removed from the model.

Content operations

Changing the content A typical operation is to edit the content of an ITEM. This results in a new content unit attached to the ITEM, together with the author and time of change - recorded as creation date of this content unit.

When the content of a NAMEITEM or RELATION is changed – which is the same as to say that the content-type is c_{NAME} – then the same rules apply as for creating such units of content from scratch: The new name may not be empty and the MODEL may not contain another NAMEITEM or RELATION with the new desired name.

Changing the content type Changing the content type is not a useful operation. A user should always supply a complete unit of content consisting of *bit-string*, *content-type* and *author* to edit an ITEM.

The creation date should be set to the time when the operation is performed on the MODEL.

Formally, it is not possible to change the author or creation date without changing the content. However, it is possible to "touch" an ITEM by editing it and setting the same content. This results in a new content (although with the same bit-string) and a new author and a new creation date, reflecting the time of the edit operation.

Adding and removing NameItems

Adding a NameItem A new NAMEITEM must ensure that name remains unique throughout the MODEL, i. e., that $n(x)$ remains injective. Furthermore, the new NAMEITEM must also use a new, unique URI.

Removing a NameItem Again, first all STATEMENTS using this NAMEITEM need to be removed. See *Adding and Removing Statements* for details on removing statements. If no STATEMENT uses the NAMEITEM, it is simply removed from the MODEL.

Adding and removing relations

Adding a Relation A new RELATION must have two new unique names and two unique URIs. It is possible to re-use existing NAMEITEMS in the process of creating a RELATION. The advantage is that those NAMEITEMS have already the feature of unique names and URIs so that this check can be omitted. Therefore it is cognitively cheaper to create new RELATIONS out of existing NAMEITEMS.

If a RELATION is created using existing NAMEITEMS, then those NAMEITEMS are converted to RELATIONS. Existing TRIPLES and STATEMENTS which uses these NAMEITEMS are kept intact and now link to RELATIONS of the same name instead.[8]

Creating a RELATION always requires creation of an inverse RELATION. When the user creates a new RELATION p she also creates a RELATION $-p$. It holds: $r(p) = -p$ and $r(-p) = r$. For symmetric RELATIONS, $p = -p$.

Removing a Relation Removing a RELATION, no matter how it was created, removes it from the MODEL. I., e. if the RELATION was created using NAMEITEMS, those NAMEITEMS do not reappear ever again. Before the RELATION is removed, all STATEMENTS using this RELATION as their source, relation or target part, are removed. See *Adding and Removing Statements* for details on removing statements.

[8] Example: If there was a STATEMENT "Heiko" "enjoys his" "drink" where "drink" was a NAMEITEM; and now "drink" is converted to a RELATION; then the existing STATEMENT is kept intact. Only the type is changed from NAMEITEM to RELATION and there is now an inverse RELATION, e. g., "drunk by".

Adding and removing explicit triples

Adding an explicit triple The identity of a Triple is defined by its source, relation and target. If a Triple (a, b, c) exists already in the Model adding it a second time does not change the state of the Model in any way. Otherwise the new triple is added to the Model and the corresponding inverse Triple is added to $M_{inferred}$.

Removing an explicit triple Removing a Triple is much simpler than removing any kind of Item. As the Triple cannot be used in other Entities, the Triple to be removed is simply removed from the Model.

Adding and removing statements

Adding a Statement Statements, like any other Item, are identified by their URI. Therefore adding a Statement to a Model requires only that its URI has not been used before in this Model. The source, relation and target must be any existing Item in the Model. Together with other definitions, this ensure that no Statement can point to itself or in other ways create cyclic structures.

Removing a Statement A user can choose to remove a Statement or the removal of a Statement can be caused by the desired removal of Item using this Statement as their source, relation or target. Note that also Statements can be used as the source or target of other statements. Therefore, before a Statement x can be deleted, all other Statements which use x as source or target must be deleted. As only the creation of Statements referring to existing Items is allowed, the recursive process of deletion always terminates. Finally, the Statement is removed from the Model.

Changing statements

Changing the source of a Statement The source of a Statement may be changed. However, to avoid creating cycles the new source of a Statement may only be a NameItem or a ContentItem. This restricts the flexibility of the model to ensure cycle-free structures in a simple way. Theoretically, the new source could also be another Statement as long as no cycles are directly or indirectly created. As such cycle-checks are rather expensive in algorithmic terms and most users are unlikely to easily understand the resulting warnings, this is not allowed in $\mathfrak{D}$.

Changing the relation of a Statement The relation of a Statement can be changed to any other Relation. In fact, this is one of the most important ways to gradually add more semantics to a Model. Of course, switching from any Relation back to the "no semantics" Relation is possible. Details about the Relation ontology are explained in Sec. 4.2.

Changing the target of a Statement For the target of a Statement the same rules apply as for changing its source: Only ContentItems and NameItems are allowed to guarantee cycle-free-ness in an easy to understand and easy to implement way.

Conversion operations Conversion operations retain all Statements using the Item as source or target.

Converting a NameItem to a ContentItem This is always possible. The new ContentItem has the same character string as the NameItem. The new content type is set to c_{STIF}, a content type to denote textual content. STIF is introduced and defined in Sec. 4.3.1. Note that this content type allows, but does not require, formatting instructions.

Converting a ContentItem to a NameItem This conversion is not always possible, as the ContentItem might currently contain binary content or character content with formatting instructions. Even if both is not the case, the ContentItem might have a content which is already used as a name for another existing NameItem or Relation. There is no predefined way to obtain a new name from the content of a ContentItem. To convert a ContentItem to a NameItem, the user must supply a name which is not already used for any other NameItem or Relation.

Converting a symmetric relation to an asymmetric relation This conversion creates a new Relation which serves as the inverse Relation of the existing Relation. All existing Statements remain unchanged.

Converting an asymmetric relation to a symmetric relation For a Relation p and its inverse Relation q, this operation requires to replace all occurrences of q by p and to set the inverse of p to p itself. The Relation q is deleted in the process.

Higher-order operations Based on these primitive operations, higher order operations such as copy and move can be defined. A copy operation simply re-creates an existing set of Items

4.2. CDS relation ontology

This section presents a carefully selected set of Relations. The main intention is to use these on top of $\mathfrak{D}$, which is described in Sec. 4.1. However, the resulting relation ontology is generic enough to be used in other formalisms, such as OWL (Schreiber and Dean, 2004), the NEPOMUK Representation Ontology (NRL) by Sintek et al. (2007), Topic Maps (Durusau and Newcomb, 2005), or Concept Maps (Jüngst, 1992).

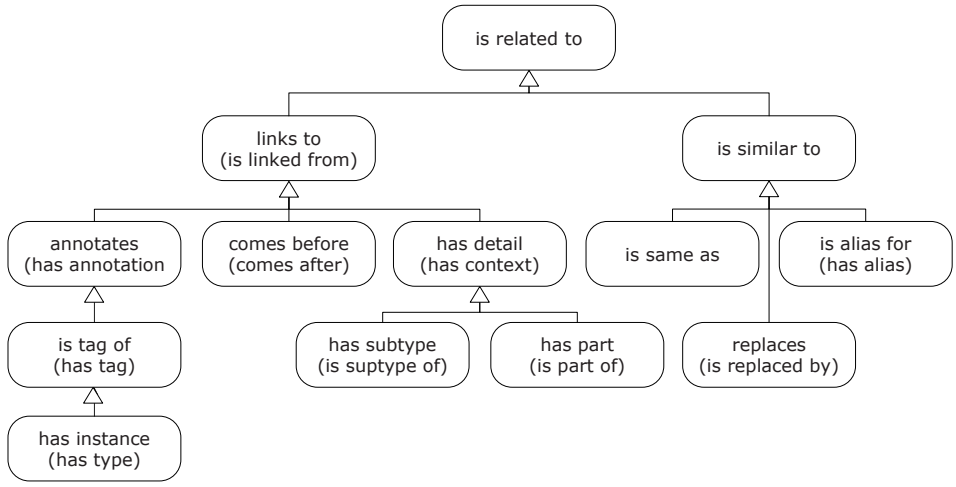


Figure 4.5.: The complete CDS relation subsumption hierarchy

Each RELATION is defined with a corresponding *inverse relation*, as required by $\mathfrak{D}$, and as supported by OWL, NRL, and Topic Maps. Furthermore, the RELATIONS in CDS are arranged in a subsumption hierarchy, i. e., RELATIONS with more specific semantics imply RELATIONS with more general semantics. This semantic construct of sub-RELATIONS is also present in OWL and NRL. The resulting artefact is the *CDS Relation Ontology*, or $\mathfrak{R}$ for short.

$\mathfrak{R}$ is the result of an extensive analysis of existing conceptual knowledge organisation models. The complete analysis is in Sec. 3.5. First all RELATIONS are introduced informally, then Sec. 4.2.2 gives a formal summary.

4.2.1. Informal description

The core relations deal with identity, order, hierarchy, different forms of annotation (i. e., free-text annotations, tagging, and formal typing), and generic hyper-links. As the RELATION ontology is represented in $\mathfrak{D}$, the user can (and should) extend it in CDS-based tools. A typical user is expected to use $\mathfrak{D}$ together with the relations defined in $\mathfrak{R}$.

The RELATION names are chosen in a way that a triple (s, r, t) in which s and t are NAMEITEMS can be verbalised as an English sentence. E. g., the triple ([Dirk], [has context], [SAP]) can be pronounced as "Dirk has context SAP". As a notational convention, RELATION names are typeset in square brackets. As defined in $\mathfrak{D}$, the names suffice to uniquely identify them.

The complete relation ontology is depicted in Fig. 4.5. The RELATIONS and the subsumption arrangement is explained informally in the rest of the section. In Sec. 4.2.2 a formal description is given.

As a notational convention, square brackets ("[" and "]") are used to indicate the names of NAMEITEMS *and* RELATIONS.

Related or not?

[is related to]

The Relation [is related to] is the top-relation. Whenever two Items are connected with a Statement, this implies also a [is related to] Triple between those Items. All relations are sub-relations of this top-relation by default. This relation allows stating very vague knowledge, i. e., "these Items are related, but I can't or wouldn't tell why".

Linking

[links to], [is linked from]

If two Items are related in any way, the next step towards a more formal representation is to decide on a direction. Once the direction of a connection between two Items has been defined, this connection is considered a hyperlink in CDS. The semantics of a hyperlink is pretty generic. A user interface can distinguish between incoming and outgoing links. Most web pages show only outgoing links as underlined parts of the text. Many wiki systems on the other hand, show the incoming links (often called "backlinks") in a special section of the page. $\mathfrak{R}$ supplies the Relation [links to] with the inverse [is linked from] for hyperlinks. If there is no direction of the connection between two Items, it might be the case that both Items refer more or less to the same conceptual entity. See Sec. 4.2.1 for these cases.

Order

[comes before], [comes after]

The Relation [comes before] and its inverse [comes after] model ordering relations. It might be order in space, time or by other means, e. g., priority or rank. Sequences such as arrays and lists are used in virtually any information system. This Relation allows representing partial order or even cyclic order. Such freedom is important to let a user experiment with different orders, e. g., for prioritising tasks or sequencing parts of a document.

This Relation is used, e. g., to represent order among parts of a document, as explained in Sec. 4.3.3.

Hierarchy

[has detail], [has context]

The Relation [has detail] and its inverse [has context] represent any kind of hierarchy and nesting. Hierarchies are very common information structures present in documents, organisational charts, file systems, and user interfaces. A hierarchical nesting is also directly present in nature: Every physical area or volume of space can be sub-divided into smaller parts therein. The typical organisation of physical space also uses hierarchically nested sub-spaces, e. g., for mail services (country, city, street, street number, name).

This Relation models hierarchies in a *generic*, *non-strict* way. It is, e. g., perfectly legal to have cycles in the hierarchy or have Items with multiple parents. Note that each CDS Model can represent only one hierarchy with one Relation. A user can create different sub-Relations of [has detail] to represent different hierarchies or use multiple Models for this purpose.

[has subtype], [is subtype of]

The RELATION [has subtype] and its inverse RELATION [is subtype of] model subsumption hierarchies between ITEMS and RELATIONS. Different from other ontology languages, CDS models both type-hierarchies as well as RELATION-hierarchies with the *same* RELATION. As type-hierarchies of all kind are also hierarchies after all, the RELATION [has subtype] is itself a sub-RELATION of [has detail]. The inverse, [is subtype of] is a sub-RELATION of [has context]. This leads formally to a self-referential assertion: [has detail] [has subtype] [has subtype]. This allows to, e. g., browse type hierarchies in the same way as all other kind of hierarchies. The formal semantics of [has subtype] is defined in Sec. 4.2.2.

[has part], [is part of]

Another sub-RELATION of [has detail] is [has part] with its inverse [is part of]. Part-whole relations have an implication about the semantics of copy, move and delete. If an ITEM is deleted, all of its parts should be transitively deleted, too. User interfaces should show all ITEMS that would be deleted and let the user decide if the parts should be deleted as well or not. The default semantics for [has detail] is to keep the details of an ITEM and optionally and on special request delete recursively the details, too. CDS does not define stronger semantics for [has part], as it varies a lot in different contexts.

Annotating and typing

The RELATIONS [annotates] and its inverse [has annotation] models annotations of ITEMS. The ordering RELATIONS [comes before] and [comes after and the hierarchical RELATIONS [has detail] and [has context] both relate typically ITEMS of the same kind. E. g., it makes sense to state order between tasks or document sections, but less so between a task and a document section. The same is true for hierarchies, which often contain ITEMS of the same kind, also probably in a broader sense, e. g., an organisational hierarchy might contain divisions and people. However, both ordering and hierarchical statements usually do not relate things such as a person and a note about this person or an ITEM and the type of that ITEM. The annotation-RELATIONS share that they relate typically ITEMS from different conceptual modelling layers. Annotating ITEMS covers everything from virtual sticky notes up to tagging and formal typing.

[annotates], [has annotation]

If one ITEM annotates another one, the intuitive equivalent is having put a virtual sticky note on the first ITEM, where the second ITEM represents the content of that note. In user interfaces, whenever the ITEM is rendered, the user should be reminded about the annotation, either by showing the annotation directly or by showing an indicator of the presence of annotations. To allow meaningful browsing of annotations, each annotation implies a link from the annotation to the ITEM it annotates, i. e., [annotates] is a sub-RELATION of [is linked from].

[is tag of], [has tag]

With the rise of "Web 2.0" an old technique used for categorising books in libraries becomes popular for digital information items such as web pages or photos: tagging. A tag is essentially a short, unique, easy-to-type keyword that is applied to another information item. In CDS, [is tag of] is a special kind of annotation, hence it's a sub-RELATION of [annotates]. Similarly,

[has tag] is a sub-RELATION of [has annotation]. As tags inherently have the characteristics of being a unique name, users in CDS *should* tag with NAMEITEMS. Tagging in practice is not restricted to text. E. g., the ImageNotion project[9] let's people tag images with images. To allow such cases, too, CDS does not *restrict* tagging to NAMEITEMS.

Tagging can also be used as a generic way to highlight or emphasize a an arbitrary sub-set of ITEMS in a model: A user simply adds a new, common tag to the collection.

A user interface should represent tags in a way familiar to users of other tagging system, e. g., often a tag-cloud is shown, in which all strings used as tag names occur and the font size is proportional to the usage frequency of the tag.

[has instance], [has type]

Annotations are very generic, tagging encourages already the usage of short, descriptive names. The next step in a process of gradual formalisation is to formally *type* ITEMS. The RELATION [has instance] and its inverse [has type] are designed for just that. Whereas tagging has no formal semantic consequences, typing does. If an ITEM a is typed with an ITEM b (i. e., a [has type] b) then a implicitly gets also – transitively – all super-types of b. The formal definition can be found in Sec. 4.2.2. Modelling formal typing as a sub-RELATION of tagging has the consequence that each assignment such as "[Peter] [has type] [Person]" implies a TRIPLE "[Peter] [has tag] [Person]". This in turn allows a user to click on the tag "Person" in, e. g., a tag-cloud and find "Peter" listed in the result set. For a user the navigation from tags to tagged ITEMS and type to instances of the type just happens in the same user interface which reduces the amount of concepts to learn at no cost in expressivity. It also allows one user to successfully browse and explore a MODEL with types without requiring an understanding of even the existence of formal types.

Primitive data-types

Note that many data models distinguish between primitive data-types such as `boolean`, `integer`, and `float` and object-types such as `Person`, `Customer`, and `Product`. In CDS, there is no such distinction. Each CONTENTITEM has some content of which the formal semantics can be stated using [has type]. E. g., typing a number as a "price in Euros" or as a "phone number" is treated in the same way as typing an instance as a "person" or "friend".

Identity

After having several ways in which ITEMS can relate to each other, this section discusses RELATIONS between ITEMS that represent the same or similar conceptual entity.

[is same as]

Several things that have been conceptualised differently, might turn out later to represent in fact the same object. A popular example for this is the "evening star" and the "morning star" which both turned out to be what is called today "Venus". $\mathfrak{R}$ represents this with the [is same as].

[is similar to]

To allow a stepwise path from two entities that turn out to be related in a

[9] `http://www.imagenotion.com/` (accessed 06.01.2010)

way of similarity, rather than difference, $\mathfrak{R}$ offers the RELATION [is similar to]. This symmetric RELATION captures the vague notion that two entities represent a similar, maybe even the same, conceptual entity. Only [is same as] has formal consequences, cf. Sec. 4.2.2.

[is alias for], [has alias]

Another form of similarity is one ITEM being a shorthand name for another one. Nicknames and abbreviations are examples for this. In CDS, this is modelled with the RELATION [is alias for] and its inverse [has alias]. A STATEMENT about an alias is treated as if it has been made about the ITEM with the longer name. If a user had defined
([BaWü], [is alias for], [Baden-Württemberg]) and later creates a STATEMENT
([BaWü], [is located in], [Germany]), then a STATEMENT
([Baden-Württemberg], [is located in], [Germany]) is added to the MODEL. So aliases allow a user to interact more efficiently with a CDS-based tool. If the STATEMENT ([BaWü], [is alias for], [Baden-Württemberg]) is later removed, nothing in the MODEL changes.

This is different from [is same as]-STATEMENTS. They influence the inferencing behaviour and hence the way queries are answered. If the MODEL of a fictional user Dirk contains three STATEMENTS like
([SAP], [has detail], [Claudia]),
([SAP], [is same as], [my employer]), and
([my employer], [is located at], [Karlsruhe]), then a query
([SAP], [*], [*]) would return
([SAP], [employs], [Claudia]) and
([SAP], [is located at], [Karlsruhe]). If later the [is same as]-STATEMENT is removed, the same query returns only
([SAP], [employs], [Claudia]). So the fact that [SAP] and [my employer] represent the same conceptual entity can end to be true.

[replaces], [is replaced by]

The RELATION [replaces] with the inverse [is replaced by] is even stronger than [is alias for] and applies only to textual user interfaces. If a STATEMENT a [replaces] b is in a MODEL, then, when a user types the word b, the editor replaces the string with a. This feature is only relevant if STATEMENTS can be entered using some kind of syntax. Such a syntax is explained in Sec. 4.3.2. If a user later edits the text again, it contains only a.

4.2.2. Formal definition

$\mathfrak{R}$ is defined as a set of RELATIONS and axiomatic STATEMENTS in $\mathfrak{D}$. The notation from Sec. 4.1.2 is thus re-used.

Let $x_1 \dots x_n \in R$ be CDS RELATIONS.
Let the namespace prefix (cf. 2.6) "`cds:`" be mapped to the URI `http://www.semanticdesktop.org/ontologies/2007/09/01/cds#`.
The functions $UID(x)$, $c(x)$, and $r(x)$ are defined on page 134. The RELATIONS in $\mathfrak{R}$ are defined formally as follows:

is related to

Unique Identifier	$UID(x_1) =$	`cds:hasRelated`
Unique Name	$c(x_1) =$	"is related to"
Inverse RELATION	$r(x_1) =$	x_1 (symmetric)

links to

Unique Identifier	$UID(x_2) =$	`cds:hasTarget`
Unique Name	$c(x_2) =$	"links to"
Inverse RELATION	$r(x_2) =$	x_3
Inverse Identifier	$UID(x_3) =$	`cds:hasSource`
Inverse Name	$c(x_3) =$	"is linked from"

annotates

Unique Identifier	$UID(x_4) =$	`cds:hasAnnotationMember`
Unique Name	$c(x_4) =$	"annotates"
Inverse RELATION	$r(x_4) =$	x_5
Inverse Identifier	$UID(x_5) =$	`cds:hasAnnotation`
Inverse Name	$c(x_5) =$	"has annotation"

is tag of

Unique Identifier	$UID(x_6) =$	`cds:hasTagMember`
Unique Name	$c(x_6) =$	"is tag of"
Inverse RELATION	$r(x_6) =$	x_7
Inverse Identifier	$UID(x_7) =$	`cds:hasTag`
Inverse Name	$c(x_7) =$	"is tagged with"

has instance

Unique Identifier	$UID(x_8) =$	`cds:hasInstance`
Unique Name	$c(x_8) =$	"has instance"
Inverse RELATION	$r(x_8) =$	x_9
Inverse Identifier	$UID(x_9) =$	`rdf:type`
Inverse Name	$c(x_9) =$	"has type"

The semantics of this RELATION are described in "[has subtype]".

comes before

Unique Identifier	$UID(x_{10}) =$	`cds:hasAfter`
Unique Name	$c(x_{10}) =$	"comes before"
Inverse RELATION	$r(x_{10}) =$	x_{11}
Inverse Identifier	$UID(x_{11}) =$	`cds:hasBefore`
Inverse Name	$c(x_{11}) =$	"comes after"

This RELATION is transitive. It models *partial ordering*.

has detail

Unique Identifier	$UID(x_{12}) =$	`cds:hasDetail`
Unique Name	$c(x_{12}) =$	"has detail"
Inverse RELATION	$r(x_{12}) =$	x_{13}
Inverse Identifier	$UID(x_{13}) =$	`cds:hasContext`
Inverse Name	$c(x_{13}) =$	"has context"

has subtype

Unique Identifier	$UID(x_{14}) =$	`cds:hasSubType`
Unique Name	$c(x_{14}) =$	"has subtype"
Inverse RELATION	$r(x_{14}) =$	x_{15}
Inverse Identifier	$UID(x_{15}) =$	`cds:hasSuperType`
Inverse Name	$c(x_{15}) =$	"is subtype of"

This RELATION, which models both subsumption hierarchies of types

as well as RELATIONS, is transitive. Formally,
$\forall u, v, w : (u, x_{14}, v), (v, x_{14}, w) \in M_{true} \Rightarrow (u, x_{14}, w) \in M_{true}$.

An ITEM inherits super-types of a type. Formally,
$\forall a, b, c \in I : (a, x_9, b) \wedge (b, x_{15}, c) \in M_{true} \Rightarrow (a, x_9, c) \in M_{true}$.

Furthermore, to make sub-RELATIONS consistent with inverse RELATIONS, the inverse RELATION of a RELATION must also be a sub-RELATION of the RELATION. Formally,
$\forall u, v \in R : (u, x_{14}, v) \in M_{true} \Rightarrow (-u, x_{14}, -v) \in M_{true}$. This is possible, because each RELATION in $\mathfrak{D}$ is required to have a defined inverse RELATION.

Note that the type-system can also be used to denote data-types such as integer, float, string, or boolean.

has part

Unique Identifier	$UID(x_{16}) =$	`cds:hasPart`
Unique Name	$c(x_{16}) =$	"has part"
Inverse RELATION	$r(x_{16}) =$	x_{17}
Inverse Identifier	$UID(x_{17}) =$	`cds:isPartOf`
Inverse Name	$c(x_{17}) =$	"is part of"

is similar to

Unique Identifier	$UID(x_{18}) =$	`cds:hasSimilar`
Unique Name	$c(x_{18}) =$	"is similar to"
Inverse RELATION	$r(x_{18}) =$	x_{18}

is same as

Unique Identifier	$UID(x_{19}) =$	`cds:sameAs`
Unique Name	$c(x_{19}) =$	"is same as"
Inverse RELATION	$r(x_{19}) =$	x_{19}

This transitive relation effectively defines one ITEM to be an alternative symbol for another one. Definition: A MODEL that contains no TRIPLES of the form (a, x_{19}, b) for any $a, b \in$ MODEL is called *lean*. If it does contain such triples it is called *non-lean*.

A non-lean model can transformed into a lean model in the following way:

1. Compute the transitive closure over (a, x_{19}, b) for all as and bs.
2. For each triple (a, x_{19}, b) in M_{true}:
   2a. Replace each occurrence of a in the MODEL with b.
   2b. Remove the triple (a, x_{19}, b) from M_{true}.
   This can mean to remove it from $M_{explicit}$ or $M_{implicit}$.

The transformation can be thought of as a syntactic pre-processing step. The same replacement operations have to be carried out for answering queries, so that the same normalisation takes place.

$$
\begin{aligned}
s_{triple}(a_1) &= (\ x_1, \ x_{14}, \ x_2\ )\\
s_{triple}(a_2) &= (\ x_2, \ x_{14}, \ x_4\ )\\
s_{triple}(a_3) &= (\ x_4, \ x_{14}, \ x_6\ )\\
s_{triple}(a_4) &= (\ x_6, \ x_{14}, \ x_8\ )\\
s_{triple}(a_5) &= (\ x_2, \ x_{14}, \ x_{10}\ )\\
s_{triple}(a_6) &= (\ x_2, \ x_{14}, \ x_{12}\ )\\
s_{triple}(a_7) &= (\ x_{12}, \ x_{14}, \ x_{14}\ )\\
s_{triple}(a_8) &= (\ x_{12}, \ x_{14}, \ x_{16}\ )\\
s_{triple}(a_9) &= (\ x_1, \ x_{14}, \ x_{18}\ )\\
s_{triple}(a_{10}) &= (\ x_{18}, \ x_{14}, \ x_{19}\ )\\
s_{triple}(a_{11}) &= (\ x_{18}, \ x_{14}, \ x_{20}\ )\\
s_{triple}(a_{12}) &= (\ x_{18}, \ x_{14}, \ x_{22}\ )
\end{aligned}
$$

Table 4.3.: Axiomatic CDS Statements

is alias for

Unique Identifier	$UID(x_{20}) =$	`cds:isAliasOf`
Unique Name	$c(x_{20}) =$	"is alias for"
Inverse Relation	$r(x_{20}) =$	x_{21}
Inverse Identifier	$UID(x_{21}) =$	`cds:hasAlias`
Inverse Name	$c(x_{21}) =$	"has alias"

The semantic of this Relation applies only when new Triples are created either explicitly by a user or indirectly by processing text syntax which implies the creation of Triples. In both cases, each occurrence of an Item a for which an alias has been defined by (a, x_{20}, b) is replaced with b. To guarantee consistent Models, b must be a Relation if a is a Relation. If this is not the case, the alias directive is not in effect for Statements.

Formally, the following rules apply at Statement insertion time if the Model contains a Triple (a, x_{20}, b):
$(a, r, t) \mapsto (b, r, t)$
$(s, a, t) \mapsto (s, b, t)$ (Note: here $b \in R$ is required.)
$(s, r, a) \mapsto (s, r, b)$

replaces

Unique Identifier	$UID(x_{22}) =$	`cds:replaces`
Unique Name	$c(x_{22}) =$	"replaces"
Inverse Relation	$r(x_{22}) =$	x_{23}
Inverse Identifier	$UID(x_{23}) =$	`cds:replacedBy`
Inverse Name	$c(x_{23}) =$	"is replaced by"

The semantics of this Relation applies only when a user edits text in a CDS-based tool. If the Model contains a Statement of the form (a, x_{22}, b) and b is a NameItem ($b \in N \cup R$), then after the last character of b's name has been typed the editor should replace the string b with the string a. a needs not to be a NameItem. The requirement $b \in N \cup R$ is necessary to guarantee unambiguous replacement.

The subsumption hierarchy can be expressed as a set of axiomatic STATEMENTS, which are depicted in Table 4.3.

For RELATIONS that are, e. g., sub-RELATIONS of a non-symmetric RELATION, there are implications for navigation in MODEL. E. g., [links to] has the sub-RELATION [has detail], (x_2, x_{14}, x_{12}). Thus if a pure hypertext-based tools renders a MODEL, a user can navigate from an ITEM to its detail. This implies a kind of natural linking direction from generic to specific.

Formal semantics

This definition builds upon the formal semantics of $\mathfrak{D}$, described in Sec. 4.1.2.

- Let G be the set of all *formal types* in a MODEL. That is, G contains all ITEMS *used* as a type, either in a TRIPLE with [has type] or [has subtype]. G is a subset of the vocabulary V, $G \subseteq V$. Formally, $F = \{v|(u, x_9, v) \in M_{true}\} \cup \{x, y|(x, x_{14}, y) \in M_{true}\}$.
- Let IG_{EXT} be the *type extension* in the interpretation H, with $IG_{EXT} : H(G) \rightarrow \text{powerset}(U)$.

  A triple (x,[has type], y) in interpreted as $x \in V$ having the formal type $y \in G$.

  Formally, $x \in IG_{EXT}(y)$ if and only if $(x, y) \in IR_{EXT}(H([\text{has type}]))$.
- $IR_{EXT}(H([\text{has subtype}]))$ is transitive and reflexive on $R \cup G$.
- Next the semantics of sub-RELATIONS can be formalised as: $\forall x, y \in R, (x, y) \in IR_{EXT}(H([\text{has subtype}])) : IR_{EXT}(x) \supseteq IR_{EXT}(y)$.
- In a similar way the more generic sub-type relationship can be defined: $\forall (x, y) \in IR_{EXT}(H([\text{has subtype}])) : x \in IG, y \in IG, IG_{EXT}(x) \subseteq IG_{EXT}(y)$.
  Note that a sub-RELATION-TRIPLE has both implications for its RELATION extension (IR_{EXT}) as well as for its type-extension (IG_{EXT}).

4.3. Syntax and structured text

This section describes handling of textual content in CDS. First STIF, a model for structured text, is presented. STIF can be used independent of CDS to represent structured text, as it occurs, e. g., in word processing, web pages, content management systems, the text field of mind-map nodes, or within blog comments.

STIF is used as a content-type in the content of ITEMS in several CDS tools. STIF can also be used for *import* from and *export* to other formats.

STIF can be edited in a textual syntax, which is also presented in this section. The syntax is thus used to *create* and *augment* knowledge cues.

Furthermore, a single content in STIF can be transformed to multiple interlinked Items, each having a part of the original content as their content. Taken together, the transformations from syntax to structure and from structure to CDS allow a user to work always in the most convenient formalism.

Former research on STIF has been published by Völkel and Oren (2006). A previous version of STIF has been published as Wiki Interchange Format (WIF, Völkel and Oren, 2006). STIF is also used outside CDS, e. g., for an open source wiki migration and access framework called WikiPipes, available at `http://code.google.com/p/wikipipes` *(accessed 06.01.2010).*

4.3.1. Structured text interchange format (STIF)

STIF is a representation language for structured text. It is designed as a simple markup language allowing to represent only structural features (i. e., headlines, list, tables, images and links), but not visual features (such as font style, font size and colour).

HTML as basis

Design goals and decisions One of the most prominent document formats today is HTML (cf. 2.4). Therefore HTML is taken as a basis for representing structured text. HTML provides many elements; for STIF only those representing document structures (cf. 2.2) are relevant. Therefore, **STIF is a strict subset of HTML**. Being a subset of HTML allows all STIF documents to be rendered by all HTML browser engines. This simplifies implementation of CDS tools.

The design of STIF is a trade-off between usability and expressivity. It must be small, so that it is easy to learn and simple to write transformation procedures to and from other data formats. Transformations of structured text to other formalisms can help to create semantic statements cheaper, as explained in Sec. 4.3.3. It must be expressive, so that a user can express all kinds of textual structures and so that all kinds of existing text content formats can be represented in STIF. STIF is a small subset of HTML, which is easy to learn and use. On the other hand, STIF allows arbitrary other elements to be present - those are legal to be ignored by STIF and CDS tools.

XML is the de-factor standard for representing semi-structured data. With XSLT a powerful language standard for writing transformations from XML to XML and from XML to text is available. The current version of HTML is already specified as a subset of XML, namely XHTML. **STIF is a subset of XHTML**, and hence XML, too.

To be precisely, STIF is a strict subset of XHTML 1.1 (Altheim and McCarron, 2001). It uses the *structure module*, *hypertext module*, *list module*, *table module*, *image module* and parts of the *text module* and *presentation module*. Appendix A.3.2 shows the formal document type definition (DTD) of STIF which states clearly which elements are used and how they can be nested.

	Used in WikiCreole	Added in STIF
Inline formatting	`<strong>`, `<em>`, ` `	`<code>`
Headings	`<h1>`, `<h2>`, `<h3>`, `<h4>`, `<h5>`, `<h6>`	–
Links	`<a>` with attribute `href`	–
Block-level	`<p>`, `<hr>`	`<pre>`
Lists	`<ol>`, `<ul>`, `<li>`	`<dl>`, `<dd>`, `<dt>`
Images	`<img>` with attributes `src` and `alt`	–
Tables	`<table>`, `<tr>`, `<th>`, `<td>`	–

Table 4.4.: Comparing Wiki Creole and STIF

Restricting to WikiCreole

In 2007, a unified wiki syntax, called *Wiki Creole*, was created (Sauer, Smith, and Benz, 2007). The syntax was created after investigating the syntax of the twenty most popular wiki engines followed by an open discussion among many wiki users and wiki engine authors for a period of almost one year. Therefore, Wiki Creole represents the most important concepts for authoring structured text. Wiki Creole supports 20 HTML elements, which all contribute to the document structure. The set of Wiki Creole elements is depicted in Tab. 4.4.

STIF uses Wiki Creole as a base but adds some few more elements, which are depicted in the right column in Tab. 4.4. Wiki Creole supports two kinds of lists, numbered and bulleted, but leaves out definition lists, which are added to STIF to allow representing all three common list structures present in HTML. Furthermore, definition lists are the most structured kind of list. Wiki Creole has syntax for escaping wiki parsing, but does not prescribe how such sections should be rendered in HTML. STIF adds the HTML elements for inline verbatim text (`code`) and block-level verbatim text (`pre`). These elements allow STIF to represent unformatted text, which is important to allow representing text on all levels of formality.

Linking in STIF One design goal of STIF is interchange between different systems. Most hypertext systems allow linking *within* itself as well as to *other* systems.

In the World Wide Web, there are links to other parts of the same web page (with `href="#..."`) and links to other web pages or parts thereof (e. g., with `href="http://..."`). In wikis, there are links to other wiki pages or links to other WWW pages.

Usually all entities in one hypertext system share a common layout or navigation approach. Therefore the user is interested before clicking a link, whether it will be an inter-system-link or an intra-system-link. STIF en-

Inline formatting	, , , <code>
Headings	<h1>, <h2>, <h3>, <h4>, <h5>, <h6>
Links	<a> with attribute href
Block-level	<p>, <hr>, <pre>
Lists	, , , <dl>, <dd>, <dt>
Images	<img> with attributes src and alt
Tables	<table>, <tr>, <th>, <td>

Table 4.5.: Summary of STIF elements. Inside the <a>-element, the attribute class must (compatible with the CSS-specification, see 2.4) contain stif-internal for links within the same system or knowledge base and stif-external for links to another system or knowledge base. Only one of the two classes may be used for one element.

codes this distinction by using the class-attribute of the <a>-element[10]. For internal links, i. e., those linking to other entities in the same system or knowledge base, the class stif-internal is used. For external links, the class stif-external is used.

Specification

The complete list of elements and attributes used in STIF is depicted in Tab. 4.5

STIF is intended to be used as a markup language within other knowledge representations, especially for the content of CDS Items. There is no required document header which turns each plain text string that does not use XML special characters into a valid STIF string. E. g., "Hello World" is a valid STIF document. This allows using STIF for all phases in the gradual migration from plain text to fully-fledged structured documents.

STIF string A *valid STIF string* is any sequence of characters s with the following properties:

- Either "s" itself or embedded in an XML root element, "<root>" +s+ "</root>", is well-formed XML 1.1, as defined by Paoli, Cowan, Bray, Yergeau, Maler, and Sperberg-McQueen (2006, Sec. 2.1).
- The STIF-elements that are used in s – if any – follow the *nesting rules* of HTML 4.01 Strict (Raggett, Hors, and Jacobs, 1999). The nesting rules, e. g., dictate that a <table> element is not allowed within a list element <li>. Note: using DTDs it is not possible to formally describe, e. g., "this element may contain recursively any element except this and that element".

[10] HTML defines that the class-attribute of any HTML element may take an infinite set of values, separated by the space character. Therefore STIF does not limit the expressivity by adding some elements to this set.

Note that this definition does not require *valid* HTML or XHTML. A study in 2002/2003 (Noga and Völkel, 2003) found that less than one percent of all web pages are valid (X)HTML as defined by W3C.

STIF document A *STIF document* is an XML 1.1 document that is *valid* (as defined by Paoli et al. (2006, Sec. 2.8 pp)) *s* with respect to the STIF *Document Type Definition* (DTD). The complete STIF DTD is shown in appendix A.3.2. The STIF DTD is essentially a stripped-down HTML 4.01 DTD with lowercase element names, to remain XHTML-compatible. HTML (but not XHTML) allows uppercase and lowercase element names. A typical body of a strict STIF document is depicted in Fig. 4.6. Every valid STIF document contains a valid STIF string inside the `<stif>`-element.

```
<?xml version="1.1"?>[a]
<!DOCTYPE stif PUBLIC "-//XAMDE//DTD STIF 1.0//EN">
<stif> ...a STIF string ... </stif>
```

[a] XML 1.1 is XML 1.0 with improved Unicode support

Figure 4.6.: Document template for validating STIF strings

STIF-compliant processor A *STIF-compliant processor* is defined analog to HTML-compliant user-agents (browsers) as defined by (Pemberton, 2000, Sec. 3.2): In valid STIF strings or STIF documents, a STIF-compliant processor must process all elements defined for STIF. Behaviour for other elements can be chosen freely as long as the general guidelines for user-agents stated by (Pemberton, 2000, Sec. 3.2) are met. The most important of those rules are:

- If a user agent encounters an element it does not recognize, it must process the element's content.
- If a user agent encounters an attribute it does not recognize, it must ignore the entire attribute specification (i. e., the attribute and its value)
- If a user agent encounters an attribute value it does not recognize, it must use the default attribute value.

These rules ensure that all STIF strings can be rendered as part of HTML pages.

4.3.2. From syntax to structured text

The syntax for STIF is based on the *CommonSyntax* of the open-source project *WikiModel*[11]. WikiModel provides a powerful framework for parsing wiki syntax. Note that most wiki engines do not provide a re-usable parser, but instead intertwine text parsing and specific wiki business logic deeply.

[11] `http://code.google.com/p/wikimodel/` (accessed 06.01.2010)

The appendix A.3.1 lists the STIF wiki syntax for creating STIF content in a set of comparison tables. The STIF wiki syntax is a refined version of CommonSyntax. I. e., CommonSyntax does not define link parsing within "[" and "]" markers. Note that STIF can also be used with another wiki syntax or without any wiki syntax, e. g., for migrating structured text from one tool to another one.

4.3.3. From structured text to CDS

This section describes transformations between structured text (which can be created with a textual syntax as explained in the previous section) and formal statements. The overall idea is depicted in Fig. 4.7.

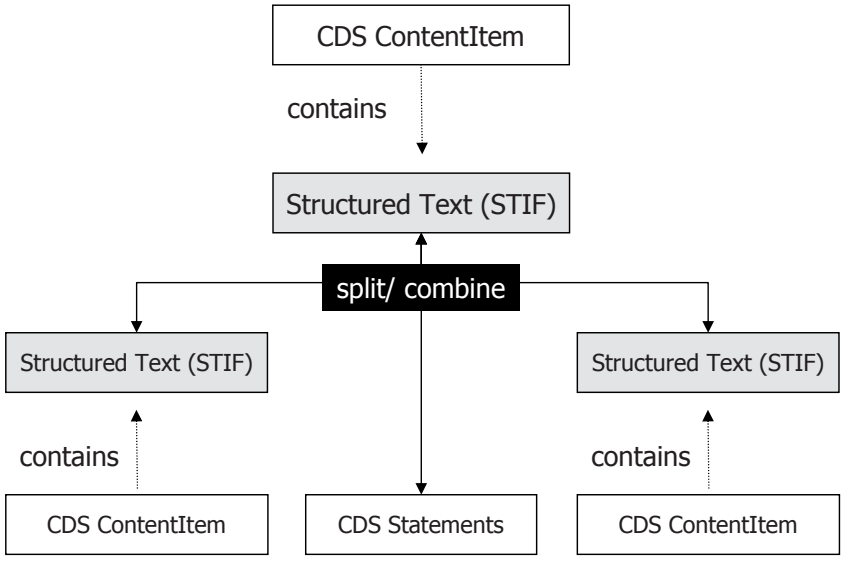


Figure 4.7.: Transformations between structures in text and structures in relations

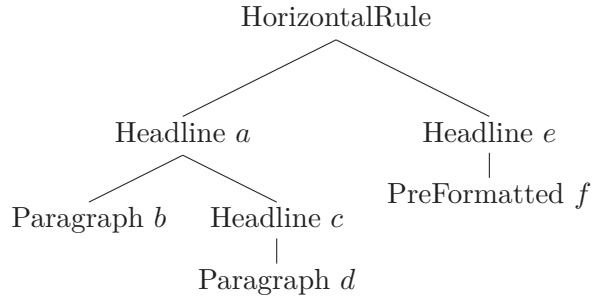
Using STIF in CDS

There are two prerequisites to use STIF as content in CDS Items:

- Sec. 4.1.2 introduced a set Q of characters which must now also include the XML special characters. Formally, let Q' be the extended alphabet, with $Q' = Q \cup \{$ "<", ">", "=", "'", "" $\}$. A STIF string is a string from the set A', the infinite set of all possible words over the extended alphabet Q'. The special characters are required to encode the markup of element names and attributes, as specified by Paoli et al. (2006).
- The content-type for STIF content is $c_{STIF} \in C_{TYPE}$. This content-type is added to the set of available content types.

Note that both changes happen within the formal model described in Sec. 4.1.

Given an ITEM u with
STIF content:
<h1>a</h1>
b
<h3>c</h3>
d
<hr>
<h3>e</h3>
<pre>f</pre>



Notation:
In each node n the type ($type(n)$) is followed by the content ($content(n)$). The CDS model contains at least one ITEM u with $c_{type}(u) = STIF$ and $c(u) =$ the STIF content listed.

Figure 4.8.: From STIF to a Logical Document Tree

Representing STIF as CDS

STIF → Logical Document Tree STIF content implies a logical document structure (cf. Sec. 2.2).

The HTML block-level elements such as headings (`<h1>`, `<h2>`, `<h3>`, `<h4>`, `<h5>`, `<h6>`), paragraph (`<p>`), horizontal rule (`<hr>`), pre-formatted block (`<pre>`), and tables (`<table>`, `<tr>`, `<th>`, `<td>`) impose a logical structure. Note that all structural elements except horizontal rules have content of their own. E. g., a headline `<h2>Introduction</h2>` has the content "Introduction" and maybe a paragraph underneath. A `hr`-command has potentially also a paragraph *underneath*, but never content *within* the logical node representing the `hr`-command itself.

Paragraph, pre-formatted block and table can only appear as leaves of a logical document tree. Within those, inline formatting (`<strong>`, `<em>`, `<br>`, and `<code>`), links (`<a>`), images (`<img>`) and lists (`<ol>`, `<ul>`, `<li>`, `<dl>`, `<dd>`, `<dt>`) may occur.

Fig. 4.8 shows STIF content (left) and a corresponding logical document tree (right). The different levels of headlines (`<h1>`-`<h6>`) have been abstracted away. Only the relative structure between the headlines is captured in the structure of the tree. Each node of the tree contains an ordered list of children in order to represent the order of elements in the source document.

This analysis leads to this set of logical document tree types: "STIF" for content that has not been split up; "HorizontalRule", "Headline", "Paragraph", "Table" and "PreFormatted" otherwise.

Each Node n is mapped to an ITEM i such that:
// **representing content**
$content(i) = content(n)$
// **representing types**
create a STATEMENT $(i, [\text{has type}], \text{matching type from Tab. 4.6})$
for each child n_c of n create a corresponding ITEM i_c recursively and
// **representing hierarchy**
create a STATEMENT $(i, [\text{has part}], i_c)$
// **representing order**
for each pair of child nodes n_i, n_j in $n.L$:
if $(i + 1 = j)$
create a STATEMENT $(n_i, [\text{comes before}], n_j)$

Figure 4.9.: Algorithm for representing a Logical Document Tree in CDS

STIF element	Logical Document Tree	Unique Identifier	Unique Name
`<hr>`	*HorizontalRule*	`cds:stif-HorizontalRule`	Horizontal Rule
`<h1>`	*Headline*	`cds:stif-Headline`	Headline
`<h2>`	*Headline*	`cds:stif-Headline`	Headline
`<h3>`	*Headline*	`cds:stif-Headline`	Headline
`<h4>`	*Headline*	`cds:stif-Headline`	Headline
`<h5>`	*Headline*	`cds:stif-Headline`	Headline
`<h6>`	*Headline*	`cds:stif-Headline`	Headline
`<p>`	*Paragraph*	`cds:stif-Paragraph`	Paragraph
`<pre>`	*PreFormatted*	`cds:stif-PreFormatted`	Pre-formatted Block
`<table>`	*Table*	`cds:stif-Table`	Table

Table 4.6.: Mapping from STIF elements to CDS types represented as NAMEITEMS

Logical Document Tree → CDS A logical document tree can be represented in CDS by representing *order*, *hierarchy*, and *types* of tree nodes. Order can be represented with [comes before], hierarchy with [has part], and typing with [has type]. For each kind of logical document tree node, an ITEM representing that type must be created. Given the NAMEITEMS defined in Tab. 4.6, an algorithm for representing a logical document tree can be defined (cf. Fig. 4.9.

Normalisation A STIF string can be *normalised* by transforming it into a logical tree and back to a STIF string. The logical tree contains by definition all information of the structure and content of a STIF string, but looses the exact definition of `<h1>` ... `<h6>` and retains only the relative nesting of these elements. Thus on representing the logical tree back to a STIF string, effectively only the headline numbering has changed. Example: If the initial STIF string was `<h1>a</h1><h3>b</h3>` the normalized STIF string is `<h1>a</h1><h2>b</h2>`.

Splitting and merging CDS items with STIF content

So far a single STIF content has been represented very fine-granular as CDS. In practice, a user will rarely want to transform a single document represented as STIF into hundreds of ITEMS. However, being able to split big ITEMS apart and merge several small ITEMS into a single one is an important feature to let the user control the size of ITEMS.

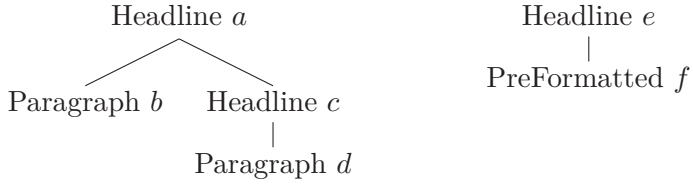


Figure 4.10.: Splitting a Logical Document Tree

The last section explained how a logical document tree can be represented both as STIF and as CDS. By defining how to split and merge logical document trees, a split and merge for ITEMS with STIF content is implied. Given the tree from Fig. 4.8, it can easily be split into smaller trees, depicted in Fig. 4.10. The resulting CDS ITEMS and STATEMENTS are depicted in Fig. 4.11. In general, the root node becomes an ITEM with few or no content and all children of that node become ITEMS containing the corresponding sub-part of the logical document tree, encoded as STIF.

Merging

This process can be reverted in order to *merge* (combine) several ITEMS. However, if a user deletes, e. g., the [has part]-STATEMENT, then the reverse process is no longer possible, as information is lost.

Summary

Both processes ease the creation of structured content, as one can start to write structured documents, e. g., while sitting in a meeting and later transforming the text into a more fine granular CDS model.

Resulting CDS ITEMS (note the normalisation of headline-numbering):

ITEM	c_{type}	c
u	STIF	–
v	STIF	`<h1>a</h1> <p>b</p> <h2>c</h2> <p>d</p>`
w	STIF	`<h1>e</h1> <pre>f</pre>`

Resulting CDS STATEMENTS:

source	relation	target
u	[has type]	`cds:stif-HorizontalRule`
u	[has part]	v
u	[has part]	w
v	[comes before]	w

Figure 4.11.: CDS items and statements resulting from a split operation

Interpreting structures as CDS

Besides just representing the logical document structure, further possibilities to advance from text structures to semantic statements are described in this section.

Automatic linking Given a textual content like "... Dirk works at SAP ..." and a NameItem with content "Dirk", links can be inserted automatically, leading to "...Dirk works at SAP", where *d* is the URI of the *Dirk*-Item.

Interpreting links in STIF First of all, each wiki-link inside square brackets, e. g., [Dirk], is interpreted as a link to the NameItem with matching content, e. g., "Dirk". If no such NameItem exists, it is automatically created with a random URI.

Example:
Let *a* be a NameItem with content *AAA* and URI *aaa*.
Let *b* be a NameItem with content *BBB* and URI *bbb*.
Let *c* be a ContentItem with textual content like
"...*BBB*...".
Given further the Statements $(c, [\text{has alias}], a)$.
Then the a Statement can be extracted in order to represent the link from *a* to *b*: $(a, [\text{links to}], c)$.

Semantic links via wiki syntax Similar to the syntax used in *Semantic MediaWiki* (cf. Sec. 2.9), the syntax presented in Sec. 4.3.2 is extended with *semantic links*. A normal link to a NameItem with content *AAA* can be created with
"... [AAA] ...".
A semantic link with type *ttt* can be created via
"... [ttt::AAA] ...".
In STIF, semantic links are represented in a way compatible with a way of embedding RDF in HTML (cf. eRDF, Sec. A.1). CDS uniformly puts relations in the cds.-namespace. This works, because Relations must be named uniquely in CDS. The STIF fragment for the example looks like this:

```
... <a href="#urn:example:dirk"
       class="stif-internal cds-explicit"
       rel="cds.knows"
      >Dirk</a> ...
```

```
wikiTurtleDoc       ::= statement*
statement           ::= triples '.' | ws+
triples             ::= subject predicateObjectList
subject             ::= resource | blank
predicateObjectList ::= verb objectList ( ';' verb objectList )* ( ';')?
verb                ::= resource
objectList          ::= object ( ',' object)*
object              ::= resource | literal
literal             ::= '"' any characters except quotation mark '"'
ws                  ::= whitespace characters
resource            ::= '[' any characters except closing brace ']'
```

Figure 4.12.: Semantic Wiki Turtle Syntax

Interpreting list structures Often lists are used to express semantic relations as well. The CDS syntax framework interprets lists like this:

```
* [AAA]
** [knows]
*** [BBB]
*** [CCC]
```

as the STATEMENTS $(aaa, [\text{knows}], bbb)$ and $(aaa, [\text{knows}], ccc)$ with *aaa* being the URI of a NAMEITEM with content "AAA" etc.

Advanced semantic syntax A popular yet compact syntax for RDF is *Turtle* (Beckett and Berners-Lee, 2008). Analogous to turtle a corresponding semantic wiki syntax has been designed for CDS.

As an example, the syntax `[AAA] [knows] [BBB], [CCC]; [likes] [DDD].` is interpreted as the STATEMENTS $(aaa, knows, bbb)$, $(aaa, knows, ccc)$, and $(aaa, likes, ddd)$. The EBNF grammar of Semantic Wiki Turtle Syntax is shown in Fig. 4.12.

Provenance For each semantic STATEMENT s created from wiki syntax in a CONTENTITEM a, CDS creates an additional STATEMENT $(s, [\text{has provenance}], a)$. The inverse of [has provenance] is [defines]. The RELATION [has provenance] is a sub-RELATION of [is related to].

4.3.4. Summary

Creating knowledge cues with inner structure at low costs is an appealing feature. Using a formal representation of structured text, such structured documents can transformed. In particular, structured text can be split up into several smaller, interlinked parts, retaining the structural links but representing them differently. This transforms structures *within* a knowledge cue into structures *between* knowledge cues.

4.4. Using CDS

This section presents how the different parts ($\mathfrak{D}$, $\mathfrak{R}$, and STIF) are used together for PKM. First authoring is presented, then a mapping to semantic technologies, especially Resource Description Framework (RDF), is given. Finally retrieval in the knowledge model and some modelling examples are shown. A more concrete example for using CDS embedded in the tool HKW is shown in Sec. 5.2.1.

4.4.1. Authoring and stepwise formalisation

There are several ways for stepwise formalisation possible in CDS:

Taking a note in CDS

- The simplest thing a user can do is to create a plain ITEM. This is equivalent to take a piece of paper and write the date on it.
- Next the user can write text in this ITEM. Or set the content to contain an image. In any case, some piece of content becomes an addressable entity in the model.
- The user can turn the ITEM into a NAMEITEM. The system has to check if another NAMEITEM with the same name (content) exists already, and if so, the user has to decide what should happen: Delete the new ITEM? Rename it? Merge the two? Even small formalisation steps can come at some cost.
- A user can link any two existing ITEMS via one of the built-in relations. The simplest possible link is an undirected [is related to]-link between two ITEMS. It is analogous to drawing a line between two pieces of paper. E. g., sometimes it is easy to say that two ITEMS are related but it is hard to say how.
- The user can *refine* any existing relation. For [is related to]-relations, the user might, as a next step, choose a directed hyperlink ([links to]) or express a kind of similarity ([is similar to]).
- The user can also create new relations, as they are needed. Browsing a knowledge model is easier when the most suitable parent relation is chosen from the existing relations (cf. Fig. 4.5).
- The user can structure the content of an ITEM using wiki syntax, e g. format text in sections, lists and tables.
- The user can add formal statements to the STIF-content (cf. Sec. 4.3.3).

Creating relations

How to use? To clarify the semantics of new RELATIONS, a user should create a sub-RELATION of the existing RELATIONS given in $\mathfrak{R}$. E. g., to indicate that [is located in] has semantics of hierarchical nesting it should be made a sub-RELATION of [has context].

Extending the built-in relation ontology The user is expected to extend the provided relation ontology $\mathfrak{R}$ all the time. In most CDS-based tools (cf. Ch. 5), new RELATIONS ce be introduced simply by using them. These new RELATIONS are by default a sub-type of [is related to].

Extending the built-in relation ontology is easy, one just needs to create new RELATIONS as sub-RELATIONS of existing RELATIONS (via [is subtype of]). This allows tools falling back on more generic semantics, if the new RELATION cannot be understood. E. g., [met at a conference] could imply [knows], which in turn might be layered underneath [links to]. Creating sub-relations of existing RELATIONS is simply performed by adding a STATEMENT $(x, [\text{has super-relation}], y)$.

One example would be the introduction of a RELATION [comes directly before] as a sub-RELATION of [comes before]. For the user, the semantics of [comes directly before] could characterise total ordering, e. g., of chapters in a book. It makes a difference to say that a chapter on chemical components comes somewhere before a chapter on polymers or *directly* before. Ultimately, an author must decide on the total order of chapters, before a book can be finished. Next the author might realise that she needs to represent two orderings: one for the order in the book and one for herself, the order of writing. So she creates another RELATION [write before] to record her thoughts about the dependencies of the chapters in writing order.

4.4.2. Mapping to semantic technologies

This section explains a representation of CDS data-model ($\mathfrak{D}$) in the RDF data model. In a second step, the relation ontology $\mathfrak{R}$ is mapped to existing ontologies.

Representing the CDS data-model in RDF Mapping CDS to RDF allows knowledge models to be exported into other semantic technology-based tools. The following namespace bindings (Turtle syntax, described by Beckett and Berners-Lee (2008)) are used:

@prefix rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#> .
@prefix rdfs: <http://www.w3.org/2000/01/rdf-schema#> .
@prefix xsd: <http://www.w3.org/2001/XMLSchema#> .
@prefix cds: <http://www.semanticdesktop.org/ontologies/2007/09/01/cds#> .

The examples given in this section use random URIs, encoding the current system time and a locally unique disambiguation number, allowing several random URIs to be created at the same time point. An example for such a random URI is `urn:xam.de:t20090816-11.18.25.824-0`.

ContentItem Each CONTENTITEM with URI i is represented as i `rdf:type cds:ContentItem`. If the CONTENTITEM has content attached to it, it is represented as i `cds:hasContent` *content*. Additionally, each CONTENTITEM has a change date (`cds:hasChangeDate`) and an author (`cds:hasAuthor`).

An example of an CONTENTITEM with content:

```
<urn:xam.de:t20090816-11.18.25.824-0> rdf:type cds:ContentItem;
cds:hasContent "Lorem ipsum dolor sit amet,
  consectetuer adipiscing elit. Proin id enim a
  velit cursus tempor. Aenean non erat. Mauris
  imperdiet, sem in iaculis interdum, velit libero
  aliquam nisi, scelerisque tincidunt leo dui eget pede.";
cds:hasChangeDate "2010-11-26T14:48:12Z"^^xsd:dateTime;
cds:hasAuthor     <urn:xam.de:t20090714-12.37.55.890-0> .
```

NameItem NAMEITEMS are represented almost like ITEMS. They have the type `cds:NameItem` assigned. An example:

```
<urn:xam.de:t20090816-12.27.33.109-0> rdf:type cds:NameItem;
cds:hasContent "Dirk Hageman";
cds:hasChangeDate "2010-11-26T14:37:09Z"^^xsd:dateTime;
cds:hasAuthor     <urn:xam.de:t20090714-12.37.55.890-0> .
```

Relation A RELATION has an inverse relation specified via `cds:hasInverse`. And it has the type `cds:Relation` assigned. An example:

```
<urn:xam.de:t20090816-11.19.25.798-0> rdf:type cds:Relation;
cds:hasContent "writes PhD at";
cds:hasChangeDate "2010-11-26T14:38:09Z"^^xsd:dateTime;
cds:hasAuthor  <urn:xam.de:t20090714-12.37.55.890-0>;
cds:hasInverse <urn:rnd:252f391c:1167c5d5744:-7fcf> .
```

Statement A STATEMENT has the type `cds:Statement` assigned. A STATEMENT may not be a NAMEITEM or RELATION at the same time. A STATEMENT, just like a normal ITEM, may have content. Below is an example of a statement without content attached. Note that the statement is in RDF terms "reified", to allow addressing it. The plain, un-reified statement is also recorded in RDF to allow answering of SPARQL queries:

```
<urn:xam.de:t20090816-11.46.15.777-0> rdf:type cds:Statement;
cds:hasChangeDate "2010-11-28T15:18:19Z"^^xsd:dateTime;
cds:hasAuthor    <urn:xam.de:t20090714-12.37.55.890-0>;
cds:stmtSource   <urn:xam.de:t20090816-11.28.31.396-0>;
cds:stmtRelation <urn:xam.de:t20090816-11.34.32.754-0>;
cds:stmtTarget   <urn:xam.de:t20090816-11.46.15.772-0> .

<urn:xam.de:t20090816-11.28.31.396-0>
  <urn:xam.de:t20090816-11.34.32.754-0>
    <urn:xam.de:t20090816-11.46.15.772-0> .
```

Triple A TRIPLE is not a kind of ITEM. A triple is almost the same as an RDF triple. The only difference is that a CDS triple contains only URIs and no *literals* or *blank nodes*.

```
<urn:xam.de:t20090816-11.28.31.396-0>
  <urn:xam.de:t20090816-11.34.32.754-0>
    <urn:xam.de:t20090816-11.46.15.772-0> .
```

Example

As an example, *Claudia* wants to express the facts that *Dirk* works at *SAP* and that *SAP* has the context *Karlsruhe*. In CDS, she could use NAMEITEM to represent *Dirk*, *SAP* and *Karlsruhe*, as they are unique concepts for her. 4.13 shows a CDS model representing this example. The same model, encoded in RDF is shown in 4.14 on page 173.

Mapping to RDF Schema

Mapping the Relation Ontology to existing Ontologies Most of the CDS semantics are close to RDFS and can be mapped to corresponding constructs. RDFS has no way to represent the NameItem idea of human-readable strings serving as unique names. RDFS also has no inverse properties. However, RDFS does provide a class and property hierarchy, which maps well to CDS. Types in CDS map to classes in RDFS. Therefore CDS uses already *rdf:type* as the URI for `[has type]`. Sub-typing in CDS can be modelled as a special case of sub-classing in RDFS. This axiomatic triples provides the mapping:

```
cds:hasSubType  rdfs:subPropertyOf  rdfs:subClassOf.
```

In a similar way, a sub-relation in CDS maps to a sub-property in RDFS. A second axiomatic triple is needed:

```
cds:hasSubType  rdfs:subPropertyOf  rdfs:subPropertyOf.
```

Note how CDS is different from, e. g., RDFS where separate class and property hierarchies are established. In CDS, only one type hierarchy is used. The usage of the types determines the role as Relation (second part of triple) or type (first or third part in a triple). This mapping maps the CDS hierarchy to both the RDFS class *and* the RDFS property hierarchy.

More about the relation from CDS to RDFS and OWL is given in Sec. 6.4.

4.4.3. Retrieval

CDS offers by its design three parallel ways to work with personal knowledge:

1. Content of Items, e. g., simple keyword search for Item retrieval, using structural and formal knowledge only to improve ranking,
2. Relation structure for retrieval by associative browsing as well as for composing documents from existing Items, and
3. Semantics of Items and Relations for reasoning.

Navigation in CDS knowledge models NameItems allow a user to jump directly into certain nodes of their Model, similar to the world wide web, where a user can reach any node directly if she knows the URL. From there on, content can be explored by associatively following links. The same works

```
NameItems:
[Dirk], [SAP], [Karlsruhe].
Relations:
[works at]/[employs].
Statements:
[Dirk] [works at] [SAP].
[SAP] [is located in] [Karlsruhe].
```

Figure 4.13.: A simple CDS model

in CDS Models: After selecting a NameItem via its name, the user can follow Statements about that Item. The Statements can also lead her to Items that have no name, e. g., to other ContentItems, Relations or Statements.

Queries A CDS knowledge model can be queried in several ways. First, the data-model $\mathfrak{D}$ clearly separates addressable entities (Items) and content (one per Item). Textual content is thus modelled separate from structural information. All structures are stored as Statements or Triples, connecting other Items.

Second, as the model can be embedded easily in RDF, existing query languages for RDF can be reused. SPARQL (Prud'Hommeaux et al., 2007) is the most popular RDF query language and released as a *technical recommendation* from W3C. In principle, the full power of SPARQL can be used to query CDS models. However, as SPARQL is rather complex, a subset of SPARQL has been developed to allow query formulation by casual users.

CDS Query Language (CDS-QL)

Queries in CDS are basically SPARQL SELECT queries with only one projected variable. This allows them be used in set-like operations such as intersection, union and set difference.

The formal query language is based on atomic triple *patterns* $p \in P$ composed of Items i, Relations r and the wildcard $*$. Note that all Items and Relations in CDS are identified with a URI. Formally,

$$P = \{(i, r, *), (i, *, i), (*, r, i)\}, i \in I, r \in R$$

where the first parameter of the pattern denotes the source Item of a Statement, the second the Relation of a Statement and the third parameter the target Item of the Statement.

A query $q \in Q$ is either an atomic pattern, a negated query ($\neg$), or the intersection ($\cap$) or union ($\cup$) of two queries. Formally,

$$Q = \{p, \neg q, q_a \cup q_b, q_a \cap q_b\}; p \in P; q_a, q_b \in Q.$$

As an example, *all friends of Dirk living in Karlsruhe that do not work at SAP*, would be represented as the query:

$$\begin{aligned} q_1 &= q_2 \cap q_3 \\ q_2 &= q_4 \cap q_5 \\ q_3 &= \neg p_1 \\ q_4 &= p_2 \\ q_5 &= p_3 \\ p_1 &= (*, worksAt, SAP) \\ p_2 &= (Dirk, knows, *) \\ p_3 &= (*, livesIn, Karlsruhe) \end{aligned}$$

or in one query:

$$q = ((Dirk, knows, *) \cap (*, livesIn, Karlsruhe)) \cap (\neg(*, worksAt, SAP)).$$

Mapping to SPARQL The query language CDS-QL can be mapped to SPARQL as follows:

- CDS-QL patterns are mapped to SPARQL patterns by replacing each $*$ with `?var` and each ITEM or relation with its URI put between angle brackets, that is <URI(item)>.
- Negation can be represented in SPARQL via OPTIONAL and FILTER. For any pattern $p = (x, y, z)$ where one of the components (which is now called v) is a wildcard $*$, the mapping to SPARQL is

```
OPTIONAL { x y z } .
FILTER( !bound(?v) )
```

- Intersection of two patterns is expressed as simply separating the two corresponding SPARQL patterns via a single dot ("`.`").
- Union in CDS-QL is mapped to SPARQLs "UNION" keyword.

An example for a CDS-QL query mapped to SPARQL is given in Fig. 4.15 on page 174.

When running the queries, the semantics of the relation ontology $\mathfrak{R}$ need to be taken into account. Two aspects are relevant:

Inverses For each STATEMENT its inverse triple is part of the query answering process.

Type Hierarchy For the hierarchies implied by [has subtype] the transitive closure needs to be calculated and used.

4.4.4. Modelling examples

PIM example As a small example, consider a person called "Dirk Hagemann" with a birthday on April 2nd 1975. In CDS, this can be modelled as a NAMEITEM a with content "Dirk Hagemann", a CONTENTITEM b with content "02.04.1975", and STATEMENTS (a, [has birthday], b), and ([has birthday], [has super-relation], [has detail]).

Representing Node-and-Link Diagrams ITEMS and STATEMENTS allow representing node-and-link diagrams, e. g., like concept maps. Comparing CDS with nodes and links, a link is modelled as a STATEMENT. Each STATEMENT links a source ITEM with a target ITEM. The label of the link can be stored as the content of the STATEMENT. Such STATEMENTS should use [is related to] as the RELATION type for undirected links and [links to] for directed links.

Importing a Wiki To allow for collaborative usage, CDS assigns to each piece of content an author, denoted by a URI and a time stamp which records the change date of the content. This collaboration model has been inspired from wikis. Can a CDS knowledge model import the content of a wiki? Yes, in the following way: Each wiki-page a with a title $a.title$ and content $a.content$ is represented as a NAMEITEM

b with content $a.title$, a ContentItem c with its content being a STIF-representation of $a.content$[12]. A Statement $(b, [\text{is alias of}], c)$ connects the name and the content.

Fuzzy Knowledge In order to record vague knowledge as well, one could state the *truth value* of each Statement as a numeric floating point value between 0 and 1. By default, a degree of 1 (100%) could be assumed. Technically, a new [has truth value]-relation could be assigned to each Statement. The linked ContentItems would then carry the numeric value.

4.5. Summary and conclusions

This section presented the core of the thesis. Together, the CDS data-model $\mathfrak{D}$, the CDS relation ontology $\mathfrak{R}$, and the STIF model for structured text realise a number of features, which are summarised in the next section.

4.5.1. Feature summary

In brief, CDS are a lean model suitable for representing and using personal knowledge in various degrees of formalisation in a uniform fashion, allowing stepwise formalisation.

Feature 1: Addressable entities All entities in CDS are addressable. This enhances significantly the ability to define mappings to other knowledge models and transformations to other formalisms. Advanced features, such as automatic synchronisation with other people's models, versioning and access rights are also easier to implement.

Feature 2: Meta-modelling in CDS data models There are several ways in which meta-modelling can be done. The most common kind of meta-modelling is to assign model entities a type (via [has type]) and then describe this type in terms of higher-level types. CDS supports this, as no strict separation of modelling layers is required. Expressivity

A second way to do meta-modelling is to make formal statements about formal statements, which is rarely done in software engineering. CDS allows this, too, although it is up to users of the model to define semantics for it. CDS supplies only the structural ability to treat Statements as Items, and hence make Statements about Statements ad infinitum. A realistic usage scenario could be an annotation of Statements with subjective votes such as "I believe this is true/false" and then run a query under the constraint to treat only Statements from a certain author or group of authors as true.

[12] Such a representation can be obtained for many popular wiki engines with the *WikiPipes*-project.

Feature 3: Stepwise formalisation One case of stepwise formalisation is the usage of NameItems. First, a NameItem can be used like a named container. A user just links snippets of content underneath the NameItem, e. g., via [is related to]. When navigating to the NameItem, all the snippets are accessible. Later, the user might start to consider the NameItem a tag and relates things via [has tag] to it. In CDS tools that show, e. g., "tag clouds", the NameItem would now appear in the tag cloud. As the next formalisation step, the user can promote the tag to a *type*, simply by linking it via [has type] to another Item. The NameItem itself stays the same all the time and can play all three roles (named container, tag, formal type) at the same time.

Feature 4: Lean Model CDS is a quite lean model, as it consists of very few conceptual entities. The data-model has only five key elements ContentItem, NameItem, Relation, Statement and Triple. The basic XML info-set model has already seven elements (element, attribute, comment, processing instruction, entity declaration, entity reference, and document type declaration).

The relation ontology is slightly more complex, however, users can learn it gradually, as they need it. On the other hand, $\mathfrak{R}$ has only 13 Relation types defined.

Feature 5: Retrieval CDS models can be used by following links associatively, asking formal queries or performing full-text search over the content of Items. Furthermore, CDS models can be exported into other formats.

Feature 6: Degrees of Granularity CDS allows representing a wide range of granularity. Many Items, especially NameItems, are usually very short, denoting concepts. Other Items, i. e., ContentItems, can represent complete documents or parts thereof.

Feature 7: Degrees of Formality Knowledge cues in CDS can range from being very informal (plain text) to more formal (structured text) up to fully "semantified" knowledge bases. The different artefacts co-exist and can be authored and queried in a uniform fashion.

CDS allows not only structuring but also to formalising knowledge. This allows to retrieve, e. g., "white shark" using an expression like
$(?x, [\text{has type}], [\text{Lamniformes}])$ although no one ever told the system that this is true. One might just have entered
$([\text{white shark}], [\text{has type}], [\text{Lamnidae}])$ and also
$([\text{Lamnidae}], [\text{is subtype of}], [\text{Lamniformes}])$.
This allows CDS to infer that white sharks also belong to the Lamniformes. It remains the responsibility of the user to decide which content should be formalised up to which degree.

Feature 8: Multiple perspectives As structures can be transformed from STIF to CDS and vice versa, the user gets more freedom to author knowl-

edge cues in the formalism of choice. The re-use of structures in text lowers the cost of authoring structured knowledge models.

Feature 9: Consistent naming scheme CDS uses a simple, consistent naming schema for NameItems and Relations. No content must be named, but any content can be named. The only exception are Relations, which are required to be named. Conceptually, Relation names are also names from the same namespace, i.e. there cannot be a relation named 'knows' being something different from a NameItem 'knows'. The explicit handling of names as addressable entities allows meta-modelling for names, too. Is the name out-dated? Has it been replaced by another name? Does it refer to more than one entity? The user can model these things as desired.

Feature 10: Extensible relation ontology The relation ontology is extensible by a user. This allows modelling any domain and makes sure that the user is not restricted. The hierarchical design of the relation ontology allows a gradual refinement of Statements. A user starts with [is related to], then decides on the direction of the link by choosing [links to] or [is linked from]. In a next step, an sub-relation can be chosen, e. g., [has detail], [annotates] or [comes before] (or their inverses). Alternatively, a new sub-relations of [links to] (or its inverse) can be created. With each refinement step, more precise semantics are present in the model. The soft migration from [is related to] to more precise relations makes it easier for the user to create such formal Statements.

Feature 11: Semantic queries Queries allow exploiting the semantic statements modelled in a personal knowledge model. Examples for queries are given in Sec. 4.4.3.

Feature 12: Context tracking Each Item has meta-data about the creation date and its author. This allows them to become more powerful knowledge cues, as the meaning is often clearer in a context of other knowledge cues. Items created by the system are marked with a different author and are thus clearly distinguishable.

Feature 13: Hyper-Linking The link is one of the five core CDS relation types. As a knowledge model represents the content of many documents, represented as many, interlinked small Items. Thus the classical document boundary is crossed: One Item can be linked from many other Items (like transclusion in hypertext research). As all Items are addressable, links can go from and to any Item. Using STIF, Item can also create links to any other WWW resource.

Feature 14: Ordering One of the five CDS relations types is [comes before] and [comes after] which models partial order. Used consequently, a partial order can also be a total order.

Feature 15: Hierarchy Hierarchical modelling is a core principle in CDS. CDS provides several hierarchical RELATION types.

4.5.2. Summary

CDS, consisting of a data-model ($\mathfrak{D}$), a relation ontology ($\mathfrak{R}$) and model for structured text (STIF), is a general-purpose modelling language. It emphasises modelling in text (STIF), structured text (STIF) and typed relations (STIF, $\mathfrak{D}$, $\mathfrak{R}$) over graphical modelling. CDS can handle semi-structured data (STIF, $\mathfrak{D}$, $\mathfrak{R}$). Authoring in structured text is a means to cut down the costs of externalising structured artefacts, which in turn lowers the cost of retrieval.

To sum up, CDS can

- (1) act as a formalism for recording, managing, and sharing personal knowledge,
- (2) bridge the gap between informal/unstructured information and fully formal semantic models in PKM;
- (3) serve as the least common denominator for knowledge exchange among humans and among different tools, and
- (4) encode vague structures (e. g., "this is nested within that, but I can't say how").
- (5) CDS can act as a guideline for PKM tools: Each tool should be able to create, represent and manipulate at least the structural elements defined in CDS.

Define ITEMS (in this case: NAMEITEMS):

```
<dirk>  rdf:type        cds:NameItem.
<dirk>  cds:hasContent  "Dirk".
<sap>   rdf:type        cds:NameItem.
<sap>   cds:hasContent  "SAP".
<ka>    rdf:type        cds:NameItem.
<ka>    cds:hasContent  "Karlsruhe".
```

Define Relations and their inverse Relations:

```
<wa>     rdf:type        cds:Relation.
<wa>     cds:hasContent  "works for".
<wa>     cds:hasInverse  <emp>.
<emp>    rdf:type        cds:Relation.
<emp>    cds:hasContent  "employs".
```

Make two Statements:

```
<s1>     rdf:type       cds:Statement.
<s1>     rdf:subject    <dirk>.
<s1>     rdf:property   <wa>.
<s1>     rdf:object     <sap>.
<s2>     rdf:type       cds:Statement.
<s2>     rdf:subject    <sap>.
<s2>     rdf:property   cds:hasContext.
<s2>     rdf:object     <ka>.
```

Figure 4.14.: A CDS model represented in RDF (N3 Syntax)

Given this URI mapping:

$worksAt \mapsto$ http://example.com/worksAt
$SAP \mapsto$ http://example.com/SAP
$Dirk \mapsto$ http://example.com/Dirk
$knows \mapsto$ http://example.com/knows
$livesIn \mapsto$ http://example.com/livesIn
$Karlsruhe \mapsto$ http://example.com/Karlsruhe

the CDS-QL query *all friends of Dirk living in Karlsruhe that do not work at SAP* results in the SPARQL patterns:

$p_1 \mapsto$?var,
<http://example.com/worksAt>,
<http://example.com/SAP>
$p_2 \mapsto$ <http://example.com/Dirk>,
<http://example.com/knows>,
?var
$p_3 \mapsto$?var,
<http://example.com/livesIn>,
<http://example.com/Karlsruhe>

The full CDS-QL example query as a SPARQL query:

```
SELECT ?var WHERE {
  <http://example.com/Dirk>
    <http://example.com/knows>
      ?var .
  ?var
    <http://example.com/livesIn>
      <http://example.com/Karlsruhe> .
  OPTIONAL {
    ?var
      <http://example.com/worksAt>
        <http://example.com/SAP>
  } .
  FILTER( !bound(?var) )
}
```

Figure 4.15.: Example for mapping CDS-QL to SPARQL

5. Realisation

This chapter presents a realisation of the concepts presented in Chapter 4. The realisation is split into two main parts: First, the CDS API and a reference implementation of the CDS model (CDS-RI), targeted for developers. Second, the PKM tool *Hypertext Knowledge Workbench* (HKW), targeted for end-users, is presented.

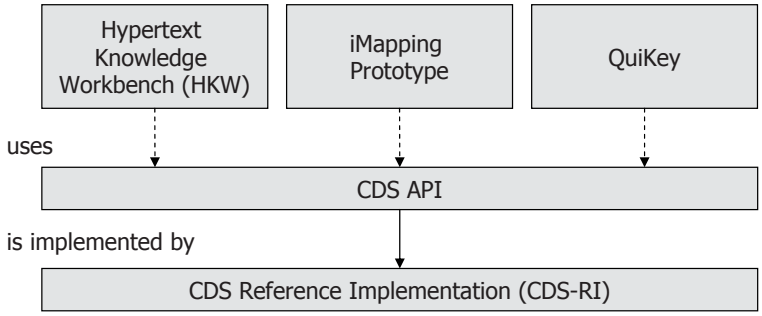


Figure 5.1.: High-level view on the CDS software eco-system

Fig. 5.1 gives a high-level overview on the realised architecture. Three different tools access the CDS-RI through the same CDS API. The different tools and different implementations are detailed in this chapter: *Hypertext Knowledge Workbench* is a browser-based prototype described in Sec. 5.2. *iMapping* is a desktop application based on a zooming user interface metaphor. It is described in Sec. 5.3.1. *QuiKey* is a command-line-like desktop application, described in Sec. 5.3.2.

5.1. CDS Reference Implementation

This section presents the CDS *Application Programmer Interface* (API) and CDS reference implementation (CDS-RI).

Generic architecture

The architecture of CDS-RI consists of three parts: The **content layer** needs to be able to persist CDS content, which is a URI mapped to a content-type plus the actual content itself. The content itself may be a string or a binary format. The **structure layer** manages structured data such as Relations and their inverses, Statements and their components, and the metadata of Items. The **semantics layer** is a component responsible for interpreting Statements and inferring new Triples.

This separation into layers allows the content layer to stay very simple and hence easy to implement, use and maintain. The structure layer becomes smaller and hence is easier, e. g., to keep in memory for knowledge models

up to a certain size. Furthermore, each lower layer can be re-used without the higher layers.

Application Programmer Interface (API) The *Conceptual Data Structures* API is a Java API for creating and manipulating CDS Models. The conceptual model of CDS itself has also been described in (NEPOMUK Consortium et al., 2008) and (Völkel, 2007b).

The API consists mainly of two layers, namely the CDS model layer (corresponding to $\mathfrak{D}$) and the CDS semantics layer (corresponding to $\mathfrak{R}$ and STIF). The CDS model layer is the *content* and *structure* layer. It represents the state of a CDS data model ($\mathfrak{D}$). This layer can be used without the higher CDS semantics layer.

Content and structure – The CDS data model layer

The concept of a re-usable, stand-alone *Semantic Web Content Repository* has been described by Völkel (2007b).

The main interface, `IModel` gives access to all operations described in Sec. 4.1.4. Furthermore, the IModel can be queried for all NameItems, all ContentItems etc. There are also some query operations to facilitate auto-completion in user interfaces such as `getAllNameItemsMatching(String regex, long limit)` which returns all NameItems up to the given *limit* that match the given regular expression. Each Item type is represented by its own interface, arranged in an inheritance hierarchy. The API is thus completely type-safe, making heavy use of Java Generics. Furthermore, the API is based on *RDF2Go* which has been developed to abstract away different triple store implementations.[1] For binary content a component called *BinStore* is used.[2] On top of RDF2Go, a component called RDFReactor (Völkel, 2006) has been developed[3] to generate object-oriented wrappers for accessing triple-based RDF data.

Semantics layer

The semantics layer is an extension of the CDS model layer. It provides mainly three additions: Constants for all CDS built-in Relations, handling

[1] RDF2Go is a triple store abstraction layer, developed by the author of this thesis, which allows switching between, e. g., *Sesame* and *Jena* implementations at startup time. RDF2Go is used by several other projects such as SemVersion, Aperture, Hyena, NEPOMUK, SemFS and Theseus. RDF2Go is available under an open source BSD license at `http://semanticweb.org/wiki/RDF2Go` (accessed 06.01.2010). In November 2009, the RDF2Go home page has had over 22.800 page views since its creation in June 2006.

[2] The *BinStore* component, also created by the author of this thesis, is modelling a map from URIs to blobs (binary large objects). Different from other content management APIs for Java, BinStore allows random-access to the binary blobs, which makes it suitable for layering, e. g., virtual file systems on top of it. For each entry, the BinStore records content-type and creation date. BinStore is a re-usable component. The API together with a file-backed implementation is available under a liberal BSD license at `http://semanticweb.org/wiki/BinStore` (accessed 06.01.2010). BinStore is used in the *Semantic File System* (SemFS) (Bloehdorn et al., 2006).

[3] RDFReactor has been developed by the author of this thesis from 2004 to 2009. It is available at `http://semanticweb.org/wiki/RDFReactor` (accessed 06.01.2010) under a BSD open source license. The home page has been accessed over 11.300 times as of November 2009. RDFReactor is used in SemFS, NEPOMUK and the MEDICO project, which is a part of the Theseus program.

of wiki syntax and STIF, and support for queries with inferencing according to the CDS semantics.

Code statistics

The Java source code consists of 48 files with just 1,370 lines of code (plus 2,243 lines of comments, all excluding blank lines).

The complete CDS API description can be found online at `http://semweb4j.org/site/cds.api/apidocs/` (accessed 06.01.2010).

Reference Implementation (CDS-RI) The reference implementation is used in HKW, iMapping and QuiKey. The CDS API has been implemented several times during the course of this thesis. In order to justify and illuminate the final design, a little bit of history is necessary:

RDF-based implementation This implementation used a local RDF model for the state (running in *Sesame*), RDFReactor for the mapping to CDS concepts and an adapted rule-based RDFS reasoner for the inference.

NEPOMUK implementation During the course of NEPOMUK, an integration with NEPOMUKs back-end was needed.

The goal of $\text{RI}_{\text{NEPOMUK}}$ was a CDS API implementation that could transparently read NEPOMUK data, interpret it as a CDS knowledge model, manipulate it in CDS tools and write it back to the NEPOMUK store according to the semantic desktop ontologies.

NEPOMUK has the architectural concept to connect third-party services, such as a CDS-based tool, via a web service interface. This has the advantage that a malicious or malfunctioning component in the third-party part cannot damage the NEPOMUK server part.

Minimising HTTP calls

However, the web service interface turned out to be a serious performance bottleneck, especially when used in a fine-granular way. Unfortunately, the design of RDFReactor results in a high number of very fine-granular calls. To remedy for this problem, a work-around has been designed. At startup time, the $\text{RI}_{\text{NEPOMUK}}$ loads all relevant data with one special SPARQL query from the NEPOMUK server. At runtime, all calls to the underlying RDF store are sent both to the internal store as well as to a dedicated buffer. Two new commands `save` and `load` have been introduced in the CDS API to give tools control about when to commit the buffer back to the NEPOMUK server and when to reload data from it. Unfortunately, even with this design where HTTP calls had been reduced to the absolute minimum, the overhead of transporting data over HTTP was still too high and resulted in lagging user interfaces.

In 2008, applications based on RDF requiring very many or very large queries and update operations did not run fast enough over HTTP, even when client and server were running on the same desktop machine. The technical details of these problem are described in (NEPOMUK Consortium et al., 2008, Sec. 4.5.2).

1. $(x, \text{nrl:inverseProperty}, y) \wedge (a, x, b) \Rightarrow (b, y, a)$
2. $(p, \text{nrl:inverseProperty}, q) \wedge$
   $(p, \text{rdfs:subPropertyOf}, t) \wedge$
   $(t, \text{nrl:inverseProperty}, u) \Rightarrow$
   $(q, \text{nrl:inverseProperty}, u)$
3. $(p, \text{nrl:inverseProperty}, q) \Rightarrow$
   $(q, \text{nrl:inverseProperty}, p) \wedge$
   $(p, \text{rdf:type}, \text{rdf:Property}) \wedge$
   $(q, \text{rdf:type}, \text{rdf:Property})$

Figure 5.2.: Examples for rules used in CDS inferencing that are not included in RDFS or OWL

JavaScript implementation Within HKW most parts of CDS have been re-implemented for performance reasons: A triple store, an inference engine, unique names, etc. However, the runtime performance of the JavaScript-based reasoner was still too slow for an acceptable performance. Although not systematically tested, JavaScript seems to have a penalty factor of about 10 in terms of memory requirements and run time compared to Java.

After these three mistakes, the final, fourth implementation has been created. It is not integrated into NEPOMUKs back-end. As it uses custom data structures for maximal performance, it is called the *native* CDS implementation ($\text{RI}_{\text{Native}}$).

Native implementation No internal RDF store is used, instead, all functionality has been implemented from scratch, for better runtime performance and memory usage. It has the following components, from top to bottom:

Facade API for ease-of-use

MemCdsModel implements the convenience methods declared in the CDS API and delegates all non-trivial calls to a MemCdsCoreModel.

MemCdsCoreModel uses a MemModel for persistence and a MemInferenceModel for inferencing.

Semantics layer implementation

MemInferenceModel listens for change events from a MemModel and keeps an index of inferred STATEMENTS up to date. The basic concept of the MemInferenceModel is thus similar to a *Truth Maintenance System* (Doyle, 1987). Inference is triggered at query-time, to prevent calculating inferences. E. g., if a STATEMENT is created and deleted afterwards, it would be a loss of performance to calculate inferences after the STATEMENT has been created.

The inference engine is based on a rule-based RDFS-inference engine from the OpenRDF (*Sesame*) project. The inference engine uses incremental

materialisation, that is, when adding a STATEMENT, very few rules are triggered.

For every STATEMENT that is inserted, a number of – potentially recursive – further add-TRIPLE events are triggered, according to hard-coded inference rules. The process always terminates, as there are no infinite recursions in CDS semantics and each inferred TRIPLE is marked as such.

However, after STATEMENT deletion the complete inference cache has to be re-materialised. [4] The OpenRDF-RDFS-algorithm has been reduced to calculate only the inferences required by CDS semantics. This improves performance. Furthermore, some rules have been added to cover the complete CDS semantics. These rules are depicted in Fig. 5.2.

Content and structure layer implementation

MemModel is the implementation of the CDS data layer in the CDS API. It manages listeners on change events, provides methods for all SWCM-related tasks such as adding or deleting any kind of ITEM, and allows querying the data. MemModel delegates persistence to BinStore and MemStore.

MemStore represents the state of a CDS model (together with the data in BinStore). It maintains indices from name to NAMEITEM, from URI to ITEM, and a TripleIndex. State in MemStore can only be changed by change events, which ensure a very compact API, abstracting away many variants for creation ITEMS, supported by MemModel.

MemStore supports optional GZip-compression of strings in CONTENT-ITEMS to preserve memory. However, compressed strings cannot be matched against regular expressions.

For persistence a custom XML serialisation has been created for MemStore, using *XStream*[5]. The resulting XML-based persistence format is depicted in Appendix A.4.

Export to RDF

For interoperability, CDS-RI has an **export function to RDF**. This export can be customized to use certain URIs for certain ITEMS by adding STATEMENTS of the form $(x, [\text{export as}], y)$. The content string of y is used as the export URI for x. The built-in properties are mapped to SKOS properties, this allows generic SKOS browsers to browse CDS' `[has context/has detail]` structures. For the SKOS export, content is not exported as `cds:hasContent` but as `rdfs:label`.

TripleIndex is a space-efficient index from triples to STATEMENTS. It has to (a) store triples and (b) index the triple data for faster access. To allow arbitrary query patterns, several indices are used. Inspired by a paper by Harth and Decker (2005), a minimal number of indices has been used. Three indexes are enough to answer all eight triple patterns (cf. Tab. 4.2 on page 139). One possible mapping (the one implemented) from triple patterns to indexes is given in Tab. 5.1.

Source code statistics

The code of the implementation consists of 32,956 non-whitespace lines (of which 10208 are comments) in 267 files.

[4] Of course, this is a performance bottleneck. The same forward-chaining strategy is used, e. g., in OpenRDF (*Sesame*) project. Backward-chaining in general has slower query answer performance, although it handles delete operations faster. In the end, such decisions are always a trade-off. As adding ITEMS to a knowledge model will be much more frequent compared to deleting ITEMS, forward-chaining was chosen.

[5] `http://xstream.codehaus.org/` (accessed 06.01.2010)

Pattern	Index
(s, r, t)	Index 1: $s \rightarrow (r \rightarrow (t \rightarrow \text{STATEMENT}))$
$(s, r, *)$	Index 1: $s \rightarrow (r \rightarrow (t \rightarrow \text{STATEMENT}))$
$(s, *, *)$	Index 1: $s \rightarrow (r \rightarrow (t \rightarrow \text{STATEMENT}))$
$(*, *, *)$	Index 1: $s \rightarrow (r \rightarrow (t \rightarrow \text{STATEMENT}))$
$(*, r, t)$	Index 2: $r \rightarrow (t \rightarrow \text{STATEMENT})$
$(*, r, *)$	Index 2: $r \rightarrow (t \rightarrow \text{STATEMENT})$
$(s, *, t)$	Index 3: $t \rightarrow (s \rightarrow \text{STATEMENT})$
$(*, *, t)$	Index 3: $t \rightarrow (s \rightarrow \text{STATEMENT})$

Table 5.1.: Mapping query patterns to indexes

5.2. Hypertext Knowledge Workbench

This section presents a tool that allows end-users to work with CDS-based knowledge models. The tool is called *Hypertext Knowledge Workbench* (HKW) and is based on the CDS RI presented in the last section. The tool has been built for validating the CDS model (cf. Ch. 6).

5.2.1. User interface

The user interface is presented in three scenarios: Note taking, personal social network and the HKW internal help system.

Scenario: Note taking

The user interface is introduced with a scenario of a fictional biological scientist called Linda, who manages her personal notes with HKW.

The next pages give a step-by-step explanation of using the Hypertext Knowledge Workbench.

Step 1: Creating initial items

Fig. 5.3 shows the initial screen after starting HKW for the first time. Linda starts a new project on "Shark Research" so she enters this into the top bar and hits enter. As a result (c.f. Fig. 5.4) she just created a new NameItem with the name "Shark Research". If she later repeats the same procedure, she is being taken to the same NameItem, as NameItems are identified by their name.

The nature of the Item is indicated with the little icon on the top left corner above. It shows an sticky note with an "N" for NameItems. For other item types it shows an empty sticky note for a ContentItem, a pink arrow for a Relation, and two sticky notes connected via a pink arrow to symbolise a Statement.

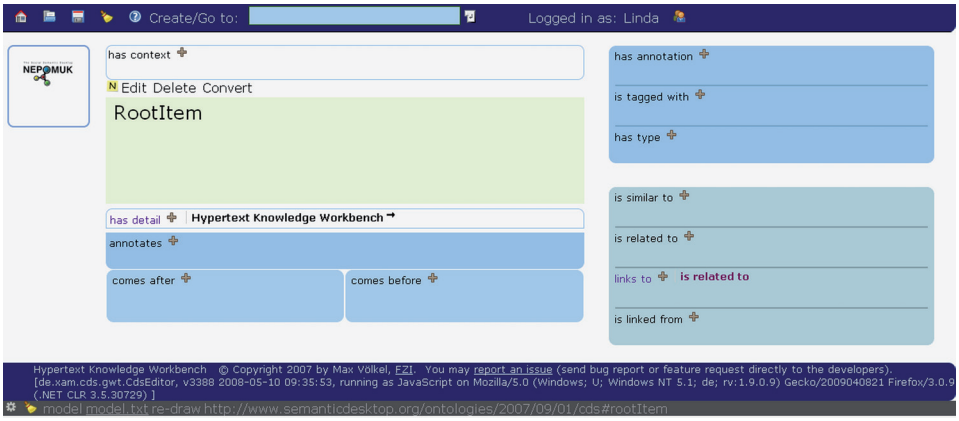


Figure 5.3.: The initial screen in HKW (step 1)

Step 2: Creating linked items

Linda's primary research subject is the "'Great white shark", so she adds it within the *Shark Research* context by clicking on the plus-icon behind "has detail".

A pop-up widget with a larger font for entering text comes up (cf. Fig. 5.5). She enters *Great white shark* and commits by clicking the *Enter*-button or by using the keyboard shortcut `Ctrl-Enter`. She could also have cancelled the creation of a new detail Item by clicking the red X or pressing `Esc` on her keyboard.

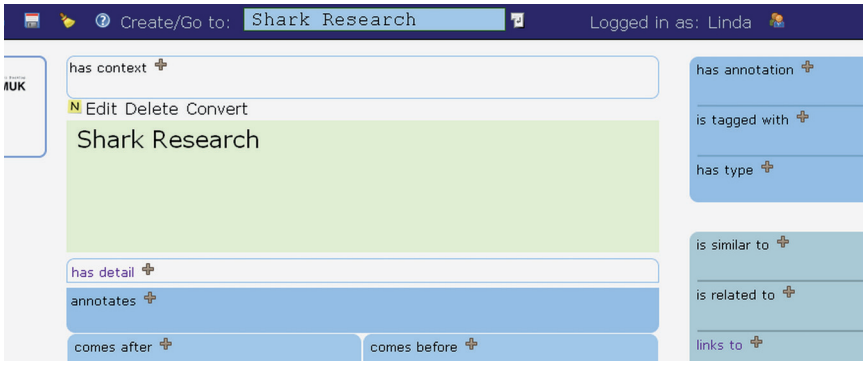


Figure 5.4.: Creating a new Nameitem (step 2)

Step 3: Navigation

The new ITEM *Great white shark* is placed in the *has detail* box, just where Linda created it. A little black arrow behind the ITEM (as depicted in Fig. 5.3 behind the word *Hypertext Knowledge Workbench*) allow her to navigate to the STATEMENT ([Shark research], [has detail], [Great white shark]). This view, the STATEMENT change widget, is depicted in Fig. 5.6. Here she

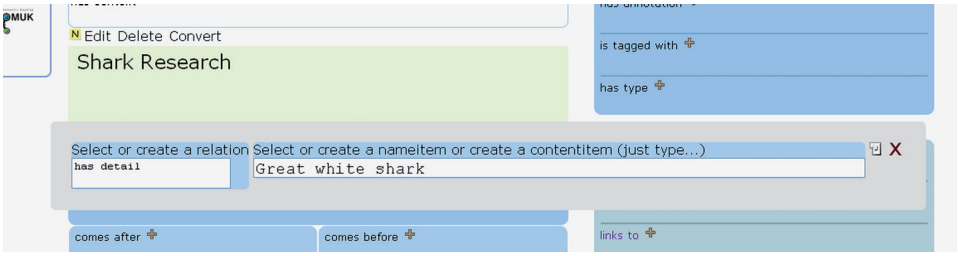


Figure 5.5.: Creating a linked Nameitem (step 3)

can navigate to all three components of the STATEMENTS (top row) or change each component of the STATEMENT. In the example scenario, there is nothing that makes sense to be changed.

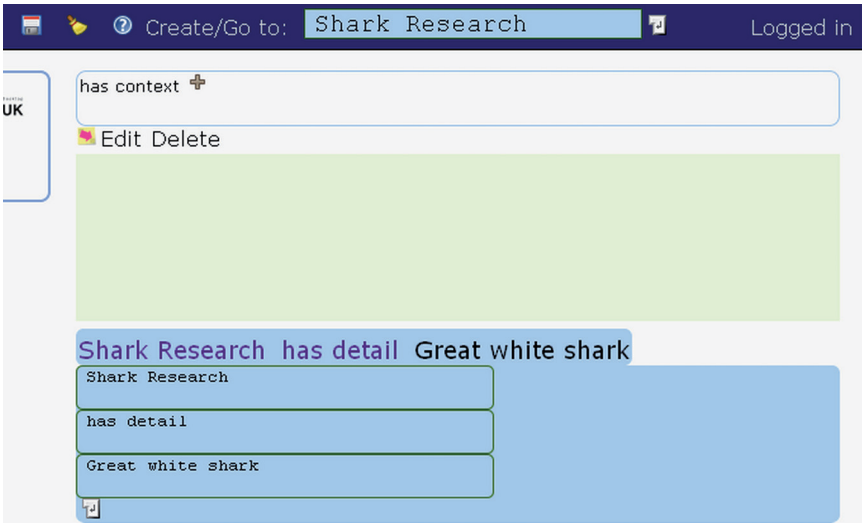


Figure 5.6.: Statement change widget (step 3b)

Step 4: Navigation

Linda wants to add more information about the great white shark, so she navigates to it by clicking on that name. At each time, one ITEM is the *central item* of the HKW application. Now the central ITEM shifted from *Shark Research* to *Great white shark*, as depicted in Fig. 5.7. She now sees *Shark Research* as the context of the current ITEM. As explained in Sec. 4.1, [has context] is the inverse RELATION of [has detail].

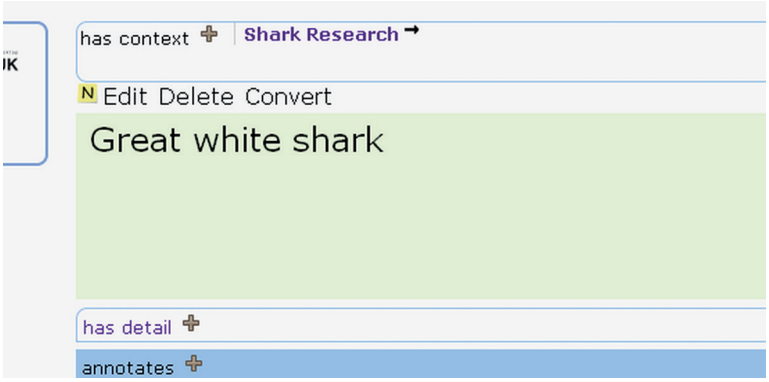


Figure 5.7.: Navigation to "Great white shark" (step 4)

Auto-completion for navigation

Alternatively, Linda could have entered "shar..." in to HKW's address bar, to see an auto-completion list with all NAMEITEMS or RELATIONS containing the character sequence "`shar`" (cf. Fig. 5.8). In the list, she can navigate with cursor keys or use a single mouse-click.

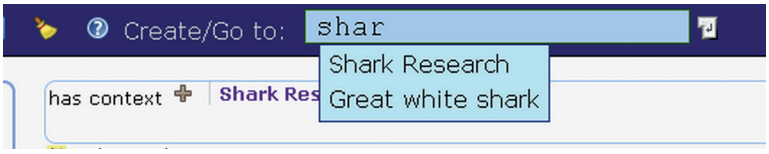


Figure 5.8.: Auto-completion after entering "shar" (step 4b)

Step 5: Creating relations

Next, Linda wants to express that this shark has a particular sense. She considers this as a kind of detail, but a special one. She enters the usual ITEM creation dialog, changes "has detail" to "has sense" (cf. Fig. 5.9), enters "Ampullae of Lorenzini" on the right side and commits.

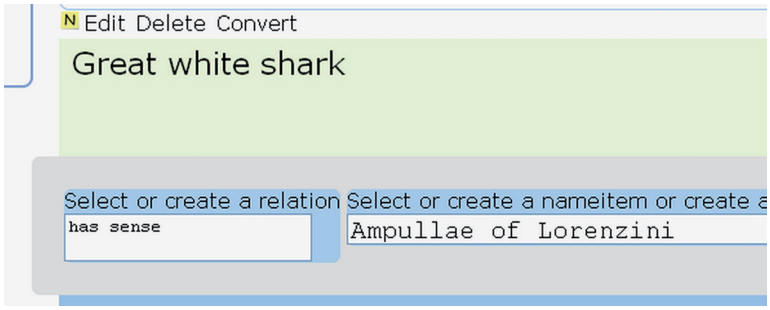


Figure 5.9.: Extending the relation ontology with custom relations (step 5)

This creates a new NAMEITEM with the name *Ampullae of Lorenzini*, a new RELATION with the name *has sense* and a STATEMENT ([Great white shark], [has sense], [Ampullae of Lorenzini]). Additionally, because Linda added the ITEM on the [has detail]-box, the new RELATION has been stated as a sub-RELATION of [has detail] (cf. Fig. 5.10).

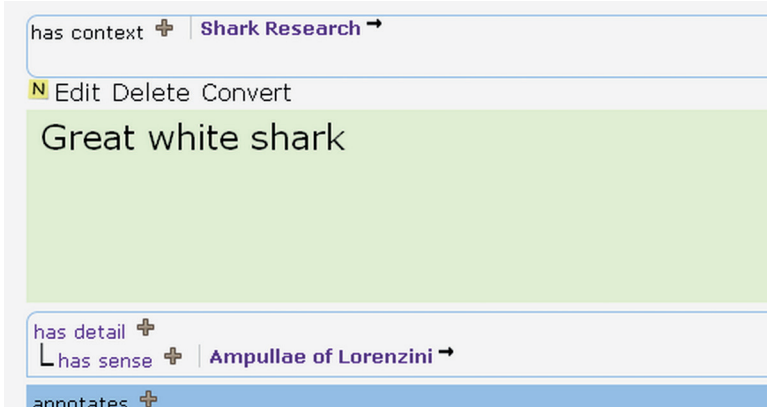


Figure 5.10.: The new item, relation and statement (step 5b)

Inverse relation naming

As each RELATION must have an inverse RELATION, HKW created one automatically. The inverse RELATION name is *relation-name*`-inverse`. In the common case that a name has the form *has relation-name* (as in the example), HKW generates `is relation-name of`. A similar algorithm is used by the semantic web browser tabulator[6].

[6] `http://www.w3.org/2005/ajar/tab` (accessed 06.01.2010)

Step 6: Creating content items

After having entered several other facts, Linda stumbles over an interesting story. She decides to add it as a generic detail to the white shark ITEM. She clicks the plus icon again and starts to type. As soon as the text gets so long that it would require two lines, the text entry widget expands (cf. Fig. 5.11) and allows Linda to paste the complete story.

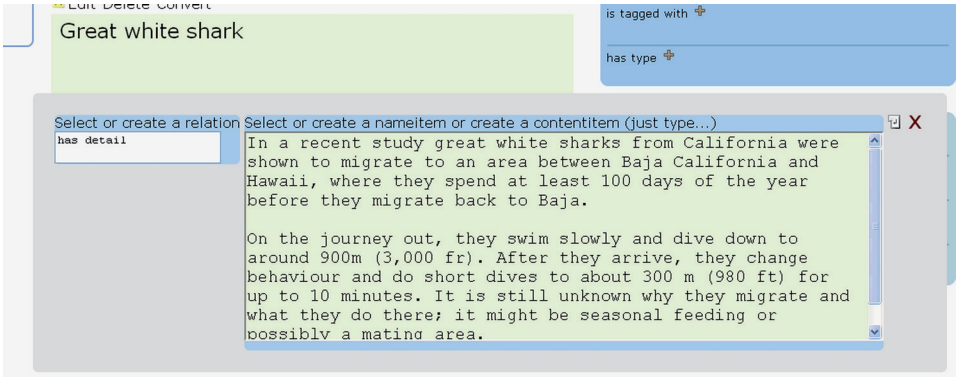


Figure 5.11.: Adding a content-item (step 6)

Linda re-reads the story and spots "100 days". This is important for her work, so she decides to use some wiki formatting to make this stand out. She navigates to the new CONTENTITEM by clicking on it (cf. Fig. 5.12).

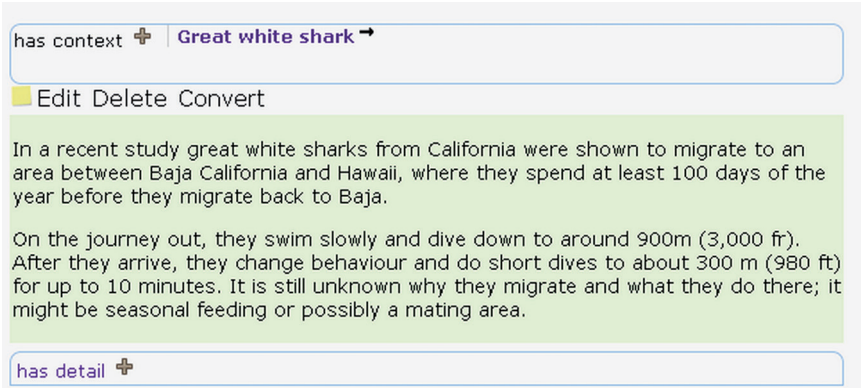


Figure 5.12.: Navigating to the new content-item (step 6b)

Generic item operations

Linda can now *edit* or *delete* the ITEM by clicking `Edit` or `Delete`. These operations are available for all ITEMS. For a CONTENTITEM, a special operation to *convert* it to a NAMEITEM is offered. However, converting to a NAMEITEM would truncate the content to make it suitable for a name, i. e., short enough and not containing line breaks. NAMEITEMS can also be converted to CONTENTITEMS. This makes it easier to add new content from the navigation bar.

Step 7: Semantic wiki functionality

Linda clicks on "edit" to enter the *edit* mode of the ContentItem (cf. Fig. 5.13). She re-formats the text "`at least 100 days`" to "`**at least 100 days**`". This renders that part of the text in bold. Linda spots that the white sharks dive down to 900m. She is not sure this is the *maximal* dive depth, but at least this is a depth the sharks can dive. She decides to create a new Statement right from the text, like in a semantic wiki. The formats "`around 900m`" to "`around [can dive::900m]`" and saves. The result is shown in Fig. 5.14.

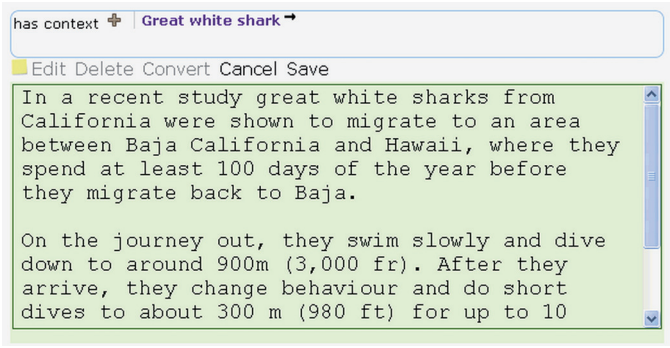


Figure 5.13.: Editing a content-item (step 7)

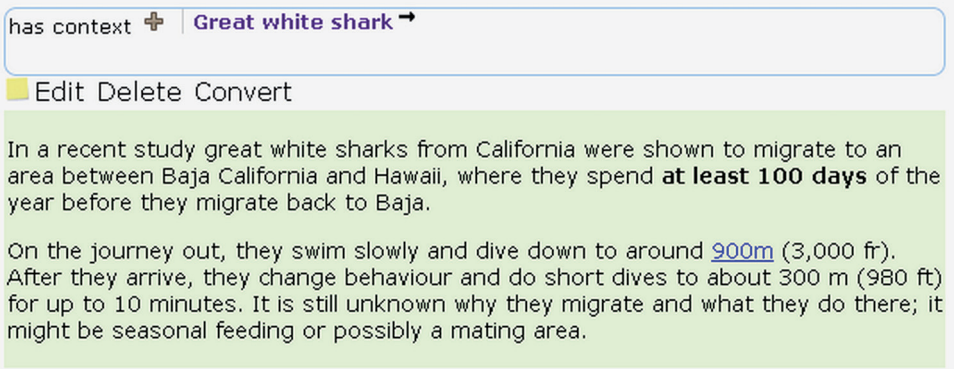


Figure 5.14.: A formatted content-item (step 7)

The underlined "900m" indicate a hyper-link. Linda navigates back to the context *Great white shark* and sees a new Statement rendered on the right side: ([Great white shark], [can dive], [900m]). She sees that HKW made the new Relation [can dive] a sub-Relation of [links to], when she saved the edited text ("...at least 100 days..."). The Relation [links to] is the default super-Relation for all Relations created from parsing wiki text.

Step 8: Meta-modelling

Linda tags the white shark as "dangerous" and records the biological family "Lamnidae". Then she navigates to *Lamnidae* and clicks on the plus behind [has context]. She changes the relation to [is subtype of] and enters "Lamniformes" into the second field. Now she wonders if she will ever remember that *Lamnidae* is the biological family and *Lamniformes* the biological order. She uses meta-modelling to type *Lamnidae* with "family (biology)". She chooses the name "family (biology)" to not confuse it with here private "family" tag that she knows she created earlier. The result is shown in Fig. 5.15. Linda continues to enter broader biological classifications as super-types.

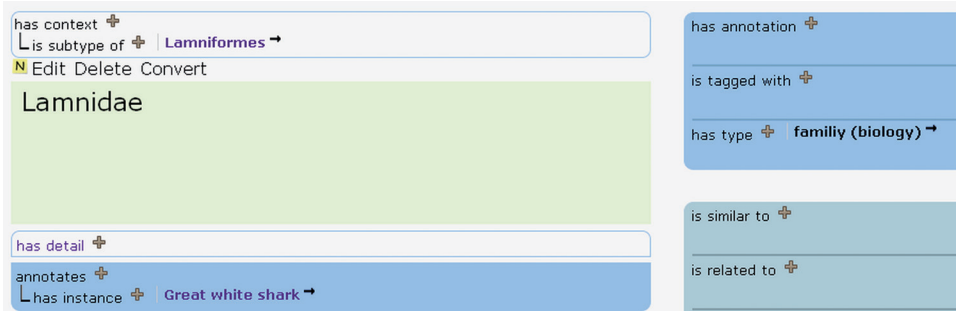


Figure 5.15.: Meta-modelling in HKW (step 8)

Step 9: Inference

Navigating back to the *Great white shark*, she sees the super-types in light gray in the [has type] box on the right (cf. Fig. 5.16). These *inferred* ITEMS have no black arrow. They are not linked to the *Great white shark* by a STATEMENT, but follow indirectly from other STATEMENTS in Linda's model. Here the transitive closure over her type-hierarchy is the cause.

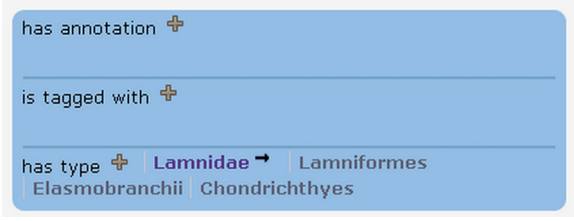


Figure 5.16.: Meta-modelling in HKW (step 9)

Step 10: Vocabulary re-use

Linda adds an alias-name and the fact that white sharks eat tuna fish. She investigates a bit and finds out that the largest kind of tuna fish, the "Thunnus thynnus" can get as large as 4,58m. When she is about to enter "has maximum size" the auto-completion list shows her already after entering just "ha" all relations she used before as sub-relations of [has detail] (cf. Fig. 5.17). This supports consistent usage of her own vocabulary.

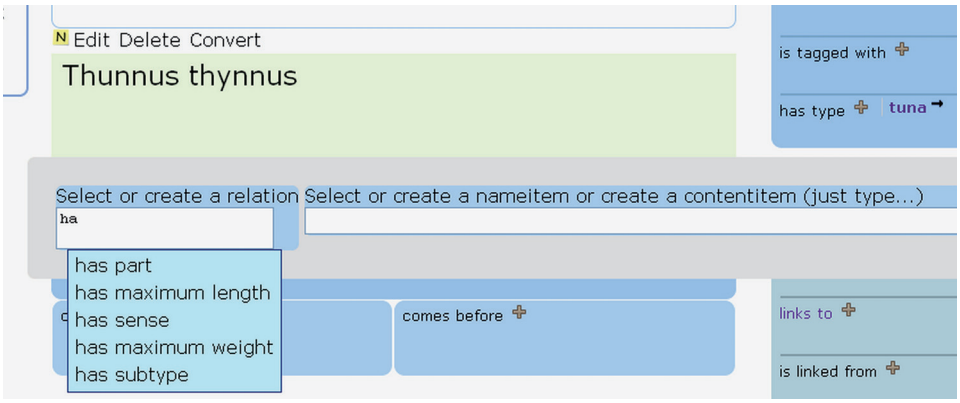


Figure 5.17.: Auto-completion for relations restricted by relation ontology supports consistent usage of terms (step 10)

Step 11: Auto-linking

Linda adds some places where white sharks live. Among them is "California". When she goes back to the study which she pasted earlier, she finds the word "California" linked to that NameItem. HKW auto-links all expressions in a text that match the content of a NameItem. Matching is done from left to right, longer NameItems match first. Fig. 5.18 shows the changed wiki formatting with bold formatting and automatically created links.

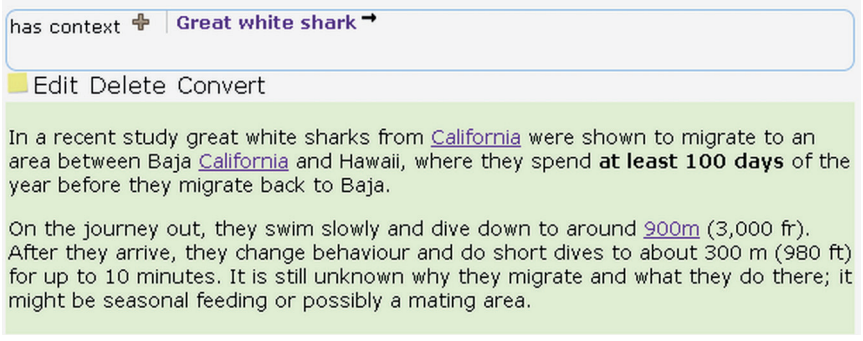


Figure 5.18.: Auto-linking of existing name-items and relations (step 11)

Step 12: Editing the relation ontology

Linda adds the *Basking shark* as a related ITEM by clicking on the plus icon behind [is related to]. The result is shown in Fig. 5.19.

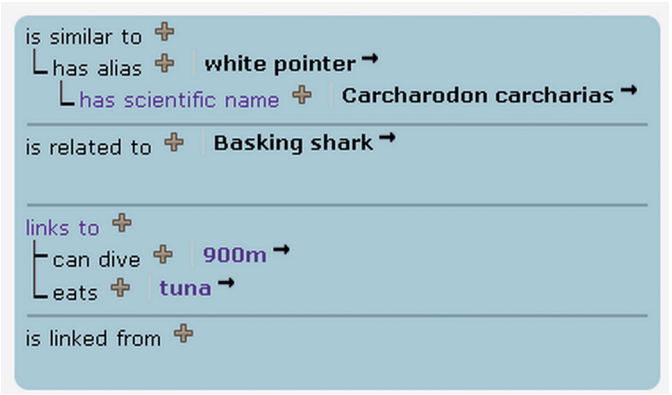


Figure 5.19.: Adding a related item (step 12)

She finds out that the basking shark is generally even larger than the great white shark, although she cannot find an exact value for the maximum size if the basking shark. So she changes the STATEMENT ([Great white shark], [is related to], [Basking shark]) into ([Great white shark], [is smaller than], [Basking shark]) via the statement widget. She clicks the black arrow and changes the text "is related to" to "is smaller than". She goes back to the white shark an sees that [is smaller than] (which she just created) became a sub-relation of [links to]. She decides to change the sub-relation from [links to] into [comes before]. Auto-completion helps her to save some keystrokes. The final screen is shown in Fig. 5.20.

Note that the order of ITEMS rendered in the [has detail]-pane takes STATEMENTS between these ITEMS into account that make use of [has before] or [has after]. Such STATEMENTS influence the order in which the ITEMS are listed. I. e., if there would be several detail-snippets under *Great white shark*, they would be display in an order which is as consistent as possible with [comes before]-STATEMENTS among these details.[7]

[7] Of course, this algorithm to create a total order out of a set of pairwise comparisons has to deal with underspecified order (random result for these pairs) and contradictions (resolved by taking into account as many pairs as possible).

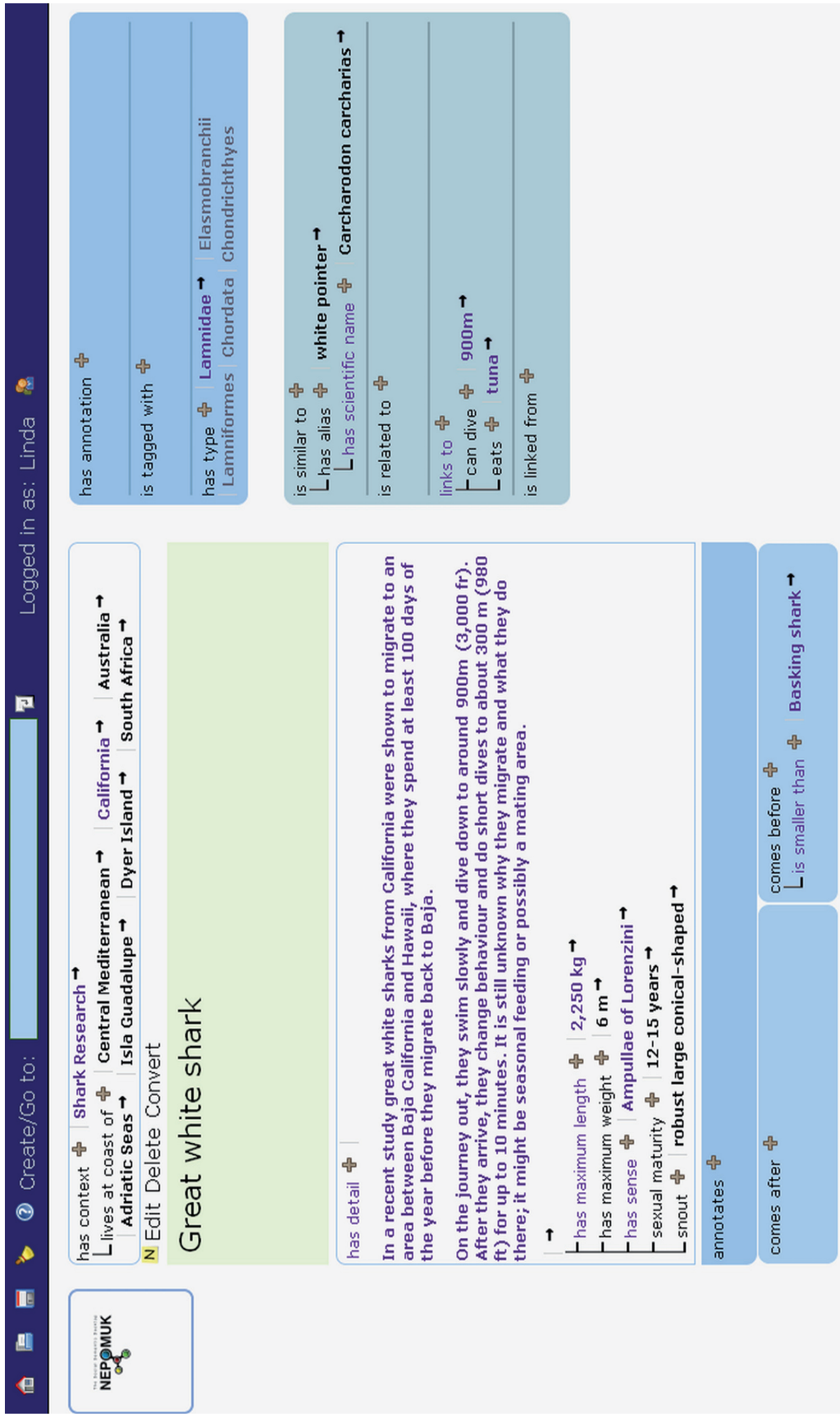


Figure 5.20.: Final screen about the Great white shark (step 12b)

Scenario: HKW help system

The HKW tool has a built-in help system that is itself represented in CDS. When the user clicks on the help icon (fifth menu icon), the cursor turns into a cross-hair and the user can click for help on each button. Linda first clicks on the help icon, then on the icon with the house on it. The result is shown in Fig. 5.21.

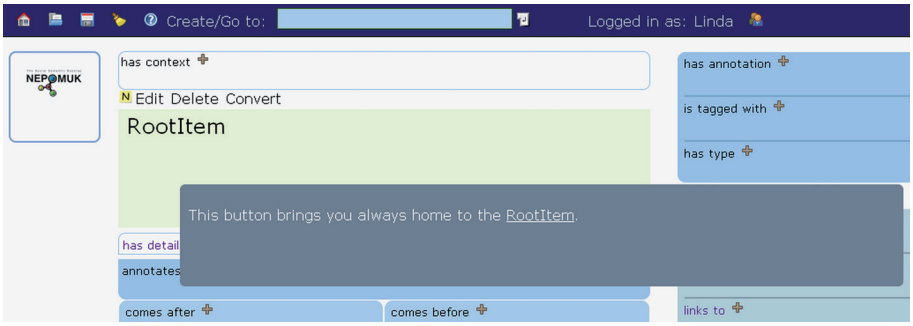


Figure 5.21.: HKW's built-in help system

Behind the scenes, the rendered content is a ContentItem with URI `http://www.semanticdesktop.org/ontologies/2007/09/01/cds:hkw-help-HomeButton` and can be edited like any other ContentItem.

Scenario: Personal Social Network Management

In this scenario, Linda records notes about a friend called *Dirk Hageman*. Fig. 5.22 shows a screenshot of HKW with Linda's model of Dirk Hageman.

Note the mix of formal data (type: person) and informal data (is tagged with: friend, has annotation: a very nice guy), as well as the structured data that does not fit in usual PIM software data models such as [is supervised by].

5.2.2. Implementation

High level design decisions

This section details the technical realisation of the user interface. To allow for easy deployment and ubiquitous usage, HKW was designed to run in a browser. A rich user experience in a browser, i. e., one without constant waiting for the server requires heavy usage of JavaScript on the client. Coupled together with an intelligent caching strategy, this design is often called AJAX (cf. 2.4).

Google Web Toolkit for developing JavaScript

Technology choice The HKW prototype has been realised with the *Google Web Toolkit* (GWT), an open-source AJAX-enabled web user interface toolkit by Google Inc. The HKW prototype uses GWT widgets and some custom-build widgets. Styling is done via *Cascading Stylesheets* (CSS). GWT applications are web-applications, which can run in any servlet container.

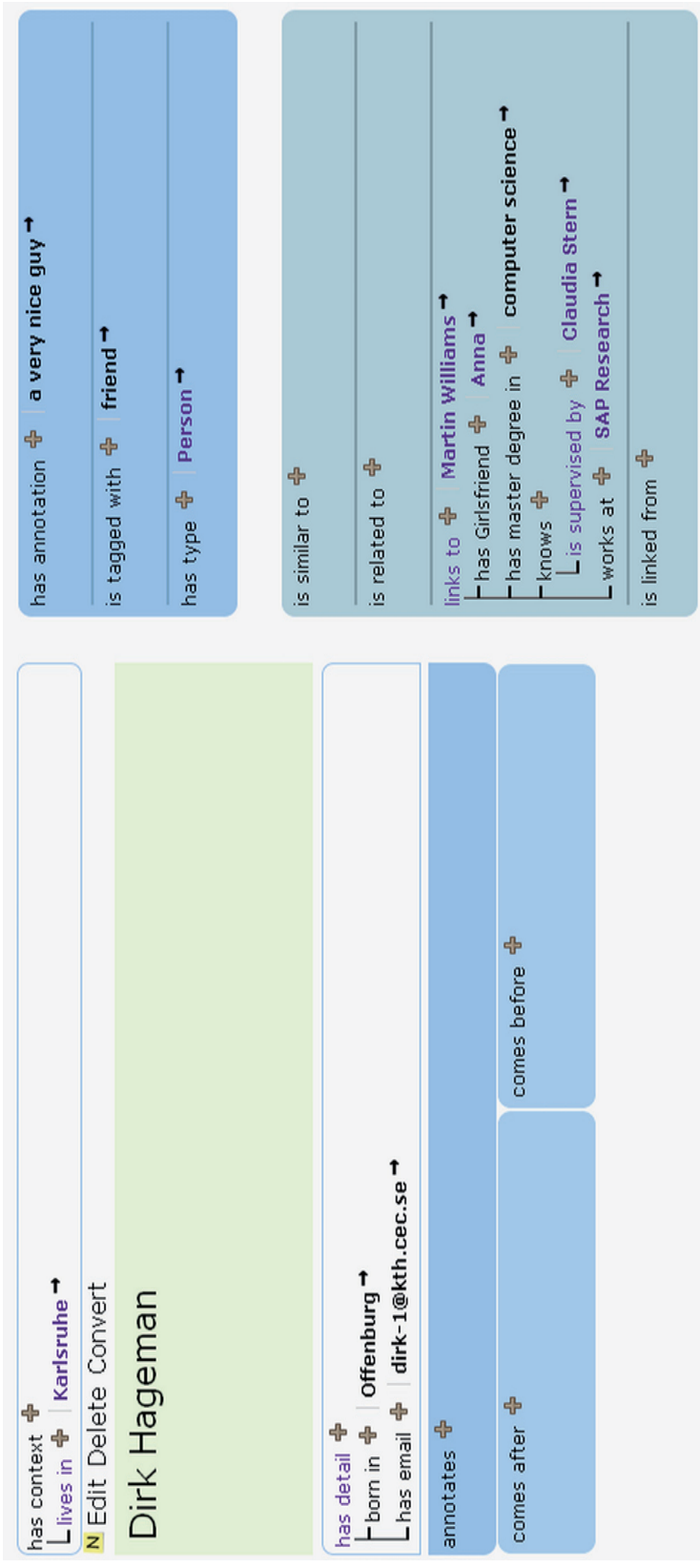


Figure 5.22.: HKW prototype screen shot, focusing on *Dirk Hageman*

Programming JavaScript, a dynamically typed script language, is not easy. For dynamically typed languages, there are no IDEs with, e. g., refactoring support. The different APIs of different browsers make the problem even harder. The approach of GWT is to write the code in Java, a statically typed programming language, for which many mature IDEs exist. GWT includes a compiler, which translates the Java code into compact JavaScript code. Additionally, GWT includes a widget library, similar to desktop GUI frameworks such as Swing or SWT (from the Eclipse project). Programming with GWT relieves the developer from dealing with browser differences and JavaScript. The GWT toolkit runs on all major browsers, i. e., Firefox, Safari and Internet Explorer. Firefox is a popular, stable browser available for free on all platforms (Windows, Linux, Mac). Therefore the choice of GWT enables end-users on all platforms to use HKW.

Client-Server synchronisation The main concern for usable performance in the browser is the minimisation of HTTP traffic to the server. Therefore, a client-side caching system has been implemented. In fact, the JavaScript code in the browser is an almost complete implementation of the CDS data model. Communication with the server is based on the change-date of Items. With each request, the browser sends the highest received Item changed date. The server responds with all Items that have been changed since that time-point. Note that this algorithm does not depend on synchronisation of clocks, instead, the server could use any kind of ordered values.

Caching The main view in HKW shows a central Item an all Items "around it". Technically, the screen shows an Item c and all Statements and Triples $(c, *, *)$ and $(*, *, c)$. The index in the JavaScript client is optimized for this usage. The cache is mainly a mapping from *ItemURI* strings $\rightarrow$ a pair consisting of an Item and a cache entry.

AbstractItemData objects. These contain all the data and meta-data of Items, as well as a link to an *ItemInfo* object. An *ItemInfo* is a local cache with a nested map *RelationURI* $\rightarrow$ (*TargetURI* $\rightarrow$ *TargetInfo*). A *TargetInfo* contains a Boolean flag to encode if the triple (*ItemURI*, *RelationURI*, *TargetURI*) has been stated. In addition to that, and unlike, e. g., RDF stores, the *TargetInfo* has a set of Statements. In CDS, several Statements can encode the same Triple. For fast access, each Statement $s = (source, relation, target)$ is indexed twice in the index: Once as $(sourceURI \rightarrow (relationURI \rightarrow (targetURI \rightarrow \{s\})))$ and again in its "twisted" form as $(targetURI \rightarrow (inverseRelationURI \rightarrow (sourceURI \rightarrow \{s\})))$. Table 5.2 shows how the eight possible triple patterns are mapped to the index in JavaScript.

Query	Lookup pattern	O-complexity
(s, r, t)	(s, r, t)	$O(1)$
$(s, r, *)$	$(s, r, *)$	$O(1)$
$(s, *, *)$	$(s, *, *)$	$O(1)$
$(*, *, *)$	$(*, *, *)$	$O(1)$
$(*, r, t)$	$(t, inverse(r), *)$	$O(1)$
$(*, r, *)$	Exhaustive search over $(*, *, *)$	$O(n), n =$ index size
$(s, *, t)$	Exhaustive search over $(s, *, *)$	$O(k), k =$ result size of$(s, *, *)$
$(*, *, t)$	$(t, *, *)$	$O(1)$

Table 5.2.: Mapping query patterns to index lookup patterns

Inference The formal semantics of the CDS data model (cf. 4.1.2) is implement in JavaScript in a procedural way. This allows the browser to calculate the inferred triples without contacting the server. In fact, the server is only needed for wiki syntax conversions and for persistent storage.

The inference strategy is *materialisation*, which is forward-chaining calculation of all triples that can be inferred from the given STATEMENTS and storing them in an index. This strategy allows answering queries in $O(1)$ but requires more memory than, e. g., backward-chaining, which takes a query and checks backwards which triples fulfill the conditions – this takes usually more steps than a simple index lookup.

For each new triple, a number of inference steps are calculated. Note that each triple is indexed twice, once as (s, p, o) and once as $(o, -p, s)$. This reduces the number of checks to be performed. The incremental, forward-chaining, recursive algorithm is depicted in Fig. 5.23 in pseudo-code. The algorithm can handle incremental adding of triples. On deletion of triples, a complete re-inferencing is required. Note that this is a state-of-the-art technique which is used also, e. g., in the Sesame OpenRDF RDFS inference engine (cf. Sec. 2.6). Semantics of [has alias], [is same as] and [replaces] is not implemented.[8]

Summary The HKW implementation consists of 329 Java files with a total of 28.226 lines of code (excluding whitespace lines, including comments). Of these files, 196 files are compiled to JavaScript, totalling to 17.889 lines. In JavaScript, this results in 22.228 lines (compiler option: PRETTY) which is 808 KB, and 296 KB with default compiler options (OBFUSCATED). All CDS operations are possible in HKW (cf. Tab. 5.3).

[8] The memory consumption and runtime of the JavaScript-based engine is already impacting the user interface reactivity in many browsers. Implementation sketch: An efficient treatment of [has alias] and [is same as] requires a normalisation of the whole knowledge model before running the materialisation. The normalisation step needs to replace all a and b with (a, [is same as], b) with a canonical symbol. At query time, the query needs to run through the same normalisation step.

Let M be the model of true triples.

function inference(s,p,o)

step 0: initialise
Let I be a temporary set of triples being inferred.
$I = \emptyset$

step 1: infer
Inverse triples
$\Rightarrow (o, -p, s) \in I$

Propagate sub-relations
$\forall(p, [\text{is subtype of}], x) \in M \Rightarrow (s, x, o) \in I$

Transitivity of [is subtype of]
IF ($p = [\text{is subtype of}]$) THEN:

$\forall(o, [\text{is subtype of}], x) \in M \Rightarrow (s, [\text{is subtype of}], x) \in I$
$\forall(x, [\text{is subtype of}], s) \in M \Rightarrow (x, [\text{is subtype of}], o) \in I$

Propagate sub-relations
IF isRelation(s) AND isRelation(o) THEN
$\forall(x, s, y) \in M \Rightarrow (x, o, y) \in I$

// Apply formal types
$\forall(x, [\text{has type}], s) \in M \Rightarrow (x, [\text{has type}], o) \in I$

ELSE IF ($p = [\text{has type}]$) THEN:
$\forall(o, [\text{is subtype of}], x) \in M \Rightarrow (s, [\text{has type}], x) \in I$

step 2: index and recurse
$\forall(s, p, o) \in I \setminus M$:
Index
Add triple (s, p, o) to M

Recurse
$inference(s, p, o)$

The algorithm stops when no new triples are inferred within one iteration, i. e., $I = \emptyset$. Since the number of triples that can be inferred is bound (no rule introduces new Items, the maximal number of triples with n Items is n^3), the algorithm always stops.

Figure 5.23.: Procedural CDS inference in HKW (JavaScript)

Item	Operation	Explained in Sec. 5.2, step
Any Item	Delete	6
NameItem	Create	1, 2
Relation	Create	5
Statement	Create	2, 5
ContentItem	Create	6
Relation	Rename	6
Statement	Edit	3

Table 5.3.: All CDS change operations are possible in HKW

5.3. Further CDS-based tools

Besides HKW, there are two CDS-based tools that have been realised by third parties, but in close collaboration with the author of this thesis. This section presents these tools briefly in order to demonstrate features of CDS. Both rely on CDS as their data model and the CDS-RI as their back-end.

5.3.1. iMapping – a zooming user interface tool

iMapping[9] (Haller, Abecker, and Völkel, 2010; Haller, 2006; Haller, Völkel, and Kugel, 2006) is a technique for visually structuring information objects. It supports the full range from informal note taking over semi-structured personal information management to formal knowledge models. With iMaps, users can easily go from overview to fine-grained structures while browsing, editing or refining the knowledge base in one comprehensive view. An iMap is comparable to a large white-board where information Items can be positioned like post-its but also nested into each other. Nesting of iMap-items is represented as CDS Items linked via [has detail] Statements. Spatial browsing and zooming as well as graphical editing facilities make it easy to structure content in an intuitive way. iMapping builds on a zooming user interface approach to facilitate navigation and to help users maintain an overview in the knowledge space. A small sample map is depicted in Fig. 5.24.

Spatial grouping of items

Using iMapping To group items, Linda could, e. g., simply move the Item "New Zealand" close to "Australia", because the two countries a geographically close to each other. This would not introduce any kind of statement in the underlying CDS model but, help her to remember associations with minimal modelling effort.

Spatial and associative re-finding

To re-find things, e. g., "From what year was that Spielberg movie about white sharks?", Linda would first zoom out, then zoom in the upper right corner for "private stuff" (not in the Figure). In there, she would go to her "Movies" Item and browse in. As there could be many movies, she first goes into the "Best Directors Ever" Item inside the "Movies" Item. In there, she selects "Stephen Spielberg" and can see all outgoing links,

[9] `http://imapping.info` (accessed 05.05.2010)

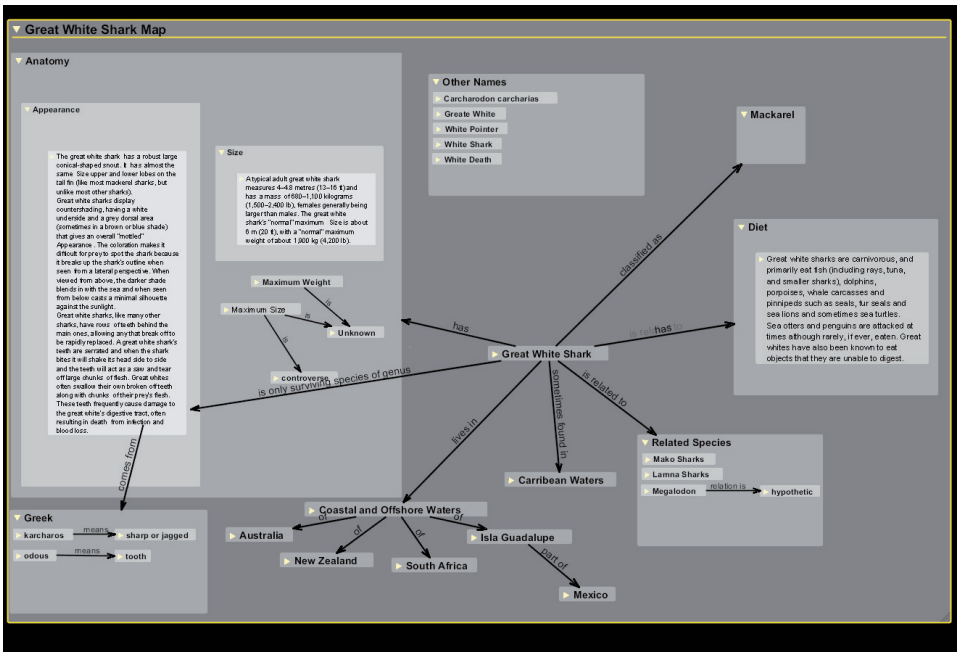


Figure 5.24.: An sample iMap from the user evaluation on the shark scenario

labelled with the type. Although there are quite some links, she just looks at links pointing at movies and quickly identifies the "Jaws" movie. After zooming onto it, she sees "1975" inside the "Jaws" ITEM. On hovering over it, a relation "release date" is shown between the outer "Jaws" ITEM and the inner "1975" ITEM. iMapping allows finding an ITEM based on spatial proximity simply by moving around in the infinite 2D space.

Every item is a container

To name containers, Linda can create an ITEM "Related Species" and put inside ITEMS named "Mako Sharks", "Lamna Sharks", and "Megalodon". She can simply click inside an existing ITEM and thereby gets a cursor to enter the text of a newly created child ITEM. She could also create first the three related species ITEMS and then add a new ITEM and drag the related species ITEMS into it.

Linking by drawing lines

Users can drag a line from one ITEM to another one to create a typed link. The type of the link can optionally be entered into an auto-complete widget that appears directly after the drag-drop.

Multiple levels of detail

iMapping allows seeing infinitely many levels at once, only limited by screen resolution.

Easy re-structuring

In iMapping, the user gets a graphical, zoom-able overview of all his ITEMS and can simply structure his ITEMS by drag-and-drop like on a physical pin-board. After grouping related ITEMS together and moving them inside another ITEM, a number of ITEMS can efficiently be manipulated at once. In this regard, iMapping has the same re-structuring capabilities like mind-mapping tools, but with the added value of spatial hypertext, i. e., the positions of ITEMS are chosen by the user which allows creating very lightweight "piles" of related ITEMS, just like on a physical desk.

Summary iMapping is an example of a concept that builds on the CDS ideas. To implement the spatial hypertext feature, additional data has to be managed. Internally, iMapping uses two stores, one for the 2D positions and a CDS-RI for the CDS data. The 2D positions are stored in an RDF model with RDFReactor.

5.3.2. QuiKey – a graphical command-line

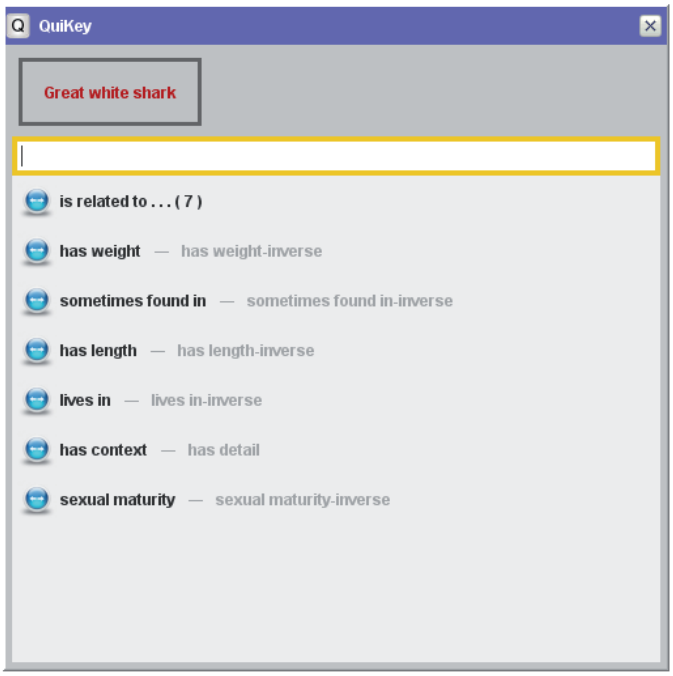


Figure 5.25.: QuiKey screen shot

QuiKey (Haller, 2008) is a kind of smart semantic command-line that focuses on high interaction-efficiency to browse, query and author CDS-based knowledge bases in a step-by-step manner. It combines ideas of simple interaction techniques like auto-completion, command interpreters and faceted browsing and integrates them to a new interaction concept. QuiKey forms a generic, extensible user interface for CDS models. Despite its versatility, QuiKey needs very little screen space, which also makes it a candidate for future mobile use. A screen-shot of its implementation (as of October 2009) is depicted in Fig. 5.25.

Fast entry of new items and extension of existing items – Less optimal for re-structuring

QuiKey is the fastest tool (out of HKW, iMapping and QuiKey) by means of mouse clicks and keys typed for entering data. QuiKey is not well suited for re-structuring existing knowledge.

With one short-cut the tool is brought into focus. Now Linda can simply enter a new short note such as “white shark: sexual maturity occurring at about 12-15 years of age”. Alternatively she can write “Great white shark” <tab>, “sexual maturity” <tab> “12-15 years”<return>. This will add a CDS statement to her knowledge model without requiring here

to navigate anywhere first and still extend the existing white shark ITEM. New ITEMS and relations are created on the fly if needed. Re-use of existing ITEMS and relations is encouraged with auto-completion. After she wrote "Great white shark" <tab> QuiKey already presented here a ranked list of existing statements about the great white shark. Thus QuiKey also includes browsing of the knowledge base.

Advanced search and query

QuiKey has advanced query capabilities, e. g., Linda could ask in QuiKey

$$\Big(([\text{Great white shark}], [\text{has sense}], ?x)\text{AND}(\text{NOT}\,([\text{human}], [\text{has sense}], ?x))\Big)$$

to find out that white sharks have a sense for electrical fields that humans don't. In systems that do not allow formalising content she would potentially have to read many articles and build up the query results in her mind. Semantic queries are especially useful for creating list of ITEMS fulfilling certain criteria. More details about the user interaction for creating such queries are given by Haller (2008).

Summary QuiKey uses a subset of CDS (no annotation of STATEMENTS) and provides a radically different user experience than HKW or iMapping. Yet it builds on the same data model. This shows the flexibility of the CDS model. QuiKey is described in more detail in (NEPOMUK Consortium et al., 2008).

5.4. Conclusions

This chapter presented the CDS API, its reference implementation and three applications built upon it.

The CDS API and the reference implementation have been refined several times. The final version has been used with minor changes and bug fixes for over ten months and can be considered stable.

The tool *Hypertext Knowledge Workbench* (HKW) allows end-users to create and use CDS knowledge models. The tool provides a number of usability features to allow for efficient authoring. The tool runs in a browser to be usable in a number of usage settings (local desktop, internet café, hosting in the cloud, ...). A walk-through demonstrated how the different CDS operations can be performed in HKW. All CDS operations are possible in HKW. Table 5.3 lists the operations and gives a summary where which operation has been explained.

Two other tools based on CDS show how very different user interfaces can be realised on top of the flexible CDS model. Table 5.4 compares the three CDS tools.

The three tools can work on the same CDS model at run-time. This allows propagating changes made in one tool to other tools instantly via change events.

Item	Operation	HKW	iMapping	QuiKey
ContentItem	Create	+	+	+
ContentItem	Edit	+	+	+
ContentItem	Delete	+	+	+
ContentItem	Navigate hyperlink	+	–	–
NameItem	Create	+	+	+
NameItem	Edit	+	+	+
NameItem	Delete	+	+	+
Relation	Create	+	+	+
Relation	Edit	+	–	+
Relation	Delete	+	–	+
Relation	Navigate to inverse	+	–	+
Statement	Create	+	+[a]	+[b]
Statement	Edit	+	+	–
Statement	Delete	+	+	+
Statement	Navigate to source	+	+	–
Statement	Navigate to relation	+	–	–
Statement	Navigate to target	+	+	–
Any Item	Show creation date and author	+	–	–
All Items	Queries	–[c]	–	+

[a] Statements can only use ContentItems and NameItems as source and target.
[b] Statements can only use ContentItems, NameItems, and Relations as source and target.
[c] QuiKey includes an interactive query-builder. This functionality would be expensive to port to the web-based user interface. Since all three CDS tools work on the same model, users can simply create queries in QuiKey to query, e. g., a knowledge model created in HKW.

Table 5.4.: Comparing CDS operations of HKW, iMapping, and QuiKey

6. Evaluation and Related Work

This chapter presents several kinds of evaluation of the contributions as well as pointers to related work.

CDS: A modelling language

The main result of this thesis is the CDS model with its three elements: data-model ($\mathfrak{D}$), relation ontology ($\mathfrak{R}$) and structured text model (STIF). CDS is a conceptual and technical model, designed for usage in PKM tools. Ultimately, CDS is a *modelling language* (or meta-model, cf. Sec. 2.1) for personal knowledge models.

Evaluating modelling languages Evaluating a conceptual model, designed to help people to manage their personal knowledge cues[1] better, is not straightforward.

Imagine you invented UML (cf. Sec. 2.5) and now you would like to find out if UML is "good". Or "better than X". This obviously depends on the tasks that UML was designed for. UML was designed for communication among developers and as a graphical modelling language. How would you evaluate that the features of UML are required? You would create list of potential use cases, derive requirements from them and from known literature, and create a model that fulfills them. Furthermore, you would invite developers to use UML for communication and for modelling software systems. If the developers confirmed that the modelled software system had similar characteristics to a system modelled with another state-of-the art modelling language, you would accept UML.

Frank (2000) notes that unlike scientific theories, modelling languages cannot be falsified. He also notes that in simplistic evaluations the biggest and most complex meta-models can claim the most features – although they might be very hard to use. Usage of models and modelling languages depends not only on its features but also heavily on the markets in which they are applicable. Many standard evaluation procedures overlook the user of a model and the usage context of it. Frank (2000) concludes that an objective evaluation of modelling – and even more so for modelling languages – is not possible. It is suggested to evaluate models by rational discourse.

Evaluating PKM systems An indirect method for evaluating the usefulness of a model for PKM is to study it embedded in a PKM tool. Lethbridge (1998) argues that software systems can be evaluated by measuring how they perform on the tasks they have been designed for. The tasks in which a knowledge base or ontology is going to be used are often not known beforehand. E. g., people often record ideas for "later use" without knowing if and when that knowledge might be of interest again. Ontologies and

[1] Knowledge cues are introduced in Sec. 1.2.3

knowledge bases are also used to uncover latent knowledge via inference rules. In these cases the task is to "derive new, unexpected knowledge". As the new knowledge is unexpected, it is hard to evaluate its value in a systematic manner. In a way, a knowledge base is designed to be used for future, unknown tasks. On the other hand, the author of a knowledge base must have at least a vague idea for the situations she will encounter in order to benefit from her knowledge cues.

There are some publications on evaluating organizational knowledge management systems (for instance Nevo, Furneaux, and Wand, 2008; Folkens and Spiliopoulou, 2004), but they do not apply to personal knowledge management systems.

Jones and Bruce (2005) report on difficulties of evaluating PIM systems, as almost no person is willing to try out an experimental, potentially unstable PIM system for a long period of time (months or years) for critical personal tasks such as remembering appointments, managing email or keeping personal notes. The same applies to PKM systems.

Worst of all, flaws in the user interface are hard to distinguish from flaws in the underlying model. Nevertheless, there is no other way to study some of the model properties besides actually using the model in practice.

Five evaluation approaches Despite all difficulties in evaluating CDS, it has been evaluated in five different ways.

Rational Discourse First, the results of this work, CDS and the tools built on top, are evaluated theoretically in a rational discourse with respect to the requirements motivated and discussed in Chapter 3. This comparison can be found in Sec. 6.1.

Expert Evaluation (December 2007) An early HKW prototype from December 2007 has been analysed in an expert evaluation by KTH[2], a partner in the NEPOMUK project. The expert evaluation method and its results are described in (NEPOMUK Consortium et al., 2008)[p. 31–36] and not repeated here. It resulted in a heavy reworking of the HKW user interface.

Formative User Study (October 2008) Some required features cannot be evaluated by rational discourse. Especially the feature "Easy to learn" (14 $^{\text{easy to learn}}$) requires a user study. Two empirical user studies have been performed. The first user study involved 16 participants from 4 different software companies. Each participant spent about 1 hour with the evaluation of HKW. This user study is described in Sec. 6.2.

Comparative User Study (July – October 2009) The second user study, based on a more mature prototype of HKW as well as on other CDS tools, involved 5 participants. Each participant spent about 27 hours with the evaluation. This study focused on evaluating the requirements "Simultaneous use of Different Levels of Formality" (8 $^{\text{formality levels}}$)

[2]Kungliga Tekniska Högskolan (Royal Technical University

and "Stepwise Formalisation" (9 $^{\text{stepwise}}$) This user study is described in Sec. 6.3.

Data Model Comparison The resulting CDS data model $\mathfrak{D}$ is compared with the RDF model.

6.1. Fulfilment of requirements

The evaluation of requirements as fulfilled or not is presented in Table 6.1 on page 204. Some requirements relate either to the knowledge model formalisms (model), other only to tools (tool), many to both. E. g., requiring the data-model to be a super-set of existing PKM data models cannot be fulfilled by a tool. Similarly, the requirement "fast entry" cannot be fulfilled by a data-model. In these cases, "n/a" is used for "not applicable". As the model is split into three parts, it is for most features enough to be covered by at least one part of the model. E. g., requirement 5 (formal articulation) *cannot* be covered by the data-model part of CDS ($\mathfrak{D}$), as it has been designed to be generic with respect to formal semantics. Formal semantics can only be achieved with the RELATION ontology ($\mathfrak{R}$). It is a feature of the CDS model, to split responsibility for fulfilling requirements among parts. Table 6.1 gives also an indication of the usefulness of using a part of the CDS model separately. Detailed explanations for all ratings can be found in the appendix A.5.1.

Requirement 1 $^{\text{auto-query}}$ and 13 $^{\text{maintenance}}$ are considered important but have not been implemented due to lack of implementation resources.

Requirement 1 $^{\text{auto-query}}$ requires a kind of user notification system which does not exist in the CDS API as described in the last chapter, however, it should be straightforward to implement. For each query the current result needs to be stored, queries need to run periodically and the new results need to be compared with the stored results. If a difference is detected, the user should be informed. A more elaborate implementation could observe the content of newly created or changed ITEMS and try to find ITEMS based on text similarity.

Requirement 13 $^{\text{maintenance}}$ can be implemented by asking the user periodically about the value of old ITEMS that have not been used recently. This requires to track the last access time of ITEMS via browsing or query results. However, to not overload the user, a smart trade-off between maintenance benefits and cost of answering automatic maintenance requests is required.

Overall, almost all requirements have been fulfilled to a high degree.

6.2. Formative user study

This section describes a formative user study with 16 participants from 4 companies, based on an early prototype of HKW. The goal of the study was to assess the learnability of the CDS concepts and to get early feedback on HKW's design.

Req. No.	Description	*Model*			*Tool*		
		CDS $\mathfrak{D}$	CDS $\mathfrak{R}$	STIF	HKW	iMapping	QuiKey
1	System should run queries automatically	n/a			–	–	–
2	Super-set of PKM data models	+	+	+	n/a		
3	Fast entry	n/a			+	+	+
4	Informal articulation	+	+	+	+	+	+
5	Formal articulation	n/a	+	+	+	∼	+
6	User decides on modelling granularity	+	n/a	+	+	+	+
7	Entities need to be addressable	+	+	n/a	+	+	∼
8	Simultaneous use of different levels of formality	+	+	+	+	+	∼
9	Stepwise formalisation	∼	+	+	+	+	–
10	Knowledge model refactoring	n/a			∼	+	∼
11	Versioning	–	n/a	n/a	–	–	–
12	Capture the context for cue creation and import	+	n/a	n/a	+	–	–
13	Active assistance in maintenance tasks	n/a			–	–	–
14	Easy to learn[1]	+			∼	+	∼
15	Grouping of items	+	+	n/a	+	+	+
16	Containment relationship	n/a	+	+	–	∼	–
17	Optional naming of knowledge cues	+	n/a	n/a	+	+	–
18	Alternative names	n/a	+	n/a	∼	∼	∼
19	Order knowledge cues	n/a	+	+	∼	–	–
20	Linking	+	+	+	+	+	+
21	Hierarchy	n/a	+	+	+	+	+
22	Sim. use of multiple levels of detail	n/a				+	–
23	Annotating content	n/a	+	∼	+	+	+
24	Tagging	n/a	+	∼	+	+	+
25	Classifying items into categories	n/a	∼	–	+	+	+
26	Queries	+	+	n/a	–	–	+
27	Browsing	+	+	+	+	+	+
28	Inverse Relations	+	n/a	–	+	?	?
29	Flexible schema	+	+	n/a	+	+	+
30	Transclusion	+	n/a	–	–	+	-
31	Meta-modelling	+	+	n/a	+	–	–

Legend: feature present ('+'), not present ('–'), partially present ('∼'), not applicable ('n/a')

[1] See Sec. 6.3, hypothesis 3

Table 6.1.: Fulfilment of requirements

6.2.1. Method

An early prototype of the Hypertext Knowledge Workbench (HKW, see Sec. 5.2) has been evaluated in a formative user study (Scriven, 1991) conducted at four different software companies. At this point in time, iMapping and QuiKey have not been ready for evaluation yet.

Participants The participants of the study have been recruited from four different software companies in the context of the WAVES[3] project. From each company, 4 people were selected for a one-hour structured interview. In total, 16 participants with an age of 30 to 50 years (average age: 38 years) have been studied. The study was conducted in September and October 2008. People had different backgrounds ranging from secretary over server administrator and programmer to team lead and financial administration. All were working within software development projects.

Tools The study has been conducted with a HKW prototype version built on 10th May, 2008, version 3388.

Data and Tasks Following the suggestions of Bernstein, Kleek, monica mc schraefel, and Karger (2008), the tool was seeded with existing notes. The tool was filled with imaginary personal notes of the fictional biology researcher Linda, introduced in Sec. 5.2.1. The imaginary notes are based on Wikipedia content[4].

Two simple tasks, a retrieval and an authoring task, have been designed to force participants to become familiar with the tool. The first task was to find out in which year a movie about the white shark appeared and who was the director. Several solutions were possible. E. g., navigating to "movie" and browsing the *instances* of it. In the second task participants were asked to record the fact that the phone number of a person called "Claudia Stern" is "9654-854".

Procedure First the topic of PKM, the CDS ideas and the HKW tool were presented to the whole group. Then each participant was introduced to the HKW prototype and given two tasks to get acquainted with the tool. The order of tasks (retrieval, creation) was held constant for all participants, because it is more logically to first learn how to read an existing knowledge model before adding something to the structure.

People played with the tool freely until they considered themselves confident to have understood the idea. Then a structured interview was carried out.

[3] WAVES: Wissensaustausch bei der verteilten Entwicklung von Software, sponsored by BMBF, Germany. Website: `http://waves.fzi.de` (accessed 05.01.2010)

[4] The data was taken from `http://en.wikipedia.org/wiki/Great_white_shark` (accessed 06.01.2010) and `http://en.wikipedia.org/wiki/Basking_shark` (accessed 06.01.2010)

Measurements Participants were interviewed with a structured interview with the following questions:

1. Are the main building blocks (NameItem, ContentItem, Relation and Statement) clear and understandable? [5]
2. Is the Relation ontology easy to understand and easy to learn?
3. For what kind of knowledge or tasks can you imagine to use HKW?

6.2.2. Results

On average, participants spent 04:36 minutes (SD[6]=02:12 min) on the retrieval task in this first study. On the creation task, participants spent on average 02:18 minutes (SD=01:33 min). That this number is lower than the one for retrieval might mean that articulation in HKW is easier than retrieval, but it is more likely that people simply got adapted to the tool – which was the whole purpose of the two tasks. On average, people played then around freely with the tool for 06:54 minutes (SD=02:50 minutes).

*Question 1: Are the main building blocks (*NameItem*,* ContentItem*,* Relation *and* Statement*) clear and understandable?* 69% of the participants found them comprehensible and easy. Three participants voiced concerns about the representation of the building blocks in HKW, about the required effort to use these building blocks, and one participant complained that post-its are a bad metaphor to explain these building blocks. 19% found the difference between ContentItem and NameItem difficult to understand and suggested to merge these two concepts. 12% of the participants found the building blocks unusual, especially the distinction between Relation and Statement was not clear to them. One participant noted being more used to wiki-concepts.

Question 2: Is the Relation *ontology easy to understand and easy to learn?* Here responses were much more mixed and more negative. 19 % found the relation ontology easy to understand and of the right size. 69 % found the Relation ontology too complex. Several participants suggested an introduction mode which hides some of the less-often used relations. Many participants suggested to remove some relations, however, they all suggested different relations to remove ([is same as] (2 times), [comes before] and [comes after] (3 times), [is alias for]). Some relation names seemed to denote the same. Again, each participant found different relation name pairs problematic ([has context] and [has annotation], [links to] and [has annotation], [has type] and [has annotation]). 12% of participants found the Relation ontology too small and suggested to add, e. g., [depends on] and more relation types under [is similar to].

Question 3: For what kind of knowledge or tasks can you imagine to use HKW? Participants listed a number of use cases, both in their private and work life. Two participants said they could not imagine to use the tool privately, because it is too slow. Mentioned private tasks were:

[5] Triples were not part of the datamodel yet, at the time of the survey.
[6] standard deviation

- cooking recipes;
- DVD collection,
- places I visited;
- family management: e. g., birthday wishes of my kids.

Mentioned task for work life were:

- Project overview, e. g., who does what, would use it together with several companies;
- requirements engineering;
- preparation of seminars;
- instead of post-its;
- new legal procedures I get via an email newsletter;
- problem solving methods, collect ideas and sort them, workshops with clients;
- notes taken while on the phone;
- who works on what topic, holiday regulations;
- contact information, appointments;
- project knowledge, terminology;
- code snippets, software serial numbers, tutorials;
- how-tos, shopping lists, passwords, ideas;
- project planning, meeting minutes;
- scientific work, enterprise infrastructure modelling.

6.2.3. Discussion

The CDS data model ($\mathfrak{D}$) is generally understandable (question 1). The relation ontology ($\mathfrak{R}$) is overwhelming for novice users (question 2). One way to tackle this would be to design CDS user interfaces, so that novice users are not confronted with the full set of relations at once. This strategy has been followed in the later CDS tools.

The fact that all participants suggested different Relations to remove, add or merge, suggests high individual differences in using and understanding Relation names. There is a large variety of knowledge types which people imagine to use in HKW (question 3).

Overall, this brief, initial evaluation confirms the design of the CDS data model.

6.3. Comparative user study

This section reports on a user study comparing the three CDS-based tools, namely HKW, iMapping and QuiKey with a state-of-the-art semantic wiki. This study is designed to assess the CDS model.

A number of note-taking studies have been presented in Sec. 2.7. As a conclusion from these studies, the CDS study (1) was designed to not rely only on a small number of notes; and (2) have at least 7 days between modelling and recall tests.

6.3.1. Method

Jones and Teevan (2007, Ch. 11) proposes four main methods for evaluating PIM tools: naturalistic (as opposed to laboratory studies), longitudinal (i. e., running over a long time span), case study, and laboratory. The study presented in this section is a laboratory study. The goal of the study is to assess the CDS model ($\mathfrak{D}$ and $\mathfrak{R}$, presented in Chapter 4) by observing the behaviour of participants using tools based on this model. Usability and completeness of the tool influence the evaluation data heavily, but a more direct assessment of the CDS model is not possible. This study compares tools based on the CDS model with a state-of-the art semantic wiki.

The overall procedure of the study consisted of these steps:

Authoring 5 participants create knowledge models using 4 different tools, based on 3 different scenarios.

Break All participants have a 4 week long break so that they forget the facts from the scenarios.

Retrieval Each participant is using his knowledge models to answer a set of questions about the scenarios.

A much more detailed description of the study procedure is given below, after participants, tools and the scenarios have been described. Following the measurements, derived metrics and results are presented.

Participants Five lab students have been recruited for the evaluation (all male, one graduate and four undergraduate computer science and math students, 21–27 years old). All were paid their usual salary to perform the evaluation tasks. Two of the participants were students working on developing QuiKey and iMapping. The other three had never used the CDS API nor any CDS tool before. None had used Semantic MediaWiki (SMW, introduced in Sec. 2.9) before.

Previously used tools for taking personal notes were: sticky notes (4×), simple text editor (3×), paper notebook, Google documents, KDE4 Plasma Notes Widget, Word, Excel, Powerpoint, cell phone, index cards, wiki (MediaWiki), and mind-maps (freemind).

Two groups of participants to avoid sequence effects

The participants are divided into two groups. Group I consists of participant p_1, p_2, and p_4. Group II consists of participants p_3 and p_5. Each group contained one student working on developing QuiKey (p_4) or iMapping (p_5). Participant p_2 received each task one week before the others. His feedback was used to improve the exact wording of the task and remove usability issues from CDS Tools.

Tools To compensate for the tool influence, *three* tools based on CDS (HKW, QuiKey, iMapping) have been packaged into one installable unit, called *CDS Tools* and deployed at a small number of users. A *CDS Tools* release was created on 13th July 2009 and made available via Java Webstart [7]. Two updated versions mostly with changes to stability and the evaluation user tracking system have been published on 22nd July and 4th of August.

It was decided to compare *CDS Tools* with another powerful modelling tool, Semantic MediaWiki (SMW), which is described in Sec. 2.9. SMW[8] together with the HALO extension[9] were chosen because they allow modelling similar structures, the tool is stable and freely available. Some members of the AIFB institute[10] at KIT Karlsruhe use SMW for their personal knowledge management. SMW and HALO together are branded as *Semantic MediaWiki+*, abbreviated simply as SMW in this thesis.

During the study, each participant used his own computer for creation of the knowledge bases and for the interview. Each participant got its own SMW installed on a central web server at the institute.

For SMW, rich help pages and email discussion lists are available, for CDS Tools, this was not yet the case.

Data and tasks Elsweiler and Ruthven (2007) describes tasks defined for evaluating PIM systems, especially for re-finding email messages and web sites. These tasks do not fit the PKM scenario of self-created content for one's own consumption (knowledge cues). As explained in Sec. 3.1, note taking is a core use case of PKM. To obtain comparable results with a reasonable amount of resources, only this use-case has been evaluated.

Source data

Three different topics have been selected and used in the evaluation to avoid a topic-based bias. The original German task descriptions can be found entirely in the Appendix A.5.2.

Scenario A: Shark Participants were instructed to *take the role of Linda (cf. Sec. 5.2.1) and create her personal fact collection on the Great White Shark. Add facts on the Basking Shark and also information about books and movies about the Great White Shark.*

[7] `http://java.sun.com/javase/technologies/desktop/javawebstart/overview.html` (accessed 06.01.2010)

[8] Installed versions: MediaWiki 1.15.0, SMW 1.4.2, available at `http://sourceforge.net/projects/semediawiki/files/semediawiki/semediawiki-1.4.2/` (accessed 06.01.2010)

[9] Installed version: 1.4.3-for-SMW-1.4.x available at `http://sourceforge.net/project/showfiles.php?group_id=207513&package_id=248472` (accessed 06.01.2010)

[10] `http://www.aifb.kit.edu` (accessed 05.01.2010)

Each participant had to submit five questions which they could answer given their knowledge base. They had to list another set of five questions they could not answer with the help of their knowledge base. These difficult questions in scenario A turned out to be so difficult that they were discarded for the rest of the study.

Source data: Two Wikipedia pages[11] are used as input data. Participants were advised to use mostly data from these pages. The same two pages have been used in the previous formative evaluation study (cf. Sec. 6.2). Although the sources were explicitly given as two German Wikipedia pages, many participants used the English pages as their source data instead.

Scenario B: Spices This scenario puts participants into the role of a four-star restaurant chef. *The chef was described as having routine in his job and working always under high pressure to create new dishes. The chef uses many spices, but has few knowledge about the spices themselves. Therefore he invests 5 hours for online research on spices. His goal is the creation of a personal knowledge base on spices that is able to answer questions such as: Which spices come from similar plants? Which spices are comparable? Which spices go well together? Which are produced in a similar way? Which are used for similar dishes? And which spices are used in similar cultures? In four weeks there will be a cooking examination.* Participants were advised to investigate popular spices first (to obtain a certain level of comparability between the different knowledge models). This time, the participants had to create a set of five well-answerable questions plus five difficult-but-possible questions.

Source data: No particular source given, participants could use any source on the internet. Most participants used Wikipedia, but not exclusively so.

Scenario C: Social Network In this scenario participants took the role of a manager of a company based in Berlin with 15 employees:

For the tenth anniversary of the company a party needs to be organised. This party is critical for the future of the company, as new projects will be discussed there. Some weeks before the party, the manager meets an employee that tells him a large number of facts about current and former employees as well as customers. To manage who will sit next to whom the manager has to answer as set of questions: Who knows whom and why? Which people are relatives of each other? Who lives together with whom? What kind of relation do the persons have? What professional background do they have? Who worked for whom? Participants were advised to use their own language and read the input text carefully. They should concentrate

[11] `http://de.wikipedia.org/wiki/Wei%C3%9Fer_Hai` (accessed 06.01.2010) and
`http://de.wikipedia.org/wiki/Riesenhai` (accessed 06.01.2010)

on decision-relevant facts. Before modelling, they had to create five questions that would be relevant to them as a manager of that company. This should prime them on encoding more useful knowledge into their personal knowledge bases. Furthermore, participants were instructed to re-use the existing CDS RELATIONS as much as possible.

Again, they had to create additionally five easy-to-answer and five difficult-to-answer questions.

Source data: To avoid any kind of copy and paste, this time the content was given to the participants as eight bitmap image files. They contained the concatenated German text from the first 31 episodes of a popular German TV series ("Gute Zeiten, Schlechte Zeiten"). These episodes have been aired in 1992 and again in 1995. Only one participant (p_1) found out the relation to the TV series.

The text was taken from a fan wiki. The full text can be found in Appendix A.5.2. The text was rated by the participants as "confusing" and "contradicting".

Retrieval Questions All the questions the participants submitted (see *Data and Tasks*) have been collected, together with the participant who proposed the question, the rating (easy or difficult), the scenario for which the questions was designed (A, B, or C) and the tool used for creating the underlying knowledge model (CDS tools or SMW).

In a next step, obvious duplicate questions from these 125[12] questions have been merged manually, while keeping all meta-data. Some questions had now more than one rating, more than one proposing participant, or more than one underlying tool. After merging, there were some conflicting ratings, e. g., some participants rated a questions as easy in one knowledge base and as difficult in their other knowledge base for the same scenario.

To limit the time required for the retrieval interview, a sub-set of 50 questions has been chosen. The resulting set is printed in Appendix A.5.3. The selected sub-set has the following properties:

- It contains 10 questions for scenario A, and 20 questions for scenario B and C, each.
- All eleven questions that have been proposed more than once are included. This enhanced the chance that the question can be answered by all participants.
- Questions have been grouped into 25 pairs of structurally similar questions. To allow this, two questions needed to be created by the evaluator.
- Questions rated *easy* and *difficult* questions have been balanced, with a bias on easy rated questions. The set includes 40 easy ratings and 23 difficult ratings.[13]

[12] 5scenarios × 5easy questions in scenario A + 5easy questions in scenario B + 5difficult questions in scenario B + 5easy questions in scenario C + 5difficult questions in scenario C

[13] As explained above: The participants did not agree on difficulty of questions

Assignment	Scenario	Task	Tool		Time
			Group I	Group II	
–	–	Answer PKM questionnaire (cf. Appendix A.2)	–	–	ca. 0.5h
–	–	Install and get familiar with CDS Tools	CDS Tools	CDS Tools	ca. 2h
1	A	Create knowledge base	CDS Tools	CDS Tools	4-6h
–	–	Tutorial on SMW	SMW	SMW	ca. 2h
2	B	Create knowledge base	CDS Tools	SMW	max. 5h
3	C	Create knowledge base	SMW	CDS Tools	max. 5h
4	C	Create knowledge base	CDS Tools	SMW	max. 5h
5	B	Create knowledge base	SMW	CDS Tools	max. 5h
–	–	Break	–	–	4 weeks
–	all	Retrieval interview: Structured interview and retrieval tasks for all five knowledge bases	both	both	ca. 2h

Table 6.2.: In the comparative user study each participant had to use each tool – but in different order

- It contains roughly the same amount of questions proposed from users working on a CDS knowledge base (28) as well as questions proposed from users working in SMW (23). Additionally, for scenario A the set contains 10 questions proposed by users working on CDS. Note that no user worked on SMW for scenario A.

- Questions have been balanced between participants who proposed them. The set includes questions proposed by participant p_1 (11), p_2 (11), p_3 (12), p_4 (19), and p_5 (10).

Procedure

The modelling tasks ran from 13th July until 4th of September, 2009. The overall procedure for each participant is depicted in Table 6.2. Scenario A had to be performed exclusively with CDS Tools. Scenario B and C were carried out both in CDS Tools and in SMW. As a result, each participant had to model scenarios B and C two times. To motivate them to create useful models, a prize was announced for the participant with the best retrieval results at the retrieval interview.

To avoid effects of sequence, the order of the tools (CDS Tools vs. SMW) was balanced out. To avoid a bias on the order in which the tools were used, the group of participants was split in two parts. The two participants which helped to develop CDS Tools and thus had previous exposure to the tools

were put in different groups. For scenarios B and C, half of the participants first used CDS Tools and then SMW.

Retrieval interview After a four week long break, each participant was interviewed and had to perform a set of retrieval tasks on the models he created before.

The structured interview asked the following questions (in German) and had to be filled out as an online survey. It contained two parts:

Learnability How easy is it to understand and learn the building blocks of the CDS model? Answer in German school grades from 1 (best) to 5 (worst). How easy is it to understand/learn the RELATION ontology? Answer in German school grades.

Evaluating Tools For each tool of HKW, iMapping, QuiKey and SMW, how do you rate the usability? How do you rate the learnability? Answer in German school grades. Complete as free text "I would use this tool for my personal knowledge management if . . . ".

For the retrieval tasks, the participants used the same computers as for creation of the knowledge bases.

Retrieval Task Design To avoid learning and sequence effects on the retrieval tasks, each question pair (a/b) has been asked equally (as far as possible with 5 participants) in the possible four ways (a-CDS, b-SMW), (a-SMW, b-CDS), (b-SMW, a-CDS), and (b-CDS, a-SMW). For each question pair, those participants answering the same question (a or b), form a subset. No question pair was answered with the same partitioning into subsets. This minimizes the effect of questions in one pair not being equally difficult. The resulting schedule assigning questions to participants can be found in Appendix A.5.4 in Table A.3.

The next decision was the order in which the two tools should be used for the questions. It seemed unnaturally often to switch tools after every question. On the other hand, if one participant would go through all questions, then switch tools and got through all opposite questions of each pair, then the order of tools could introduce a bias. Therefore it was decided to switch tools within each scenario (A, B or C) at least once. As a result, each participant had to switch tools four to five times during the complete retrieval interview. The exact schedule for using tools per scenario is depicted in A.5.4 in Table A.2. As scenario A was modelled exclusively in CDS Tools, two participants were instructed to use the original source data to answer the questions. This allows comparing CDS models against encyclopedia articles.

Retrieval task measurements

The final interview was recorded with screen capturing software. Tool changes are clearly observable in the recordings. This allowed to measure precisely the time spent per scenario by each participant for each set of questions per tool.

Dimension	Values	Description	Symbol
Participant	p_1 to p_5	The person who created the model and answered the questions.	P
Scenario	A, B or C	Denoting scenario shark, spices or social network.	S
Order	1 or 2	Indicates if the session was the first or second part of the retrieval task.	O
Tool	*CDS*, *SMW*, or *source data*	The tool used to create the model which the participant re-used to answer the questions. CDS is short for CDS Tools.	T
Questions	5 or 10	The number of questions in a retrieval session.	Q
Modelling time	Minutes	The time the participant spent to create the knowledge model. The creation dates for each ITEM in a CDS model can be measured. By sorting all these time points chronologically, an algorithm can extract consecutive sequences of time points, separated by breaks. Breaks are defined as two time points with a distance of 10 minutes or more. A similar procedure was used for the SMW installations: In Semantic MediaWiki, the *Recent Changes*[14] page of each wiki has been used to gather usage logs. The log contains one time-stamped entry for each page edit. The time between two consecutive time points was summarised, if not greater than 10 minutes.	t_m
Retrieval time	Minutes	The time it took the participant to answer all questions in this retrieval session.	t_r
Points	0 to 30	The points earned per retrieval session.	p

Table 6.3.: Main data-set of comparative user study has 5 independent and 3 dependent variables

Retrieval session measurements Each participant has modelled 5 knowledge models within 5 hours, each. Together, the participants have spent 125 hours modelling and approximately 10 hours in the retrieval interview. Approximately 75 hours have been spent using CDS Tools and 50 hours using SMW.

125 hour data collection

A *retrieval session* is a set of consecutive answered questions by the same participant, using one tool, within one scenario. At retrieval time, there were 5 to 6 sessions per participant:

Retrieval sessions

Scenario A: Three participants did one retrieval session, used only CDS Tools and got 10 questions. Two participants used CDS Tools for 5 questions and the Wikipedia pages (the source data, from which the knowledge models have been created) for another 5 questions.

Scenario B: Two retrieval sessions with 10 questions answers using CDS Tools and 10 questions answered using SMW.

Scenario C: Same as for scenario B.

Table 6.3 shows the structure of the resulting data-set. The actual data for all 27 retrieval sessions (3 participants $\times$ 5 sessions + 2 participants $\times$ 6 sessions) can be found in Table A.5 in Appendix A.5.5. For each question, a participant could earn points. A correct answer was rated maximally three points. One point was achieved for an answer pointing in the right direction. Zero points for completely wrong or missing answers.

Effectiveness and efficiency The retrieval sessions are not directly comparable. Or in other words: It does not tell much, e. g., if people tend to model longer in one tool or earn more points in total simply because there were more points to get in that retrieval session. Therefore, a combined analysis of spent time vs. earned points has been performed. This analysis is based on the concepts presented in Sec. 3.3. Of course, not the complete economic model could be applied, e. g., the value of a point earned for a question is hard to compare with the minutes invested in its creation. Furthermore, the knowledge creation costs (C_C), could be assumed as the time allowed for a modelling session minus the actual modelling time. Some participants modelled even longer than allowed, which would lead to negative knowledge creation costs. The maximal modelling time t_m was 8.17 hours, the mean was 4.44 (SD=2.28[15]). These numbers are based on all 15 CDS models and all 10 SMW models.

Costs are measured as the time required to build a model (Cost of externalisation $C_E = t_m$) and the time required to query the model later (Cost of retrieval $C_R = t_r$). Benefit is measured as the number of points ($B \sim p$) a participant received for his answers. Costs and benefit can easily be compared between tools, scenarios and participants.

Depending on the use-case, knowledge cues are likely to be retrieved more than once. The number of times k this retrieval happens, adds both to costs

[15]Notation: SD stands for standard deviation.

and to benefit. Note that the user study did not measure the retrieval of *all* knowledge cues. A certain amount of luck for each participant in each retrieval session is thus unavoidable. However, this matches well the real-world task of recording knowledge cues for unknown future use.

The total costs can be calculated depending on k as $C(k) = t_m + kt_r$. The total benefit is then the number of points earned per question, $B(k) = k\frac{p}{Q}$, with Q being the total number of questions in a session. This allows comparing the 5-question-sessions with the 10-question-sessions.

The efficiency can than be defined as $e_k = \frac{B(k)}{C(k)} = \frac{kp}{Q\times(t_m+kt_r)}$. To get nicer numbers, time is measured in hours, not minutes, for this metric. The resulting unit is then points per hour. Reasonable values for n are 1, 10, and 100, leading to the metrics

- e_1 Average number of points earned per question as the result of a given modelling effort and one retrieval effort. Times measured in hours.
- e_{10} Average number of points earned per question as the result of a given modelling effort and *ten* retrieval efforts. Times measured in hours.
- e_{100} Average number of points earned per question as the result of a given modelling effort and *hundred* retrieval efforts. Times measured in hours.

For comparison the resulting e_k values have been normalised, i. e.divided by the mean value of the respective e_k metric.

Efficiency For a single session, the mean of e_1, e_{10} and e_{100} is a measure of *efficiency*.

Effectiveness The mean value of points earned for a session (Benefit $B = p$) is a measure of *effectiveness*.

These metrics are used in the results section to compare CDS and SMW.

Measuring knowledge models Lethbridge (1998) proposes a number of metrics for semantic-net based knowledge-bases which can be used to measure features of CDS models. These metrics have been adapted to the CDS terminology, references to Lethbridge (1998) are given in parenthesis. The star symbol signifies that the metric is also defined for SMW.

- M_I The total number of ITEMS in a model (Lethbridge: M_{ALLC}).
- M_N* The total number of NAMEITEMS (Lethbridge: M_{MSUBJ}). This is a measure of size unrelated to complexity.
- M_C The total number of CONTENTITEMS.
- M_R* The total number of RELATIONS.
- M_S* The total number of STATEMENTS.
- M_S0 The number of STATEMENTS in a model that use a RELATION which has only the trivial super-RELATION [has detail].

- M_Sk The number of STATEMENTS in a model that use a RELATION which has exactly k non-trivial super-RELATIONS. The number k can also be interpreted as the formality of the STATEMENT, i. e., it is a lower bound for the number of implied triples[16].
- M_{SFORM} The number of STATEMENTS with a formality greater than one (Lethbridge M_{SFORM}). More formally: $M_{SFORM} = \sum_{k=1}^{N} M_Sk$
- M_T* The total number of TRIPLES that can be inferred (cf. Sec. 4.2.2) from the given STATEMENTS.

Measurements in SMW The pages *Special:Statistics* provides access to the total number of pages, the number of page edits, and the number of page views. The total number of pages can be used as the value of M_N, since each wiki page represents one concept.

Via *Special:Semantic statistics* the number of distinct properties as well as the number of property values (i. e., number of semantic statements) could be gathered. The number of properties corresponds to M_R, but one must keep in mind that CDS automatically creates an inverse RELATION, which is not the case for SMW. For each used property, the number of times the property has been used was extracted from the *Special:Properties* page. This corresponds to M_S.

6.3.2. Results

First a brief summary of the resulting data-set is given. Then the validity of the data-set is checked with respect to assertions of the evaluation method. Finally a number of hypotheses are stated and assessed.

Data set summary The final data set used for the evaluation consists of 15 CDS models created by 5 different participants in 3 scenarios.

There are 27 sessions with a different number of questions, of which 15 have been carried out with CDS Tools, 10 with SMW, and 2 with the source data. Table A.5 in Appendix A.5.5 shows the data gathered in the comparative user study (Sec. 6.3). It shows for each tuple of participant (P), scenario (S), order of tools (O), tool (T), and number of questions used in the session (Q), the modelling time in minutes used to construct the knowledge model (t_m), the time it took the participant to answer the questions (t_r), and the number of points he received for the answers (p).

No correlations

As the possible maximal number of points is proportional to the number of questions asked, the average points per question (ppQ) are used for correlation tests. None of the pairwise correlations (ppQ, t_m), (ppQ, t_r), and (t_m, t_r) is significant.

[16] More triples can be implied by more complex constructions such as [has type] and [has subtype].

Scenarios B and C only *The next analyses are restricted to scenario B and C to be able to compare CDS Tools with SMW.* Participants earned significantly more points in scenario C than in scenario B, measured as the number of points earned per question ($p = 0.000056$). Each participant used both tools, but in different order. Paired t-tests reveal that the order of tools does not significantly influence the outcome of ppQ (p=0.6067).

Furthermore, paired two-tailed t-tests between the group CDS Tools and the group SMW show that

- Participants' modelling time in SMW is significantly shorter than time spent in CDS ($p = 0.0025$). Note that only the time spent interacting with the tool is measured. Students were advised to spend 5 hours per task. The rest of the time *should* have been spent on research and thinking. In fact, participants modelled on average 50 minutes longer with CDS Tools than in SMW for the same scenario (p=0.0436).
- Retrieval time does not differ significantly.
- The resulting points per question (ppQ) do not differ significantly between tools.

Validity of data collection As a first step, the data set is checked for validity.

Developers perform much better?

Hypothesis: Participants that helped to develop iMapping and QuiKey perform much better. If this hypothesis is true, both these participants (p_4 and p_5) must have more points than the other participants. This hypothesis is false, as p_4 has the most points (93), but p_5 has the lowest number of points (67).

Furthermore, both should be able to model more Items per minute than the other ones, measured as the sum over all models created by each participant. This hypothesis is false, as the predicted order of participants $(p_4, p_5) > (p_1, p_2, p_3)$ does not match the observed order $p_3 > p_4 > p_2 > p_1 > p_5$[17]. Therefore participants p_4 and p_5 are not handled specially in further analyses.

In the next steps, the data set is used to evaluate the research questions presented in Sec. 1.3.

Research question 1:
Which factors influence costs and benefits in PKM?

These factors have been analysed already theoretically in Sec. 3.3. These results are used to assess the next research questions.

Research question 2:
What is a suitable model to represent *and* use *artefacts in a uniform fashion that are in different degrees of formalisation?*

[17] p_1: 1255 Items, p_2: 1341 Items, p_3: 1742 Items, p_4: 1711 Items, and p_5: 1168 Items.

First the *representation* abilities of CDS are evaluated, then the efficiency of *using* them.

Hypothesis 1: The implemented CDS Tools can successfully be used to represent artefacts in a uniform fashion that are in different degrees of formalisation.

Informal, semi-formal and formal knowledge

The content of ContentItems and the Statements with an informal Relation (M_S0) are considered to represent informal knowledge in a CDS model. Formally, $k_{informal} = M_C + M_R + M_Q + (M_S - M_{SFORM})$.

A NameItem has the implication that its name denotes uniquely some kind of mental concept. That is a stronger formalisation than a mere ContentItem, yet it is not as formal as a formal Statement, because no other knowledge can be derived from it. NameItems can thus be said to represent semi-formal knowledge. Formally, $k_{semi-formal} = M_N$.

Formal knowledge is represented in Statements with a Relation that implies further formal assertions (M_{SFORM}). Formally, $k_{formal} = M_{SFORM}$.

Each kind of knowledge should be present in user-created knowledge models to a substantial fraction.

Test: Averaging over all knowledge models, informal, semi-formal, and formal knowledge should contribute a substantial part.

Data: In total, 15 models have been created. Together, they contain 7217 Items, of which 641 are built-in Items. Built-in Items are present in every model, i. e., the built-in Relations. Each model contains 41 built-in Items, some additional built-in Items are added if needed, e. g., an Item that contains the HKW online help. The remaining 6576 user-created Items are distributed as follows:

Built-ins

User-created items

315	(5 %)	ContentItems,
1416	(21 %)	NameItems,
704	(11 %)	Relations[18], and
4141	(63 %)	Statements.

The *average* values across all 15 models measuring 6556 Items (excluding built-in Items) are

informal	37.36 %	(SD=22.41),
semi-formal	23.63 %	(SD=14.22), and
formal	40.01 %	(SD=18.78).

Eight out of the 15 *individual* models are well-balanced and have at least 15 % of their Items in each category. The remaining seven models have sometimes more informal (3 ×), sometimes more semi-formal and formal knowledge (6 ×). Even if some people model more informal and other model more formal, the overall hypothesis is accepted: CDS can successfully be used to represent artefacts in a uniform fashion, which are in different degrees of formalisation.

[18] inverse Relations count as a second Relation

Hypothesis 2: The implemented CDS Tools can successfully be used to benefit from artefacts in a uniform fashion that are in different degrees of formalisation.

Efficient and effective usage of knowledge models?

Test: The prototypical CDS Tools should perform as well as comparable, mature tools. Performance is measured in terms of efficiency and effectivity. A comparable, mature tool is Semantic MediaWiki (SMW), which has been introduced in Sec. 2.9. CDS Tools should have no significant difference in efficiency and effectivity compared to SMW.

Data: To avoid a scenario-dependent bias and have an equal number of sessions for CDS Tools and SMW, scenario A (which has not been performed with SMW) has been excluded.

The resulting values for e_1, e_{10}, e_{100}, efficiency, and effectivity are shown in Table A.4 in Appendix A.5.5. CDS Tools reach an average effectivity of 89 % of SMW and an efficiency of 85 % compared to SMW. The already small differences between the means are also not significant (Pairwise t-tests yield effectiveness: $p = 0.1997$ and efficiency: $p = 0.3325$).

CDS Tools has an efficiency and effectiveness comparable to SMW. CDS Tools has been built by two persons from scratch. SMW, on the other hand, builds on the stable MediaWiki code base and has itself been build by a large number of persons, e. g., the annotation extensions from *Ontoprise*. Therefore it can be expected, that the usability of CDS Tools ultimately trumps that of SMW. Note also that CDS has, e. g., a higher expressivity compared to the SMW data model.

The hypothesis is accepted.

Hypothesis 3: The CDS model is as easy to learn as comparable models.

Easy to learn?

Test: Each participant has been asked to rate how easy it is to learn each tool (HKW, iMapping, QuiKey, and SMW). It is expected that CDS Tools score on average as well as SMW.

Data: On average, learnability of CDS was rated 2.4 and SMW only 3.0 (in German school grades, with 1=best and 5=worst). For every individual participant, the average rating of the three CDS-based tools is better than the rating of SMW. However, again due to the small sample size: The difference is not significant ($p = 0.1369$). One participant (p_5) found QuiKey harder to learn than SMW, another one (p_2) found HKW harder to learn[19]. This hypothesis is regarded as confirmed.

Research question 3:
What is a suitable top-level ontology for personal knowledge models?

Hypothesis 4: Participants rate the proposed CDS relation ontology as easy to understand and learn.

[19] Participant p_2 had to perform all tasks one week ahead of the other participants, so he encountered the most bugs in HKW.

Test: The average rating given in the retrieval interview should be 2 ("good") or better.

Data: The average rating given by the five participants is 3 ("satisfactory"). The hypothesis cannot be accepted.

Hypothesis 5: The relations of the CDS relation ontology are used in user-created statements. The CDS RELATION ontology ($\mathfrak{R}$) provides a set of 13 RELATIONS. If this ontology is useful, one should expect users to create a number of STATEMENTS using RELATIONS from this ontology. However, $\mathfrak{R}$ is designed to be extended, so a fair amount of STATEMENTS should contain user-created RELATIONS as well.

Test: At least one third of user-created STATEMENTS should use the RELATIONS from $\mathfrak{R}$. At least one third of user-created STATEMENTS should contain user-created RELATIONS.

Data: The total number of statements in the 15 models is 4141, with 2188 (53 %) using a user-created RELATION and with 1953 (47 %) STATEMENTS using a CDS-built-in RELATION.

Which built-in RELATIONS have been used most in user-created STATEMENTS? For this analysis, a RELATION and its inverse RELATION are summed up. Table A.6 in Appendix A.5.5 shows the usage of the built-in RELATIONS in all STATEMENTS created by participants in the 15 models. About half of the built-in RELATIONS are used. Two built-in RELATIONS have not been used at all: [annotates] and [replaces]. The RELATION [annotates] is a common super-RELATION for [is tag of] and [has instance]. For the RELATION [replaces], which should at *edit-time* replace one string with another one, there was simply no tool support in all three CDS tools. Participants were apparently clever enough not to invest their time in creating STATEMENTS using this not-yet-supported RELATION.

The hypothesis is confirmed.

Hypothesis 6: Users create their own relations.

Test: Every individual participant should have created their own RELATIONS.

Data: 704 RELATIONS have been created by users. The distribution per participant is:

p_1	96
p_2	182
p_3	188
p_4	140
p_5	98

The hypothesis is accepted.

Hypothesis 7: User-created relations extend the built-in relation ontology. Test: User-created RELATIONS should be linked via [has subtype] to RELATIONS of the built-in RELATION ontology.

Data: The built-in RELATION ontology has been extended by all participants in many ways. There are 457 user-created [has subtype]-STATEMENTS

between RELATIONS. Appendix A.5.6 shows the RELATION (and concept) hierarchies from each participant in each scenario.

A look at the actual relation hierarchies (cf. Appendix A.5.6) – instead of defining sophisticated tree-metrics – should be enough to see that the CDS RELATION ontology was indeed re-used and extended. The hypothesis is regarded as confirmed.

Research question 4:
How can a tool for using personal knowledge models be built?

Hypothesis 8: CDS Tools are as interaction efficient as a comparable, mature semantic modelling tool. A wiki page is roughly a title and a snippet of content. Therefore, creating an SMW wiki page corresponds to creating a NAMEITEM and a CONTENTITEM in CDS.

Test: The sum of CONTENTITEMS and NAMEITEMS created by CDS Tools should both be in the same range as twice the number of wiki pages created in SMW. Formally: $M_C + M_N \sim 2 \times |\text{SMW wiki pages}|$.

Data: For scenarios B and C, there were 198 CDS CONTENTITEMS and 1045 NAMEITEMS created. In SMW 647 wiki pages were created. $M_C + M_N = 1243$ and $2 \times |\text{SMW wiki pages}| = 1294$ are very similar. The hypothesis is accepted.

Hypothesis 9: CDS Tools produce more user-created triples than a comparable, mature semantic modelling tool. Looking at the number of created ITEMS or STATEMENTS does not tell anything about the created "information content"[20] in the knowledge model. The number of non-trivial, non-technical semantic facts needs to be estimated.

First a quick comparison of STATEMENTS made in CDS vs. property values expressed in SMW. An n-ary property leads to n property values. Here is the sum for scenario B and C:

participant	CDS	SMW
p_1	527	437
p_2	548	697
p_3	1104	844
p_4	503	758
p_5	352	2111

A non-trivial triple in CDS is a STATEMENT, its inverse STATEMENT[21], as well as TRIPLES implied by RELATION and concept hierarchies. A query for all TRIPLES in a model yields M_T. The following patterns have been considered trivial and have been removed:

[20] From the perspective of the user. From information theory, e. g., for a user it makes no difference whether a string is represented in RDF as a string-data-typed literal or a plain literal. For information theory, it does.

[21] An inverse STATEMENT should be considered non-trivial, because a user really profits from it in queries. As an example, a user notes two persons that [work for] the KIT and later adds third one as a person that the KIT [employs]. Now she can query her CDS model for all ITEMS that ([KIT], [employs], $*$) and get all three persons back.

- $(x, [\text{is subtype of}], [\text{is related to}])$ – every RELATION is a subtype of [is related to] automatically. A total of M_R needs therefore to be deducted from M_T.
- $(x, [\text{is related to}], y)$ – every STATEMENT implies that source and target are related by [is related to]. The number of TRIPLES with this pattern is the number of STATEMENTS M_S.
- All 411 additional triples following from built-in STATEMENTS and RELATIONS.

The final formula to compute the number of non-trivial triples in CDS is

$$M_{non-trivial-triples} = M_T - (M_R + M_S).$$

SMW offers only a transitive category hierarchy, hence the transitive closure can be calculated and taken into account. The number of non-trivial triples in SMW is computed by

1. Downloading the RDF from each page.
2. Merging all obtained RDF files in a triple store.
3. Removing trivial triples that do not reflect user-created semantics. A user not being aware of RDF would never ask to see *all defined labels*. Instead, she is expected to deal with, e. g., persons, spices, sharks and other concepts from her real or imaginary world.

   In detail, the following technical[22], non-user-created triple patterns have been removed:

   - $(x, \text{owl:imports}, y)$ – imports SMWs swivt ontology[23].
   - $(x, \text{rdfs:isDefinedBy}, y)$ – links every concept to the RDF document.
   - $(x, \text{rdfs:label}, y)$ – in CDS, an ITEM and its label are only counted as one entity.
   - $(x, \text{swivt:page}, y)$ – links every concept to the Wiki page.
   - $(x, \text{swivt:creationDate}, y)$ – like the others, this is not a user-created triple.
   - $(x, y, \text{swivt:Container})$ – an artificial container to group several values of an n-ary property assignment. The n assignments of such a construct are not trivial, but the artificial container is. At least in the cases of the user study, where the order of property assignments played no role.
   - $(x, y, \text{swivt:Subject})$ – every concept is also a swivt:Subject.
   - $(x, y, \text{owl:AnnotationProperty})$ – classifies properties that link to a data value.
   - $(x, y, \text{owl:Class})$ – trivial, as every node z in a triple $(x, \text{rdf:type}, z)$ is an owl:Class, at least under OWL semantics.

[22] The list of all properties being an owl:ObjectProperty is of no practical value – unless you are doing ontology statistics.

[23] `http://semantic-mediawiki.org/swivt/` (accessed 05.01.2010)

- $(x, y,$ owl:ObjectProperty) – classifies properties that link to another concept.
- $(x, y,$ owl:Ontology) – a container for all properties used on a page.

4. Materialise all triples that can be inferred under RDFS semantics;
5. Remove all axiomatic RDFS triples[24]
   (such as `rdf:type rdf:type rdf:Property`, `rdfs:subClassOf rdfs:domain rdfs:Class`, or `rdfs:comment rdfs:range rdfs:Literal` and the triples that can be inferred from these axiomatic triples);
6. Remove trivially inferred triples. In detail these patterns have been removed:
   - $(x,$ rdf:type, rdfs:Resource) – every node in RDF is a resource.
   - $(x,$ rdf:type, rdf:Property) – every node used as the relation of a triple is an rdf:Property.
   - $(x, *, x)$ – triples linking a node to itself such as $(x,$ rdfs:subPropertyOf, $x)$ and $(x,$ rdfs:subClassOf, $x)$ are trivial.
7. And finally counting the remaining number of non-trivial triples.

Test: The number of non-trivial triples in CDS Tools must be higher than the number of non-trivial triples in SMW.

Data: Here is the sum per participant for scenario B and C:

participant	CDS	SMW
p_1	4265	1288
p_2	4697	734
p_3	6725	349
p_4	3446	2072
p_5	4092	1798

There are significantly ($p = 0,0252$, two-side paired t-test) more non-trivial triples in CDS Tools than in SMW. The hypothesis is accepted.

[24] Listed in `http://www.w3.org/TR/rdf-mt/` (accessed 06.01.2010)

Hypothesis 10: CDS Tools receive usability ratings comparable to a mature semantic modelling tool. Usability

Test: The mean usability rating for CDS Tools should be comparable to SMWs usability rating.

Data: The mean usability rating for CDS Tools overall was 2.13 compared to 2 for SMW (in German school grades, 1=best and 5=worst). This difference is not significant. However, looking at the individual ratings reveals means of 2.8 for HKW, 1.6 for iMapping, and 2.0 for QuiKey. This indicates that HKW has the worst usability.[25] The hypothesis is accepted.

6.3.3. Discussion

Overall, the CDS data model ($\mathfrak{D}$) performed well and lived up to its expectations.

The CDS data model has successfully been used to represent and use (retrieve) artefacts in a uniform fashion that are in different degrees of formalisation (hypotheses 1 and 2).

Furthermore, it has been perceived as easy to learn (hypothesis 3).

The Relation ontology ($\mathfrak{R}$) was not perceived as easy to learn (hypothesis 4). However, users still used the built-in Relations to a high degree (hypothesis 5). The dual role of the CDS Relation ontology ($\mathfrak{R}$) might not have been taken into account well enough in the design of tools: On the one hand, $\mathfrak{R}$ has been designed to be able to represent knowledge imported from a number of sources. This import from existing sources – such as file systems, documents, mind- and concept maps, and wikis – has been validated by theoretical analysis. $\mathfrak{R}$ has been *designed* as a generalisation of the aforementioned models. On the other hand, $\mathfrak{R}$ is meant to be used by users in Statements and as super-Relations for their own Relations. Using $\mathfrak{R}$ Relations as super-Relations of user-defined Relations allows converters to *understand* them and export them in a suitable form, retaining the intended semantics. The Relation [has tag] is most likely used for tagging artefacts such as files or web pages. As none of them needed to be organised for solving the scenarios, it is not surprising that users did not use this relation.

Users extended the CDS Relation ontology with their own Relations (hypothesis 6 and 7). Although also possible in SMW, no single user created any sub-relation or sub-class in SMW. This might have been underemphasized in the introduction session or the tool support in SMW+Halo might not be appropriate.

Although still research prototypes, the CDS Tools had acceptable interaction efficiency and usability ratings compared to SMW (hypothesis 8 and 9). Interestingly, the number of ContentItems in CDS Tools is significantly

[25] On the other hand, HKW supported most features of the CDS model. HKW might have been rated less good because of the more complex user interface required to support all features.

less ($p = 0.0489$) than the number of wiki pages in SMW.[26]The knowledge modelled in CDS Tools seems to be more conceptual than merely textual.

Using CDS Tools, users produced significantly more non-trivial triples than with SMW (hypothesis 9).

As a summary, CDS Tools have already in their prototype state a usability comparable to SMW. Participants extended the RELATION hierarchy in CDS, but not in SMW. They created less (unformalised) CONTENTITEMS and significantly more non-trivial triples in CDS, compared to SMW. The CDS data model has been rated as easier to learn than the SMW model.

6.4. Comparing CDS and RDF

Compared to existing Semantic Web standards, $\mathfrak{D}$ resembles most the RDF data model. The RDF specification defines both the basic data model, as well as a core vocabulary and its semantics to describe entities like lists, containers or reified statements. $\mathfrak{D}$ is an extended subset of RDF. Comparing the entities of RDF with $\mathfrak{D}$:

Removed from RDF are blank nodes and literals

URIs: Each ITEM in $\mathfrak{D}$ has a unique URI.

Blank nodes: No blank node concept exists in $\mathfrak{D}$. Blank nodes are a rather complicated technical concept which is not needed in $\mathfrak{D}$, but can be emulated on higher layers if needed. A detailed discussion is given by Heitmann et al. (2006).

Data-typed Literals: All literals in $\mathfrak{D}$ are plain literals. Typed literals are introduced in the same way as typed instances: On the $\mathfrak{R}$ layer. As an example, modelling a floating point number in CDS is achieved by storing the literal value, say "3.14" in a CONTENTITEM c and adding a STATEMENT $(c, [\text{has type}], [float])$. Float is not a built-in concept of CDS. Again, moving this feature from built-in in RDF to an optional feature on a higher layer in CDS makes $\mathfrak{D}$ easier to understand and use without limiting its expressivity.

Language-tagged literals: CDS is meant to be used by one person. There are no multiple values in different languages.

Literals : CDS $\mathfrak{D}$ has no "plain literals". Instead, each ITEM (identified by a URI) has exactly one literal attached.

Many semantic web applications hide the complexity of RDF completely for the user – for a good reason. Most users do not want to deal with URIs, blank nodes, literals, data-types for literals, language tags for literals and other RDF subtleties such as the recursively defined `rdf:List` construct. Some early semantic web applications *do* show URIs to the end user – such user interfaces are now considered immature and prototypical.

[26]None of the SMW wiki page contents was short enough to be a legal NAMEITEM; the average wiki page was 470 characters long. 1.6 % of wiki pages were redirects – this is SMWs equivalent of CDS' [has alias].

$\mathfrak{D}$ also extends RDF. The following *features* have been added, compared to RDF:

Added to RDF are addressable literals, add. statements, mandatory inverse relations, and name items

Addressable Literals: The fundamental concept of CDS $\mathfrak{D}$ is the ITEM. Each ITEM is first addressable via a URI and second it may contain zero or one content. No content can appear outside of ITEMS. Each piece of content is thus addressable, which is used, e. g., to record creation date and authorship of each ITEM. RDF does not allow addressing literals.

Addressable Statements: As STATEMENTS in $\mathfrak{D}$ are special ITEMS, all STATEMENTS also have a URI. Hence all CDS STATEMENTS are addressable. In RDF, statements are *not* directly addressable and reification and named graphs do not solve this either, out of the box.

Inverse relations: RDF and RDFS do not define inverse relations, i. e., they have no means to define them. OWL does define inverse relations, but they are not mandatory. In $\mathfrak{D}$ inverse relations are mandatory.

Name items: CDS has the notion of NAMEITEMS, which allow the user to address ITEMS via a memorable string. This was inspired by the usage of WikiWords in wikis. NAMEITEMS are ITEMS where the content has naming characteristics. RDF does not have a *naming concept* for humans.

CDS data model has same expressivity as RDF Lossless conversion is possible

The CDS data-model has the same expressivity as RDF. Each RDF statement (s, p, o) can be represented as two ITEMS (s and o) and a relation (p) with an inverse ($-p$).

The similarity between CDS and RDF allows converting RDF to CDS (creating new URIs for literals and lifting them to ITEMS). As an example, the RDF statement $(s, p,$ "value") is represented in CDS as (s, p, o) with o being a CONTENTITEM with the content "value".

Comparing CDS to RDF Schema and OWL CDS is intended to be used by end-users, not for data exchange between machines. But as RDFS has been a strong inspiration for the semantics of CDS, a brief comparison is presented.

Class and Instance: CDS does not clearly distinguish between the two. Rather, every ITEM can be used as a type for another ITEM. This is the same in RDFS, but different from OWL.

Domain and Ranges: CDS has no notion of domains and ranges. Every relation can be used with every kind of ITEM.

Class hierarchies: CDS has a [has subtype] relation, similar to RDFS' `subClassOf`.

Property hierarchies: The same RELATION ([has subtype]) is also used for property hierarchies. Hence it is also similar to RDFS' `subPropertyOf`.

Same As: Different from RDFS, but more similar to OWL, CDS has several ways to state similarity ([is similar to]) or equality ([is same as]).

A mapping from CDS to RDFS has been described in Sec. 4.4.2. Customisation of the RDF export function have been described in Sec. 5.1.

6.5. Related work

This section briefly presents related work. As the contribution of CDS is on different levels (data model $\mathfrak{D}$, RELATION ontology $\mathfrak{R}$, STIF, and HKW), the related work is clustered accordingly.

6.5.1. CDS data model

First, works related to the CDS data model $\mathfrak{D}$ are presented.

A comparison of CDS with **RDF** (cf. 3.6.1) has been presented in Sec. 6.4.

Simpler than RDF is **XML**, which has been introduced in 2.5. Once an XML Schema has been defined, XML tools can support the user with rich authoring support, e. g., auto-completion for element names and auto-creating required children elements. However, XML tools expect the schema to be stable. This assumption does not work for personal knowledge management.

Although not formally specified, the data-model of **mind-maps** (cf. Sec. 3.5.5) is comparable to the CDS data-model, as both models have been created to let humans easily create knowledge cues. Mind maps do not support formal knowledge at all.

The **UML** (see Sec. 2.5) is obviously not adequate for personal notes. As a trivial example, even free-form text is only possible as an annotation to another structural element.

Hyper-object substrate (HOS) is a groupware system built on the idea of incremental formalisation (Shipman and McCall, 1999). HOS uses prototype inheritance. The basic model of HOS is a mix of a hypertext and a frame-based system. Compared to CDS Tools it resembles much less mind-map or wiki but more tools like Microsoft Access or Protégé. HOS provides active support in formalisation using a set of built-in rules. The purpose of HOS is different from CDS, as HOS is meant to create domain-oriented systems, whereas CDS is designed for an open-end personal use.

The **iMemex** (Dittrich, Salles, Kossmann, and Blunschi, 2005) project takes another approach: iDM (Dittrich and Salles, 2006) is a unified data model which aims to dissolve the boundaries of files. Their approach is to compute a unified, structured, *read-only*, query-able resource view graph. The implementation optimizes on storage space and query answer times.

The **unified web model** (Immaneni and Thirunarayan, 2007) is a unified view on hypertext (WWW) and semantic web supplemented with a query language. Although the semantic web is typically characterised as an extension of the existing web (Decker et al., 2000), there are few formal models describing the resulting mix of content *and* meta-data. I. e., almost

all semantic web models deal only with fine-granular instance data and ontologies, whereas classical content management paradigms have usually very limited metadata and formalisation abilities.

The unified web model is rather technical and unified authoring is not in the scope of the work. However, the unified web model has a string resemblance to the CDS data model. Another related model is *Xanadu* which has been presented in Sec. 3.5.2.

6.5.2. CDS relation ontology

This section presents works related to the CDS Relation ontology $\mathfrak{R}$.

The **Simple Knowledge Organisation System** (SKOS) model (Miles and Bechhofer, 2008, cf. Sec. 3.6) has been designed to represent thesauri and taxonomies. Regarding the evaluation of the CDS Relation ontology $\mathfrak{R}$, SKOS seems to contain all essential relations for authoring personal knowledge cues. Note that SKOS has not been designed for personal knowledge management and this congruence is rather accidental. Second, SKOS has been designed for usage with RDF, a data model that is inadequate to be directly used for personal knowledge models.

The **Personal Information Model** (PIMO, Sauermann, 2009) has been developed in the European Integrated Project NEPOMUK, like CDS. The basic concept of "a PIMO" resembles the notion of a "personal knowledge model" as used in this thesis. PIMO has been designed for usage in a semantic desktop (cf. 2.6). The core concepts of PIMO are classes like Thing, Collection, Group, Location, LogicalMediaType, Organization, Person, ProcessConcept, and Topic. Core relations are "related", "has part" (which is used in a generic way, almost as generic as CDS' [has detail]), and "has topic". Different from CDS, PIMO makes an explicit distinction between class and instances as well as between Things and Topics. This follows closer the OMG layered modelling approach and has the benefit that user interfaces can exploit these constraints. The drawback is that new users (1) need to learn more concepts and (2) are required to take decisions, e. g., to define class and instance levels properly. In CDS, these decisions are still beneficial to the user, but not strictly required. Overall, goals and design of PIMO and CDS are rather close, which is no surprise as both have been developed in the same research project and many meetings between both developers have been conducted to ensure a certain degree of compatibility. E. g., the class hierarchy of PIMO can be re-used almost unchanged in CDS.

6.5.3. Structured text interchange format (STIF)

This section presents works related to the STIF model.

BBCode[27] is an unofficial best practice for formatting text in forum systems. BBCode uses syntax like "`[b]hello[/b] world`" to denote the XHTML equivalent "hello world" which is rendered in a browser as "**hello** world". The rationale behind BBCode is an easy format to

[27] `http://en.wikipedia.org/wiki/BBCode` (accessed 06.01.2010)

allow certain text formatting without opening the can of worms associated with malicious JavaScript code. It is used on collaborative websites to let community users contribute formatted text without being able to introduce malicious content. The supported subset of XHTML is roughly the same as the one provided by STIF.

DocBook (Walsh and Muellner, 1999) is a powerful XML-based format for modelling a document. It is used to create type-set books as well as online manuals – from the same source file. As DocBook is a rather complex format, **Simplified DocBook**[28] has been created. Simplified DocBook contains *just* 119 elements, 555 entities, and 29 notations. Like other XML-based formats, DocBook is not designed for ad-hoc extensibility.

More work related to STIF is given by Völkel and Oren (2006).

6.5.4. Hypertext Knowledge Workbench

This section lists works related to the Hypertext-based Knowledge Workbench (HKW). Besides Semantic MediaWiki, which is presented in Sec. 2.9 and used in the comparative user study (Sec. 6.3), there are a number of tools related to HKW.

Tabulator[29] by Tim Berners-Lee is a resource-oriented RDF browser. There are no authoring facilities.

TiddlyWiki[30] is a wiki implementation in a single JavaScript file, which makes deployment and personal use particularly easy. TiddlyWiki supports wiki links and tags but no formal knowledge.

Ludwig (2005) sees redundancy within and among documents as a hurdle to efficient information usage. He questions if documents are the best container for knowledge representations and proposes to work more direct with redundancy-free semantic knowledge management systems. In such a system, the traditional notion of a document is replaced by virtual documents, which render parts of the knowledge base as an interactive tree. The system **Artificial Memory**[31] supports – different from most semantic systems – ordered lists of content snippets.

Bernstein (2006) describes **TinderBox**, a "personal content management assistant", which offers sophisticated HTML generation via templates. Based on an internal structure, it allows editing data and structures both in a map as well as in an outline view. TinderBox provides prototype inheritance. The basic knowledge model is frame-like, links are only visible in the map view.

SEMEX (Cai, Dong, Halevy, Liu, and Madhavan, 2005) is a system that creates a unified view over data scattered in different desktop applications. It allows unified search and associative browsing. Authoring of associations is not provided.

[28] `http://www.docbook.org/schemas/simplified` (accessed 06.01.2010)
[29] `http://www.w3.org/2005/ajar/tab` (accessed 06.01.2010)
[30] `http://www.tiddlywiki.com/` (accessed 06.01.2010)
[31] `http://www.artificialmemory.net/` (accessed 06.01.2010)

Started as a PhD thesis under the name **Popcorn** (Davies, 2005; Davies et al., 2006), the tool became the commercial tool **Notewise**[32]. Popcorn emphasises knowledge refactoring and strives for scalability to large knowledge bases.

CODE4 (Skuce and Lethbridge, 1995) is an environment for managing conceptual knowledge. It supports the concept of incremental formalisation. Interestingly, CODE4 treats statements also as concepts about which further statements can be made. CODE4 supports property inheritance to supply default values for properties. It mandates a strict separation of meta-concepts and user concepts, like in OWL. CODE4 has been successfully used to create new knowledge as an act of creativity – made possible by an existing knowledge base (Lethbridge, 1991a). CODE4 has many more concepts for a user to learn, but offers also more modelling primitives, e. g., n-ary relations. HKW has a stronger bias on informal and semi-formal knowledge.

Jourknow is touted an "information scrap manager" (Bernstein, Kleek, monica mc schraefel, and Karger, 2008) with an emphasis on taking notes and re-finding them by keyword, tags, context or automatically taken photos. Jourknow has no ability to formalise data.

Haystack (Adar, Karger, and Stein, 1999; Quan, Huynh, and Karger, 2003) allow the user to manage a large set of information items in a homogenous way. A central idea is the notion of a collection, which may contain items of different kinds. The tool is built using a range of innovative ideas such as a RDF-based user interface description language as well as a programming language represented in RDF as well. The visual appearance is aesthetic and thought out. Nevertheless, the complexity and flexibility of Haystack is intimidating. Because of the flexibility, it is unclear how to represent new facts for which no pre-defined semantic types have been created. Overall, Haystack is slightly too schema-bound for the flexible PKM use-cases that motivated the development of HKW.

6.6. Conclusions

This chapter presented different kinds of evaluation for the contribution in this thesis.

The data-model ($\mathfrak{D}$), relation ontology ($\mathfrak{R}$) and structured text model (STIF) have been compared to the evaluation criteria established in Chapter 3 (cf. Sec. 6.1). The CDS data model, the RELATION ontology, and the STIF fulfill most criteria.

HKW has been evaluated in a formative user study with 16 people from 4 companies (cf. Sec. 6.2). As a result, the user interface has been reworked.

Furthermore, HKW has been evaluated in an exhaustive 125-hour comparative user study (cf. Sec. 6.3) in which all three CDS Tools (HKW, QuiKey, and iMapping) have been compared against Semantic MediaWiki (SMW). The STIF features have not been used much in the user studies.

[32] `http://www.notewise.com` (accessed 06.01.2010)

Future CDS tools might present STIF features with a rich text editor instead of wiki syntax. The CDS data model was rated as easy to learn. CDS Tools – although still prototypes – performed in the evaluation scenarios comparable to the mature SMW tool.

The RELATION ontology was not perceived as easy to learn. It seems, even 13 RELATIONS and their inverses need a lot of teaching and documentation, in order to be understood in detail. However, users can learn the built-in CDS RELATIONS one at a time. There is no need to understand them in the beginning, in fact, a user is not even required to know the existence of the built-in RELATIONS in the beginning. They are required, however, to formalise the connection between knowledge cues step by step. In this respect, iMapping has a promising user interface approach, which hides most RELATIONS and exposes one (namely `[has detail]`) prominently.

The CDS data-model model has been compared to the RDF model (cf. Sec. 6.4). It has the same expressivity but a smaller number of concepts.

7. Discussion, Future Work, and Conclusions

This chapter discusses the main results of this thesis, shows potential future work, and concludes the thesis.

7.1. Discussion

This section reviews the main contributions of this thesis and discusses them critically. Fig. 7.1 shows the relations and types of contributions.

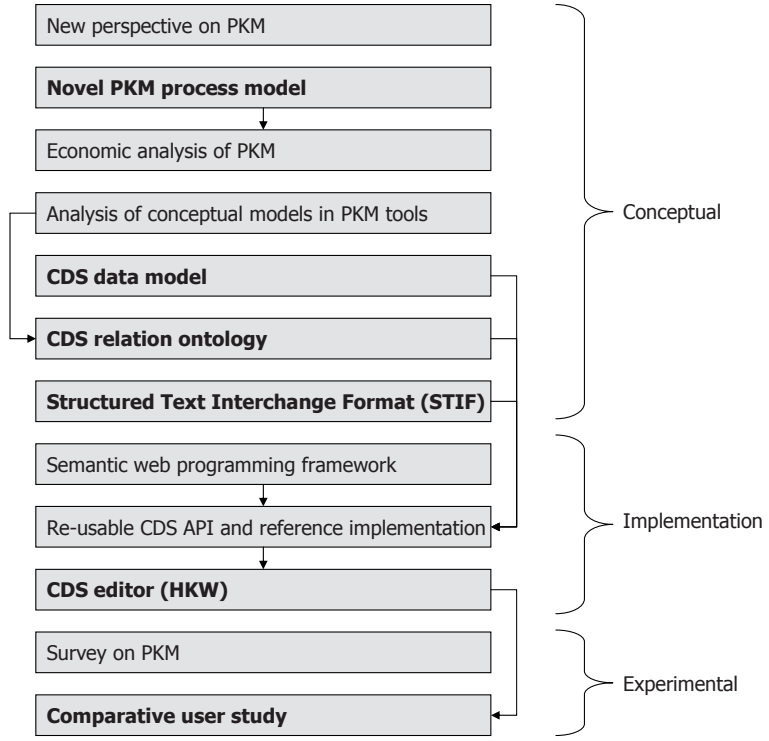


Figure 7.1.: Overview of contributions. Arrows indicate re-use of contribution within this thesis. Entries in bold are considered main contributions.

Contribution: A new perspective on PKM

Type: conceptual.

Knowledge cue

The notion of a *knowledge cue*, which has been introduced in Chapter 1, allows a pragmatic analysis of tools without debating the differences between information and knowledge.

PKM and OKM

Although a lengthy comparison between PKM and organisational knowledge management (OKM) has been given, much more could be said about this topic. Especially the interplay of individuals doing their PKM and an alignment with organisational knowledge management goals. With more individuals using PKM tools the desire for controlled sharing and questions of intellectual ownership will become more important. CDS has already some features that allow for collaborative usage of knowledge models such as globally unique identifiers for all kinds of Items, and metadata for the creation data, modification date and author of each Item.

In Chapter 2 existing work was presented. A small yet original contribution is the formal definition of a *formal model* given in Sec. 2.1.

The notion of *knowledge models* has been published by Völkel (2007a).

Contribution: A novel PKM process model

Type: conceptual.

Perspective of the individual

Existing process models for knowledge management and PIM have been reviewed, but none describes PKM satisfactory. A new process model from the perspective of the individual using knowledge cues has been presented in Sec. 3.2.2. This model explains what processes an individual knowledge worker needs to perform.

An earlier version of the process model has been presented in an invited talk at the *AKWM Symposium 2008*.[1]

Contribution: An economic analysis of PKM

Type: conceptual.

Building upon the knowledge cue life-cycle, Sec. 3.3 has presented a novel economic analysis of these processes. This model describes how rational people should behave, it is not a description of peoples behaviour in practice.

The overall benefit of using a PKM system could be characterised by summarizing over the value of successfully retrieved knowledge items (content or formal statements) for each task. Costs could be characterised as the sum of the costs of all authoring and structuring efforts. A quantification of the effect more structuring has on lower retrieval costs (or improved benefit) cannot be stated unless the semantics of the formal statements and details of the search process (browse, search, follow links) are specified. Especially the analysis of *value* turned out to be difficult. Without precise task definitions the value of retrieving a knowledge cue at a certain moment is impossible.

[1] The homepage of the event can be found at the website of the AKWM `http://www.arbeitskreis-wissensmanagement.org/wm-symposium-2008-2664.htm` (accessed 03.01.2010). The presented slides are linked from `http://pubs.xam.de` (accessed 03.01.2010).

Unfortunately, it is a characteristic of many knowledge workers that their tasks are not highly repeatable.

On the other hand, some knowledge workers *do* perform the same tasks again and again, e. g., processing insurance claims or examining X-ray pictures to classify bone fractures. In theses cases, the economic analysis could help to choose among several systems and user interfaces.

However, within this thesis, the economic model has "only‘" been used to gather an important high-level requirement: A good model to represent knowledge cues should be a super-set of conceptual models in existing tools which are used to perform PKM-tasks.

The analysis has been published by Völkel and Abecker (2008).

Contribution: An analysis of conceptual model in PKM tools
Type: conceptual.

This analysis of conceptual models and their relations has been presented in Sec. 3.5 and Sec. 3.6.3. It can serve as a guideline for future user interfaces that aim to be "mentally backwards-compatible" with existing tools – resulting in a lower learning curve.

The analysis of requirements for knowledge models from literature (Sec. 3.4, summary on page 3.7) can be used to evaluate existing and planned PKM tools.

The analysis method has initially be published in (Völkel, Haller, and Abecker, 2007).

Contribution: The CDS data model
Type: conceptual.

Chapter 4 presented CDS as a formalism to represent and use knowledge cues in different degrees of formalisation. This solution has a central data model ($\mathfrak{D}$, cf. Sec. 4.1) that can represent knowledge cues. This data model alone fulfills some important requirements from the requirements list, but it is not enough for PKM. $\mathfrak{D}$ is similar to existing approaches like, e. g., RDF and SKOS, but simpler and better suited to be used as a conceptual model. Unlike RDF, the conceptual model of CDS is designed to be used directly by users. Indeed, users rated the building blocks of CDS as easy to learn (cf. hypothesis 3 on page 220). However, as only computer science students performed the rating, it remains unclear if people with different educational background will find the CDS data model easy to learn. Given the low number of core concepts in the model – NameItems, ContentItems, Relations, and Statements– this seems to be likely.

Due to the high expressivity and flexibility, the CDS data-model can also serve as an exchange format between different PKM tools.

The first publications of the CDS data model can be found in Völkel and Haller (2006). Subsequent versions have been described in (Völkel, Oren, and Schaffert, 2008; Völkel and Haller, 2009).

Contribution: The CDS relation ontology
Type: conceptual.

The CDS relation ontology ($\mathfrak{R}$) contains the most-commonly used relations from popular tools used for doing PKM tasks. This ontology is used in the CDS tools but also has a value outside of them. In the design or evaluation phase of PKM tools, these relations can be taken as a guideline. If one of them is missing, a certain kind of functionality or import/export ability is missing.

This has been co-developed with the CDS data-model and appeared in the same publications.

Contribution: The Structured Text Interchange Format
Type: conceptual.

The data-model by itself is of limited usefulness. The STIF concepts presented in Sec. 4.3.1 allow (a) using structured text inside knowledge cues as a middle ground between collections of keywords and fully formal models; and (b) creating semantic STATEMENTS at low interaction costs using semantic wiki syntax . The STIF format can serve as an exchange format for structured text between word processors, Mind- and Concept-Map tools, and wikis. The open-source project *WikiPipes* is using STIF to import and export wiki pages from different wiki engines in a neutral way.

The first publication on a *Wiki Interchange Format* appeared in (Völkel and Oren, 2006). The generalised STIF format has been described in (NEPOMUK Consortium et al., 2008, p. 73).

Contribution: A semantic web programming framework
Type: implementation.

Although not described in this thesis[2], the underlying frameworks RDF_2Go and RDFReactor that have been created by the author of this thesis constitute a relevant contribution to the semantic web community. RDF_2Go has been used in numerous open-source projects and the liberal license allows commercial projects to be based on it, too. RDFReactor is used in this thesis by the iMapping prototype to store the graphical layout of items.

RDFReactor has been published in (Völkel and Sure, 2005) and (Völkel, 2006). The *semweb4j* framework that includes RDFReactor is described in Völkel (2005b).

Contribution: A re-usable CDS API and reference implementation
Type: implementation.

Chapter 5 the CDS API and its reference implementation (Sec. 5.1 and Sec. 5.1) have been presented. They have been used successfully for three different tools. They offer adequate performance and memory consumption characteristics to be used for commercial PKM tools. The Java implementation contains re-usable data structures for other implementations such as

[2]Because they are very technical and not linked well to the personal knowledge management topics. And they would have required even *more* space!

a type-safe generic triple-index or a thread-safe CDS model implementation that supports optional content compression.

The first implementation has been published as *swecr*[3] by Völkel (2007b).

Shortcomings of CDS There are several points not yet tackled in the CDS model:

- The notion of data-types is possible, but not defined for CDS models. In the future, a set of data-types should be defined together with definitions on comparing for equality and a defined sort-order. Most work from XML Schema Types (XSD, Thompson et al., 2008) can be re-used, as it has been done in RDF and OWL. However, given the nature of CDS as an end-user format, the data-formats should be inspired more by existing tools like spreadsheet applications, so that the end-result is not overly technical.

  Only such a definition of data-types would allow using aggregation function in queries.

- The presented CDS query language is most likely not rich enough to cover all relevant PKM tasks. This query language should be extended at least to the power of SPARQL. Given the triple-like nature of CDS, almost all SPARQL features can be re-used on CDS models. The hardest part will be the definition of an adequate textual or CDS representation. SPARQL misses full-text queries which also need to be added. Minack, Sauermann, Grimnes, Fluit, and Broekstra (2008) describe a promising mix of both approaches for *Sesame*, a similar approach exists also for the *Jena*[4] RDF framework.[5]

- CDS lacks a way to address parts of content within ContentItems. A possible extension could be the introduction of *soft references* pointing to structural parts of the STIF document within a ContentItem. Works from XPath can be re-used here. In general, no satisfactory solution has been proposed for the problem of maintaining references to parts of a text when the text is changed afterwards.

- In $\mathfrak{R}$ the distinction between [has alias] and [same as] might be hard to understand. The idea of directed aliases, although prominent in everyday life, is rarely used in computer systems.

[3] Semantic Web Content Repository

[4] `http://seaborne.blogspot.com/2006/11/larq-lucene-arq.html` (accessed 06.01.2010)

[5] The general idea is to split the query in two parts and execute them individually: One query part is delegated to an RDF store, the other part is delegated to a full-text index. Then a join is performed by the item URIs. Depending on the result set size, other join strategies should be favoured, i. e., first performing the full-text query and then binding the item URI in the RDF query to the resulting URIs.

Contribution: A CDS editor (HKW)
Type: implementation.

The *Hypertext Knowledge Workbench* (HKW) is an editor for CDS models that allows creating, deleting and modifying all elements of the CDS model. The editor makes heavy use of browser-based scripting to provide a reactive ubiquitous user interface.

The idea to run a CDS model including an CDS inference engine inside the browser turned out to be problematic, since the memory consumption in JavaScript is a factor of 10 to 100 times more.

HKW has been published in (Völkel, 2008).

Shortcomings of HKW The Hypertext Knowledge Workbench has several shortcomings:

- There is no way yet to embed queries (26 [queries]) into the content of ContentItems, as it is possible with ASK-queries in SMW.
- Transcluding one item into another one is not possible (30 [transclusion]). Embedded queries could also solve this.
- HKW should show more than one level of detail (22 [levels of detail]), e. g., the details of an item's details should also be visible to mimic more a document-like view. This is conceptually easy but has just not been implemented due to lack of time.
- Drag and drop support for easy refactoring (10 [refactor]) is also not present but is conceptually easy to add.

Contribution: An empirical survey on PKM
Type: experimental.

Online survey with 50 participants

Chapter 3 presented use-cases in PKM which have been gathered from a novel online survey with 50 participants. This survey has been published partially by Völkel et al. (2008).

Contribution: A comparative user study
Type: experimental.

Five participants worked together for a total of 125 hours with CDS Tools and a comparable semantic modelling tool. Their resulting knowledge models have been assessed and compared in various ways.

The user study has not been published prior to this thesis.

7.2. Future work

The CDS model is the basis of the *iMapping* (cf. Sec. 5.3.1) and *QuiKey* (cf. Sec. 5.3.2) tools, which are the central part of the ongoing PhD thesis of Heiko Haller.

Research creates more questions the more answers it gives. This remainder of this section lists interesting future work.

Requirements No requirements list is ever complete. E. g., the requirement of life-long data portability could be added to the requirements list for an ideal PKM model and tool. Long-term archiving of digital information has its own challenges, described by Lorie (2001).

Economic Analysis

Measuring knowledge models In this thesis, certain metrics for measuring knowledge models in CDS and SMW have been developed. However, a more generic and more precise metric of information content would allow assessing and comparing the interaction efficiency of PKM tools much better. This includes counting individual words and calculating their information content with metrics such as TF-IDF (term frequency-inverse document frequency, see Baeza-Yates and Ribeiro-Neto (1999)).

Of course, evaluating the information content of semantic statements is difficult to assess precisely, as some formal statements have a much higher influence than others, and hence more information content. E. g., a *has subtype*-statement has usually more influence on a knowledge model than a *has phone number*-statement.

Evaluating PKM tools The economic analysis can be performed on tools used for personal knowledge management. Here is a sketch how such an analysis could be performed on SMW:

In Semantic MediaWiki the page history allows in principle to calculate the change in information content for each edit. If additional access logs for the embedded ASK-queries would be available, too, a complete measurement of costs could be done. Given an additional hypothetical SMW extension asking the user to rate the usefulness of a query result, a complete measurement of the benefit of using Semantic MediaWiki would be possible.

Evaluate structure and semantics By measuring the information content and putting this into relation with the value of retrieved knowledge, the *value of structuring and adding formality to knowledge models can be assessed*, too. This allows fine-tuning the capabilities of future personal or collaborative knowledge models.

Evaluating PKM strategies Are many notes taken and never used? Are notes only found after longer and costly searches? Should more or less notes be taken?

The same idea – measuring all interaction costs and asking the user to give feedback on the value of retrieval results – can be applied within a PKM tool to assess the PKM strategies of an individual

CDS data-model model and API First, there are a number of obvious technical extensions that should be added to both the conceptual CDS data-model and to the API and reference implementation as well:

- Versioning – Every ContentItem should have a history of changes to its content, just like wiki pages. A knowledge model as a whole should contain a versioning history of structural changes, that is about all semantic Statements and added or deleted Items. Technically, the implementation is already quite close to the second par: Every change in the CDS model is done via change events. Those could be stored. A system for versioning RDF data is described in Völkel and Groza (2006).

  Less obvious is how to use versioning in CDS productively. What kinds of roll-backs are needed? What kinds of diffs does a user wants to see? How to visualize a diff between semi-formal knowledge models? What kind of queries about past states of a knowledge model are desirable? Answers to these questions are clearly future work.

- Graça Pimentel et al. (2000) describe how to derive new Statements between Items by mining a user's interaction traces with a knowledge model. The approach is not related to CDS, but could be adapted.

- Queries – Beyond SPARQL and full-text queries more graph-like queries are also desirable for knowledge models, e. g., which paths connect two given Items? A promising candidate for efficient graph pattern search, based on a kind of regular expressions over graphs, can be found in (Geiß, 2008, Chapter 5).

- Inference – Users might want to customize and extend their inference system. A rule-based approach might be helpful here. A good candidate for expressive yet computable rules might be ELP from Krötzsch, Rudolph, and Hitzler (2008).

Aside from these more technical extensions, the connection from PKM to OKM with CDS could be explored. A tool approaching this topic is described by Krohn, Kindsmüller, and Herczeg (2008). There are three interesting challenges:

- Sharing models with other people. The challenge is how to integrate somebody elses knowledge model into you own model without "messing" your own model. Certainly a way to control the display or usage of knowledge coming from other people is relevant. Should knowledge from somebody else be used for inferences? How to delete it after a while without deleting your own knowledge? If all these questions are answered, the next one is:

- What if you imported a model from somebody else and now the other model changed. How can you profit from these changes automatically by synchronising the shared part of the model without loosing your own changes? This question is related to versioning and ontology evolution.

- Finally, if automatic synchronising would work, whom do you allow retrieving which parts of your model? An answer requires to deal – among other problems – with access rights on a highly structured artefact.

Another possible future work could be the re-use of existing knowledge such as Wordnet (Fellbaum, 1998) for lexical knowledge or DBpedia[6] for factual knowledge. This can aside from technical issues also be seen as a case of importing other peoples (large) knowledge models.

Hypertext Knowledge Workbench and STIF It should be possible to embed queries in STIF content, so that the STIF render engine always shows live results.

Furthermore, the notion of *templates* as a means to specify a structure once and fill with data a number of times could be added to CDS-based tools, especially HKW. A similar idea implemented in SMW is the *Semantic Forms*[7] extension.

Further evaluation of CDS What factors influence acceptance and successful usage of CDS? In the evaluation only computer scientists have been exposed to mid-term usage. The iMapping tool could become stable and usable enough for casual users. This would allow disseminating CDS further and study acceptance among different user groups.

Applications of CDS

- A web-based, social argumentation system (cf. Kunz and Rittel (1970)) could be built on top of CDS. A good book on the state of the art is (Kirschner, Shum, and Carr, 2003). A bachelor thesis on discussion systems (Clemente Laboreo, 2007) concludes that CDS is a good candidate for building such a collaborative discussion system.
- Furthermore, given ability to import RDF to CDS and to export CDS to RDF, CDS-based tools could be used as generic RDF editors.

7.3. Conclusions

This thesis introduced the concept of a *knowledge cue* (Sec. 1.2.3) as an artefact that can remind a person about previous personal knowledge. An effective and efficient management of such knowledge cues in the form of *personal knowledge models* (Sec. 1.2.3) allows shifting individual cognitive limits.

This thesis has analysed factors that influence costs and benefit in PKM. The results are document in the novel knowledge cue life-cycle (Sec. 1.2.3) on which the economic analysis (Sec. 3.3) is based. Together with known

[6] `http://dbpedia.org` (accessed 06.01.2010)
[7] `http://www.mediawiki.org/wiki/Extension:Semantic_Forms` (accessed 06.01.2010)

literature a comprehensive requirements catalog for PKM tools and the conceptual model behind them has been created (Sec. 3.7).

As a solution, the *Conceptual Data Structures* model has been presented. It consists of a flexible yet simple data-model ($\mathfrak{D}$), a relation ontology ($\mathfrak{R}$), and an interchange format for structured text (STIF). The data-model allows representing and using (via queries, browsing, and transformations) knowledge cues in a uniform fashion that are in different degrees of formalisation.

The relation ontology ($\mathfrak{R}$) has been presented as a top-level ontology for personal knowledge models.

Authoring in structured text has been shown as a means to cut down the costs of externalising structured artefacts, which in turn lowers the cost of retrieval.

The conceptual model of CDS with all three parts ($\mathfrak{D}$, $\mathfrak{R}$, and STIF) has been implemented in a Java API and reference implementation. Based on this API, the *Hypertext Knowledge Workbench* has been created as a generic CDS editor. The CDS API has been changed and adapted four times, shaped by its usage in NEPOMUK, HKW, QuiKey, and iMapping. An additional implementation in JavaScript has been created for HKW– but the reasoning engine in JavaScript turned out to be too slow. The presented Java implementation is stable, reliable and has good performance. The CDS model and the corresponding implementations provide a robust basis for further personal semantic knowledge management tools – exemplified by QuiKey and iMapping. Both the CDS model as well as the API and its reference implementation separate the data-model $\mathfrak{D}$ from the RELATION ontology $\mathfrak{R}$. This allows re-using both parts independently and adapting the RELATION ontology to other needs, mostly by extending it.

The developed CDS data model and the relation ontology fulfill almost all of the 31 requirements from the requirement catalogue. Only versioning has not been tackled yet.

An evaluation has shown that CDS has been perceived to be as easy to learn as comparable models.

The CDS tools (HKW plus two more developed by third parties) have been evaluated in three different scenarios and have been compared to a mature, state-of-the art semantic wiki. This 125-hour-evaluation revealed that (a) the RELATIONS from $\mathfrak{R}$ have been used in user-created STATEMENTS, (b) users have created their own RELATIONS, and (c) many of the user-created RELATIONS are sub-RELATIONS of RELATIONS from $\mathfrak{R}$. The still prototypical CDS Tools have an interaction efficiency comparable to the mature semantic modelling tool Semantic Media Wiki (SMW), which has been confirmed by direct user ratings and by an analysis of interaction traces with both tools.

Study participants created significantly more non-trivial triples in CDS Tools, compared to SMW. Furthermore, CDS tools have successfully been used to represent and use knowledge in different degrees of formality.

Given the good experiences in the user studies and the fact that CDS has been derived as a unified model of the conceptual models of tools used for

PKM tasks, CDS is a good candidate for a future exchange format among PKM tools. Together with some required extensions – outlined in *Future Work* – CDS is a suitable formalism for exchange of PKM models among people.

The process of *stepwise formalisation* has been motivated in Sec. 3.2.2, described in Sec. 4.4.1, and demonstrated in a user interface in Sec. 5.2.1.

The key feature of CDS models is their ability to represent informal knowledge, formal knowledge and knowledge in varying degrees of formalisation, in one single model. This allows gradually adding more structure and formality, as desired, always under the assumption that more structure and more formality allow better knowledge cue retrieval and usage.Furthermore, CDS-based tools can be used across the whole life-cycle of PKM.

Vision *As outlined in the motivation section and described in (Völkel, 2007a), there is a vision behind the work on CDS. This vision is inspired by* MEMEX *from Bush (1945) and* Augment *from Engelbart (1963).*

PKM and knowledge work in general will move away from the tight coupling of "one formalism to one tool" and instead move to generic formalisms that can be edited in a number of tools. Such generic formalisms allow new scientific or personal insights as well as providing cheaper ways for storing, retrieving and transforming knowledge.

In an ideal future world, a broad number of PKM tools (or simply tools used for PKM tasks) can export their data as CDS models and import from CDS models. A tool that does not understand the semantics of some RELATIONS simply falls back to process it as the next super-RELATION that it can handle. As an example, the CDS model exported by a tagging-aware application might use the RELATION [has tag] to export the assignment from a file to a tag. The importing application might not understand [has tag] but its super-RELATION [has annotation]. So it renders the tag assignments as annotations (e. g., as call-outs in a mind-map). The user can edit the tag names here. Later, the knowledge model can be re-opened in the first application with correctly renamed tags, which still used the [has tag] RELATION. Of course, newly added tags would become merely annotations as the creating should not create STATEMENTS with RELATIONS which's consequences it cannot understand.

Documents, presentations and outlines can be generated from personal knowledge models[8].

In a (huge) cultural step, documents are no longer exchanged. Instead, knowledge models are published and interlinked in a fine-granular manner to other people's work. Academia moves away from documents and focuses more on distributed, formal argumentation. Schools start to teach modelling with the same rigour as reading, writing and math. The global population shifts their cognitive limits, and old problems are tackled with new solutions.

[8] The mapping from CDS models to (STIF-) documents has been presented in Sec. 4.3.3.

A. Appendix

A.1. Foundations: Embedding RDF in HTML

RDF is a meta-data standard described in Section 2.6. There are two approaches allowing to embed RDF data in HTML pages: RDFa[1] and eRDF[2]. Table A.1 contains the main points of difference between the two approaches.

Feature or Requirement	eRDF	RDFa
DRY (Don't Repeat Yourself)	yes	mostly
HTML4 / XHTML 1.0 validity	yes	no
Explicit syntactic means for arbitrary resource descriptions	no	yes
Supported by the W3C	partly	yes
Follow DCMI guidelines	yes	no
Support for not just plain literals (e. g., typed dates, floats, or markup).	no	yes
Triple bloat prevention (only actively marked-up information leads to triples)	yes	no
Possible integration in namespaced (non-HTML) XML languages.	no	yes
Tidy-safety (Cleaning up the page will never alter the embedded semantics)	yes	no
Explicit support for blank nodes.	no	yes
Compact syntax, based on existing HTML semantics like the address tag or rel/rev/class attributes.	mostly	partly
Inclusion of newly evolving publishing patterns (e. g., rel="nofollow").	no	partly

Table A.1.: Comparing eRDF and RDFa

The data is an excerpt of an analysis by Benjamin Nowack.
Published on February 12 2007 at `http://bnode.org/blog/2007/02/12/comparison-of-microformats-erdf-and-rdfa` (accessed 06.01.2010).

A.2. Analysis: PKM survey

These questions have been asked in the PKM online survey:

[1] `http://www.w3.org/TR/xhtml-rdfa-primer/` (accessed 06.01.2010)
[2] `http://research.talis.com/2005/erdf/wiki` (accessed 06.01.2010)

1. What should a Personal Knowledge Management (PKM) tool do for you? Please give examples of tasks where a PKM tool could help you to be more productive.

2. Which tools do you use to take personal notes? i.e. notes that are intended to be read only by you. Try to enumerate them all.

3. In one week, how many personal items do you write down?

   Of these personal notes, how many are still relevant (in percent) after ...

   a) ... one day?

   b) ... one week?

   c) ... one month?

   d) ... one year?

4. How much time do you spend ...

   a) *writing down* personal notes in one week?

   b) *searching for* personal notes in one week?

   c) *organising*, *structuring* personal notes in one week?

5. When do you know you have to adapt your personal KM strategy/-tool? E. g., what must happen so you know things go wrong? In other words: Which factors do you monitor yourself to know that your PKM works well enough?

6. For what kind of projects/tasks do you wish you had better PKM support?

7. Which tools do you use today for such tasks?

8. What are the features of PKM tools? Please help us to create a list to compare PKM tools in a matrix style. Which features would you look for? E. g., "full-text search".

A.3. Structured text interchange format

A.3.1. A STIF Wiki Syntax

A part of the CDS reference implementation is a STIF wiki syntax parser which parses the syntax depicted on the left side of the comparison tables and emits the STIF depicted on the right side.

Inline formatting

STIF wiki syntax	Resulting STIF
`*bold*` `__italic__` `<tt>code</tt>` `forced linebreak\\`	`<strong>bold</strong>` `<em>italic</em>` `<code>code</code>` `forced linebreak `

Headings

STIF wiki syntax	Resulting STIF

`= Heading 1`
`== Heading 2`
`=== Heading 3`
`==== Heading 4`
`===== Heading 5`
`====== Heading 6`
	`<h1>Heading 1</h1>`
	`<h2>Heading 2</h2>`
	`<h3>Heading 3</h3>`
	`<h4>Heading 4</h4>`
	`<h5>Heading 5</h5>`
	`<h6>Heading 6</h6>`

Links

STIF wiki syntax	Resulting STIF
`[AAA]`	`<a href="#urn-of-AAA"` `class="stif-internal">AAA</a>`
`[aaa|AAA]`	`<a href="#urn-of-AAA"` `class="stif-internal" >aaa</a>`

`[http://sap.com]`	`<a href="http://sap.com"`
	`class="stif-external"`
	`>http://sap.com</a>`

`[SAP|http://sap.com]`	`<a href="http://sap.com"` `class="stif-external">SAP</a>`

Block-level

STIF wiki syntax	Resulting STIF

`first paragraph`
 	
`second paragraph`
`----`
`{{{pre-`
`formatted block}}}`
	
	`<p>first paragraph</p>`
	`<p>second paragraph</p>`
	`<hr>`
	`<pre>pre-`
	`formatted block</pre>`

Ordered list

STIF wiki syntax	Resulting STIF

`+ first item`
`+ second item`
	`<ol>`
	`  <li>first item</li>`
	`  <li>second item</li>`
	`</ol>`

Unordered list

STIF wiki syntax	Resulting STIF

`* item one`
`* item two`
	`<ul>`
	`  <li>item one</li>`
	`  <li>item two</li>`
	`</ul>`

Definition list

STIF wiki syntax	Resulting STIF

`; Term A`
`: Def A`
`; Term B`
`: Def B`
	`<dl>`
	`  <dt>Term A</dt>`
	`  <dd>Def A</dd>`
	`  <dt>Term B</dt>`
	`  <dd>Def B</dd>`
	`</dl>`

Images

STIF wiki syntax	Resulting STIF
`[http://w3.org/x.gif]`	`<img src="http://w3.org/x.gif" />`

Tables

STIF wiki syntax	Resulting STIF

`!! Header 1.1 !! Header 1.2`
`:: Cell 2.1 :: Cell 2.2`
	`<table>`
	`  <tr>`
	`  <th>Header 1.1</th>`
	`  <th>Header 1.2</th>`
	`  </tr>`
	`  <tr>`
	`  <td>Cell 2.1</td>`
	`  <td>Cell 2.2</td>`
	`  </tr>`
	`</table>`

Mixed lists

<table>
<tr><th>STIF wiki syntax</th><th>Resulting STIF</th></tr>
<tr><td>With indention:

<pre>
  + item one
  + item two
    * subitem A
    * subitem B
  + item three
    ; Term X
    : Definition X
    ; Term Y
    : Definition Y
  + item four
</pre>
Without indention:

<pre>
  + item one
  + item two
  +* subitem A
  +* subitem B
  + item three
  +; Term X
  +: Def X
  +; Term Y
  +: Def Y
  + item four
</pre>
</td><td>Result for both syntaxes:

<pre>

  item one
  item two
  item three

      subitem A
      subitem B

  item four
    <dl>
      <dt>Term X</dt>
      <dd>Def X</dd>
      <dt>Term Y</dt>
      <dd>Def Y</dd>
    </dl>

</pre>
</td></tr>
</table>

A.3.2. STIF Document Type Definition (DTD)

The full Document Type Definition (DTD) for STIF is printed here.

```
<?xml version="1.0" encoding="UTF-8"?>
<!--  This is STIF 1.0 DTD.
      Draft:   2009-02-20
      Revised: 2009-02-22
      Authors: Max Voelkel (dev@xam.de), Andreas Kurz
      Further information about STIF is available at:
          http://semanticweb.org/wiki/STIF       -->

<!-- Imported names -->
<!ENTITY % URI  "CDATA">
<!ENTITY % Text "CDATA">

<!-- Parameter entities -->
<!ENTITY % heading "H1|H2|H3|H4|H5|H6">
<!ENTITY % list "UL | OL">
<!ENTITY % preformatted "PRE">

<!-- Text markup -->
<!ENTITY % special "A|IMG|BR">
<!ENTITY % phrase  "EM|STRONG|CODE">
```

```
<!ENTITY % inline  "#PCDATA | %phrase; | %special;">

<!ELEMENT EM     (%inline;)*>
<!ELEMENT STRONG (%inline;)*>
<!ELEMENT CODE   (%inline;)*>

<!ELEMENT BR EMPTY>
```

HTML has two basic content models: `%inline;` for character level elements and text strings and `%block;` for block-like elements, e. g., paragraphs and lists.

```
<!ENTITY % block "P | %heading; | %list; |
    %preformatted; | DL | HR | TABLE">
<!ENTITY % flow "%inline;| %block;">

<!-- Document body -->
<!ELEMENT STIF (%flow;)*>

<!-- Paragraphs -->
<!ELEMENT P (%inline;)*>

<!-- Anchor element -->
<!ELEMENT A (%inline;)*>
<!ATTLIST A
  href        %URI;          #IMPLIED
  class        CDATA         #IMPLIED
>

<!-- Images -->
<!ELEMENT IMG EMPTY>
<!ATTLIST IMG
  src         %URI;          #REQUIRED
  alt         %Text;         #REQUIRED
>

<!-- Horizontal rule -->
<!ELEMENT HR EMPTY>
```

There are six levels of headings from H1 (the most important) to H6 (the least important).

```
<!-- Headings -->
<!ELEMENT H1 (%inline;)*>
<!ELEMENT H2 (%inline;)*>
<!ELEMENT H3 (%inline;)*>
<!ELEMENT H4 (%inline;)*>
<!ELEMENT H5 (%inline;)*>
<!ELEMENT H6 (%inline;)*>
```

```
<!-- Preformatted text -->
<!ELEMENT PRE (%inline;)*>
```

DL are definition lists with DT for term and DD for its definition.

```
<!-- Lists -->
<!ELEMENT DL (DT|DD)+>
<!ELEMENT DT (%inline;)*>
<!ELEMENT DD (%flow;)*>
<!ELEMENT OL (LI)+>
<!ELEMENT UL (LI)+>
<!ELEMENT LI (%flow;)*>

<!-- Tables -->
<!ELEMENT TABLE (TR)+>
<!ELEMENT TR (TH|TD)+>
<!ELEMENT TH (%flow;)*>
<!ELEMENT TD (%flow;)*>
```

A.4. XML-based persistence format for CDS

The persistence component of the CDS reference implementation uses the open source library *XStream* to serialise Java domains models to XML and back again. This is a shortened sample file of a CDS file. Note that the file shows some additional experimental features (deletable, changeable) that are not describe in this thesis.

```
<org.semanticdesktop.swecr.model.memory.xml.XModel>
  <contentItems class="linked-list">
    <contentitem>
      <uri>urn:cds:fff7157d-5e08-41e3-adc6-9872e1d5ca81</uri>
      <readonly>false</readonly>
      <deletable>true</deletable>
      <changeDate>1252413250387</changeDate>
      <creationDate>0</creationDate>
      <authorURI>http://imapping.info#author</authorURI>
      <binary>false</binary>
      <content>
        &lt;p&gt;select it and press backspace&lt;/p&gt;
      </content>
      <mimetype class="org.semanticdesktop.binstore.MimeType">
        <mimeType>application/stif+xml</mimeType>
      </mimetype>
    </contentitem>
    ... Further <contentitem>...
  </contentItems>
  <nameItems class="linked-list">
    <nameitem>
      <uri>urn:cds:005053c4-e0de-4d94-9854-8415e9569c58</uri>
      <readonly>false</readonly>
      <deletable>true</deletable>
      <changeDate>1252400908183</changeDate>
      <creationDate>0</creationDate>
      <authorURI>http://imapping.info#author</authorURI>
      <name>Lernförderlichkeit</name>
```

```
    </nameitem>
    ... Further <nameitem>...
  </nameItems>
  <relations class="linked-list">
    <relation>
      <uri>urn:xam.de:20090903-13.45.34.707-1</uri>
      <readonly>false</readonly>
      <deletable>true</deletable>
      <changeDate>1251985534707</changeDate>
      <creationDate>0</creationDate>
      <authorURI>http://imapping.info#author</authorURI>
      <name>knows-inverse</name>
      <inverseUri>urn:xam.de:20090903-13.45.34.707-0</inverseUri>
      <inverseName>knows</inverseName>
      <inverseCreationDate>1251985534707</inverseCreationDate>
      <inverseChangeDate>1251985534707</inverseChangeDate>
    </relation>
    ... Further <relation>...
  </relations>
  <statements class="linked-list">
    <statement>
      <uri>urn:xam.de:20090906-10.36.33.093-0</uri>
      <readonly>false</readonly>
      <deletable>true</deletable>
      <changeDate>1252233393093</changeDate>
      <creationDate>0</creationDate>
      <authorURI>http://imapping.info#author</authorURI>
      <binary>false</binary>
      <s>urn:cds:c09e6f98-5d0e-4ff6-93ed-494535e54cbc</s>
      <p>http://www.semanticdesktop.org/ontologies/2007/09/
01/cds#hasDetail</p>
      <o>urn:cds:df7dcdf5-dd86-4783-abb1-b7eee9368490</o>
    </statement>
    ... Further <statement>...
  </statements>
  <triples class="linked-list"/>
</org.semanticdesktop.swecr.model.memory.xml.XModel>
```

A.5. Evaluation

This appendix contains a detailed analysis of fulfilled requirements, the detailed task descriptions handed out to participants, as well as data tables about the study results.

A.5.1. Fulfilment of requirements

This appendix details the evaluation of the requirements, which is presented in the compact Table 6.1.

1 This has simply not yet been implemented in the tools. QuiKey demonstrates already how queries can be executed on-demand. Adding an automatic function to run all queries periodically and notify the user can obviously be implemented. This functionality is planned for further iMapping versions.

2 The CDS model has been constructed by generalising other data-models.

3 Each of the tools has individual ways for fast entry.

4 In $\mathfrak{D}$, the user can create ContentItems which do not imply other facts in the model. In $\mathfrak{R}$, the user can use [is related to], which is the *top-relation*. It does not imply any other Triple. STIF can be used in a plain-text fashion, if the user simply uses no special wiki formatting syntax such as "===".

5 In $\mathfrak{D}$ and $\mathfrak{R}$, a Statement can be used with a Relation for which inference has been defined. STIF contains syntax elements which results in semantic statements. Each tool has a way to enter formal statements.

6 A relation ontology ($\mathfrak{R}$) can never have "different granularity". $\mathfrak{D}$ supports very small up to very large ContentItems, together with no, few or many STIF formatting instructions. HKW, QuiKey and iMapping support this.

7 All Item in $\mathfrak{D}$ are addressable. Different from data models such as XML, RDF and OWL even statements between entities can be addressed.

8 The tools could use the formal knowledge even better, e. g., by displaying icons to represent the different inferred types of items. The queries could use the informal text and structures better, eg letting a user search for a term in all text parts that have an italic formatting.

9 In $\mathfrak{D}$, a user can shift from using a ContentItem to using a NameItem with the same content. A user can explicate her knowledge as the content of items in unstructured form, or structured via wiki-syntax (see 4.3.2). The migration from unstructured text to wiki syntax is smooth. Knowledge can also be represented as formal statements. Sec. 4.3.2 describes also how such statements can be derived from wiki

syntax to smooth the transition. In $\mathfrak{R}$, stepwise formalisation is a core concept, as explained in Sec. 3.2.2. HKW and iMapping allow refining STATEMENTS with more specific RELATIONS. QuiKey allows only deleting STATEMENTS and re-creating them in refined ways.

10 Refactoring of models can only be supported by tools operating on models.

Refactoring of content is probably a key strength of mind maps and a key problem of wikis. iMapping allows refactoring of the context-detail-hierarchy via drag-and-drop. None of the tools has support for batch operations. Renaming of concepts (NAMEITEMS) is possible in all tools.

11 The CDS model has no special support for versioning. The CDS API, however, uses an event listening pattern, so that all events ever occurring in a model (creating, changing and deleting items) can be monitored and re-played.

12 $\mathfrak{D}$ tracks for each ITEM the creation date, change date and author. This date is exposed in HKW if the user hovers over an item. Additionally, the user can use formal STATEMENTS to express even richer context models.

13 This requirement for tools is poorly addressed in the prototype implementations.

14 This requirements has been evaluated in Sec. 6.2 and 6.3.

15 In $\mathfrak{D}$, any items linked to the same source or target are grouped together in a way. As an example, linking a, b, and c to the same target t via any kind of relation, e. g., p: (a, p, t), (b, p, t), and (c, p, t). Then the query $(*, p, t)$ returns the group of items.

The RELATIONS [has context] and [has detail] have been designed for grouping items, optionally in a nested way. iMapping uses these RELATIONS to group ITEMS visually.

16 The relation ontology provides [has part] to represent containment relationship. Section 4.3.3 describes how STIF documents are encoded in $\mathfrak{D}$ with [has part].

17 The concept of NAMEITEMS in $\mathfrak{D}$ maps strings to ITEMS. A user can link a NAMEITEM via STATEMENTS to other ITEMS. By doing so, she can make each ITEM addressable with a name. Different from wikis, neither $\mathfrak{D}$ nor the CDS-based tools *require* a user to use names or NAMEITEMS at all. Users can convert existing CONTENTITEMS to NAMEITEMS and vice versa.

18 The RELATION [is alias for] in $\mathfrak{R}$ is provided, but the semantics have not been implemented yet in the prototypes. Therefore, the alias-semantics can only be stated in all tools, but there is no special support for retrieval.

19 The Relation [comes after] and [comes before] let the user express partial or complete ordering among Items in a mathematical sense. HKW uses these Statements to determine the order in which detail-Items are rendered. iMapping and QuiKey let the user state order via their generic Statement manipulation abilities, but provide no special support for rendering order. STIF uses the [comes after] and [comes before] to represent the order of document parts when parsing wiki syntax.

20 In $\mathfrak{D}$, links are modelled as Statements. Any kind of resource can be linked with any other resource (e. g., ideas, persons, files, issues, tags, types, ...). The direction and type of the link can be specified.

$\mathfrak{R}$ has a special Relation to represent generic hyperlinks ([links to]). STIF supports normal links to NameItems as well as semantic links to NameItems. Links to external web resources are supported as well. HKW supports all these link types.

21 $\mathfrak{R}$ provides the Relation [has detail] and [has context] for nested grouping. iMapping has been designed around the idea of hierarchical browsing. HKW and QuiKey provide no additional support for this Relation beyond their generic abilities.

All kinds of hierarchies, including type-hierarchies are modelled in CDS as [has type] (inverse: [has instance]) or sub-relations of these relations. This allows generic tools to browse all kinds of hierarchies.

22 This feature is well implemented in iMapping. HKW supports only three levels of detail at once, but could be extended to support more. QuiKey is a concept that focuses on individual Items, especially Statements.

23 $\mathfrak{R}$ provides the Relation [annotates] to represent all kinds of annotations. HKW dedicates a part of the user interface to show and author annotations. iMapping and QuiKey have only generic annotation support.

24 $\mathfrak{R}$ provides [has tag] for tagging. All tools support tagging in a generic way.

25 Categories can be seen as hierarchical tags or names. $\mathfrak{R}$ has [has detail], this is enough to model nested categories and their members. If reasoning is desired, [is subtype of] can be used to nest category names. All tools can use this.

In CDS, each item can be used as another's items type via [has type]. This allows adding to each item a type and is required to enable full meta-modelling, i. e., assigning types to types (→ Req. 31 meta-modelling)

26 Queries operate on the data model ($\mathfrak{D}$) as defined in Sec. 4.1.3. Queries take semantics of Statements into account. Only the QuiKey prototype allows defining and executing queries.

27 All STATEMENTS in $\mathfrak{D}$ are browsable in all three CDS-based tools. STIF is rendered as hyper-linked HTML. The relation ontology ($\mathfrak{R}$) is modelled in $\mathfrak{D}$ and can therefore be browsed as well.

28 Inverse RELATIONS are mandatory for each RELATION in $\mathfrak{D}$. HKW and QuiKey allow editing and using inverse RELATIONS. iMapping can display inverse RELATIONS but has no support for renaming inverse RELATIONS. Due to the visual concept of iMapping, inverse RELATIONS are not required.

29 $\mathfrak{D}$ is a domain-free model, allowing to represent any data with or without a schema. $\mathfrak{R}$ is rather generic and extensible for cases where it turns out to be to restricted. All three tools operator on the domain-free model. Hence a user is not constrained by a schema. Users can create any number and kind of ITEMS and relation types as they need.

30 Any ITEM can be referenced, but the current implementation of STIF has support for rendering linked ITEMS inline as part of the main text.

31 As all ITEMS are addressable and STATEMENTS can link any ITEMS, full meta-modelling is provided. Annotating and linking STATEMENTS is only implemented in HKW.

A.5.2. Study task descriptions

Scenario A

Vorab
* Gehe zur CDS Tools Homepage http://cdstools.xam.de
* Installiere von dort die CDS-Tools
* Lies die CDS Tools Homepage genau durch. Besonders wichtig ist
`http://semanticweb.org/wiki/CDS_Tools#Relation_Types_-_all_CDS_tools`
Lies auch folgende Seiten:
** `http://semanticweb.org/wiki/CDS_Tools/Wiki_Syntax`
** `http://semanticweb.org/wiki/HKW`
** `http://semanticweb.org/wiki/QuiKey/Documentation`
iMapping
* Erforsche und Lese die “Welcome-Map” in iMapping
* Erforsche alle Menü-Optionen.
HKW
* Starte HKW aus iMapping
* Spiele damit herum und teste auch die Wiki-Syntax
QuiKey
* Starte QuiKey aus iMapping
* Spiele damit herum und teste auch Abfragen

Aufgabe 1a (Dauer: ca. 1h)

Verwende iMapping um den Inhalt von `http://de.wikipedia.org/wiki/Wei%C3%9Fer_Hai` in eine Faktensammlung umzusetzen. Stelle Dir dabei vor, Du seist “Linda”, eine Biologin, welche den Weißen Hai erforscht. Erstelle ihre persönlichen Notizen.

Sie hat bisher kaum Vorwissen zum Weißen Hai.

Speichere deine Wissensbasis in iMapping unter <Datum>-<DeinName>-1a.iMap ab. Also z.b. “20090713-voelkel-1a.iMap”. Sende die fertige Datei per Email an cdstools@xam.de.

Aufgabe 1b (Dauer: ca. 3-5h)

* Wechsele nun zu HKW (Ctrl-B) und löse die nächsten Schritte in diesem Werkzeug.
* Füge Fakten über den Riesenhai http://de.wikipedia.org/wiki/Riesenhai hinzu
* Füge Informationen über Bücher und Kinofilme zum Weißen Hai hinzu
* Schreibe 5 Fragen auf, die du mit Hilfe deiner Faktensammlung beantworten kannst.
** Dabei ist es egal, ob du die Fakten als Text, formatierten Text mit Wikisyntax oder als Statements (=Links) anlegst.
* Schreibe 5 Fragen über den weißen Hai auf, die du mit deinem Wissensmodell nicht beantworten kannst.

Speichere deine Wissensbasis in iMapping unter <Datum>-<DeinName>-1b.iMap ab. Also z.b. “20090713-voelkel-1b.iMap”. Sende die fertige Datei an cdstools@xam.de.

Scenario B

* Dauer: 5h
* Thema: “Gewürze”
* Sprache: Entscheide selbst, ob du die Aufgabe auf Deutsch oder Englisch lösen möchtest.

SMW:

* Werkzeug: Semantic MediaWiki (login erhälst du in separater mail)

Vorab
* Mache dich mit den Möglichkeiten der ASK-Abfragen vertraut (ca. 20 Minuten)

Beachte
* Achte darauf, Teile der Wissensbasis so weit zu strukturieren, dass du ASK-Fragen stellen kannst.

CDS Tools:

* Werkzeug: CDS Tools

Vorab
* Mache dich nochmals mit den Abfragemöglichkeiten in QuiKey vertraut. (ca 20 Minuten)
** `http://semanticweb.org/wiki/QuiKey/Documentation`

Beachte
* Achte darauf, Teile der Wissensbasis so weit zu strukturieren, dass du z.B. in QuiKey Abfragen stellen kannst.
* Speichere deine Wissensbasis in iMapping unter <Datum>-<DeinName>-3.iMap ab.
** Also z.b. “20090713-voelkel-3.iMap”. Sende die fertige Datei per Email an cdstools@xam.de.

Preis:

* Es wird später einen Preis (2 Kinokarten + 1 Popcorn) geben für denjenigen, der konkrete Fragen mit Hilfe seiner Wissensbasen am Besten beantworten kann.
* Bei der Abfrage entscheidet Max wer wann welche Wissensbasis benutzen darf (es wird fair gewechselt). SMW und CDS Wissensbasen werden dabei voneinander getrennt - also musst Du in beiden Tools geschickt modellieren (gut, aber nicht zuu lange brauchen), um den begehrten Preis zu gewinnen.

Szenario:

Du bist ein Koch in einem 4-Sterne Restaurant. Deine Gäste erwarten von Dir ständig neue Kreationen. Du kennst die Praxis und verwendest viele Gewürze. Nur über die Gewürze selbst weißt Du bisher wenig. Um das zu ändern, startest du eine fünfstündige Recherche im Internet. Dein Ziel ist der Aufbau eines Wissensbasis über Gewürze, insbe-

sondere jegliche Art von Zusammenhang zwischen Gewürzen.
* Welche stammen von ähnlichen Pflanzen ab?
* Welche schmecken ähnlich?
* Welche passen gut zusammen?
* Welche werden ähnlich erzeugt?
* Welche werden für ähnliche Gerichte verwendet?
* Welche werden in ähnlichen Kulturen verwendet?
Versuche dabei insgesamt, möglichst bekannte Gewürze bevorzugt zu betrachten. Füge zur Wissensbasis alles hinzu, was dir spannend oder relevant vorkommt.

In ca. 4 Wochen wirst du unter Zeitdruck das neue Überraschungsmenü entwerfen. Dann bgracusht du eine gute Datenbasis um zu entscheiden, welches Gewürz du durch welches probehalber ersetzen könntest. Auch deine Kochprüfung über Gewürze wird die einiges abverlangen. Achte darauf, dass Du vor allem Zusammenhänge zwischen Gewürzen findest.

Und denk daran: Ein reines copy & paste hilft dir später wenig, weil die die Fragen unter Zeitdruck beantworten musst, da ist zuviel Text hinderlich.

Aufgabe 3b:
* Stelle 5 Fragen, die Du mit deiner Wissensbasis gut beantworten kannst
* Stelle 5 Fragen, die Du mit deiner Wissensbasis nur umständlich beantworten kannst (aber überhaupt)

Sende die Fragen an cdstools@xam.de

Second round assignment After assignment 2 the names of the most often used spices have been collected and were proposed to use for assignment 3. The emails in assignment 3 had this additional text:

Zusatzhinweis:
Betrachte als Gewürze z.B. Pfeffer, Ingwer, Öl, Muskat, Chili, Vanille, Nelke, Minze, Szechuan-Pfeffer, Estragon, Rosmarin, Thymian, Oregano, Majoran, Petersilie, Zucker, Salz, Essig, und Zitrone.

Scenario C

SMW:
* Dauer: 5 Stunden
* Werkzeug: SMW
** Bitte die Inhalte in SMW modellieren. Dabei darauf achten, das manches u.U. auch abfragbar ist, aber ohne zu viel Zeit für die Modellierung zu verwenden, da du den gesamten Text modellieren musst.
** Dein Wiki-Zugang bekommst Du mit einer weiteren Email.

CDS Tools:
* Dauer: 5 Stunden
* Werkzeug: Hauptsächlich HKW, ergänzend QuiKey und iMapping
** Bitte die Inhalte in HKW modellieren. Dabei darauf achten, das
*** Bestehende Relationen wiederverwendet werden, z.B. "has type", "is subtype of".

Vorab: Achte drauf, dass deine CDS Tools auf dem aktuellen Stand sind, vor allem HKW.
Siehe hierzu
* `http://semanticweb.org/wiki/CDS_Tools#Running_and_Updates`
* `http://semanticweb.org/wiki/HKW#If_HKW_hangs`

Thema: "Soziale Netzwerke"
* Entscheide selbst, ob du die Aufgabe auf Deutsch oder Englisch lösen möchtest.

Preis:
* Es wird später einen Preis (2 Kinokarten + 1 Popcorn) geben für denjenigen, der konkrete Fragen mit Hilfe seiner Wissensbasis am Besten beantworten kann. Die Befragung wird einzeln im September stattfinden. Du hast dabei wieder Zugriff auf deine modellierten Wissensbasen (CDS und SMW, allerdings nicht gleichzeitig). Es ist daher klug, sich im Voraus schonmal mit den Abfrage-Möglichkeiten (in HKW, iMapping und QuiKey) vertraut zu machen.

SMW:
Aus Zeitgründen solltest du erwägen jetzt bereits ASK-Fragen anzulegen (siehe `http://semantic-mediawiki.org/wiki/Help:Inline_queries`).

CDS Tools:
Aus Zeitgründen solltest du erwägen jetzt bereits Queries in QuiKey anzulegen (das geht nicht in HKW).

Dann sparst Du in der Abfrage-Runde Zeit, allerdings geht dies zu Lasten von mehr Wissen in der Wissensbasis.
Dein Modell wird nur von Dir verwendet werden.

Die Aufgabe:
Du bist Chef einer Berliner Firma mit fünfzehn Angestellten. Zum zehnten Jubiläum organisierst du eine Feier, auf der auf zukünftige Projekte besprochen werden sollen, die für deine Firma essentiell sind.

Einige Wochen vorher plaudert einer deiner Mitarbeiter aus dem Nähkästchen.
* Seine Informationen sind in 8 Bilddateien enthalten.
* Verwende eigene Formulierungen.
* Der Text ist teilweise recht wirr. Man muss ziemlich genau lesen muss, um nichts wichtiges zu verpassen. Das ist beabsichtigt.

Einige der genannten Personen arbeiten heute in deiner Firma, andere bei Kunden und

in den Firmen möglicher Kunden. Ohne näher darauf einzugehen, wer in deiner Firma arbeitet: Du musst Dir gut überlegen, wen Du auf der Feier nebeneinander setzt, damit es keinen Eklat gibt und über wen du welchen potentiellen Kunden ansprechen kannst. Dazu musst Du die folgenden Fakten analysieren:
* Wer kennt sich und woher?
** Wer ist mit wem verwandt?
** Wer wohnte mit wem zusammen?
* Wie stehen die Menschen zueinander? Positiv, verfeindet, anders ... ?
** Haben sich X und Y schon einmal gestritten?
* Wer hat welchen Hintergrund (Ausbildung, Beruf)?
* Wer hat wo gearbeitet? – wer, wenn auch nicht unbedingt gleichzeitig – in der gleichen Firma gearbeitet hat, erfährt möglicherweise indirekt etwas über die anderen Personen.

Aus dem Text kann man sehr viele "X kennt Y"-Beziehungen rausziehen, aber die Art von Beziehung (also "Wie steht X zu Y") ist schwieriger.
Es geht also nicht Darum, sich alle Details zu merken, sondern grob das soziale Netzwerk, um damit Entscheidungen zu treffen. Es geht dabei nicht nur um die Jubiläumsfeier.

Bevor du anfängst zu modellieren:
* Stelle fünf Fragen, die für Dich als Firmenchef relevant wären
* Denke daran, bestehende Relationen wiederzuverwenden. Das ist wichtig für die Evaluation.

Teilaufgabe:
* Stelle fünf Fragen, die für Dich als Firmenchef relevant wären (die du dir oben schon gestellt hast)
* Stelle fünf weitere Fragen, die Du mit deiner Wissensbasis schnell beantworten kannst
* Stelle fünf weitere Fragen, die Du mit deiner Wissensbasis nicht schnell, aber überhaupt beantworten kannst

Sende diese 15 Fragen per EMail an cdstools@xam.de.

Speichere deine Wissensbasis in iMapping, also z.b. "20090713-voelkel-4.iMap‘". Sende die fertige Datei per Email an cdstools@xam.de.

The input text for the social network modelling tasks has been taken from the story plot of a popular German TV series that was aired since 1992. A German fan wiki[3] collects episode descriptions. The wiki source text from episode 1 to 31 has been copied together into one text document. This document has been converted into eight bitmap image files. The image files haven been provided by email to the study participants. Below is the full text of these image files. Spelling errors have been left in, as they have been presented to participants.

[3] Maintained by Roman Allenstein at `http://gzsz-wiki.de`. Text used with permission.

Input text Clemens kommt mit Vera aus dem Urlaub. Vera ruft ihren Sohn Heiko an, um Bescheid zu sagen, dass sie überraschend wieder da sind. Dieser hatte sich schon auf ein Woche sturmfrei gefreut und liegt gerade mit seinder Freundin Tina im Bett, als er den Anruf erhält. Heiko macht sich auf den Weg zur Schule wo er auf Elke und Peter Becker, seine Mitschüler, trifft. Außerdem besucht auch Patrick Graf diese Schule, dieser plant allerdings die Schule kurz vor dem Abitur zu verlassen. Er möchte nämlich in 3 Monaten mit seinem Vater nach Amerika gehen. Derweil taucht Tina in der Praxis ihres Bruders, Dr. Frank Ullrich, auf. Im Wartezimmer sitzt Elisabeth Meinhart, die auf ihre ehemalige Schülerin Marina trifft, welche jetzt in einem Café jobbt. Dr. Ullrich teilt Elisabeth mit, dass sie sehr krank ist und, dass es besser wäre, wenn sie aufhören würde zu unterrichten. Sie sagt, dass die Schule ihr einziger Lebensinnhalt ist, aber Dr. Ullrich besteht darauf, dass sie sich schonen muss. Er fragt sie nach ihrem Kind, weil er bei einer Untersuchung die Narbe des Kaiserschnitts gesehen hatte. Daraufhin behauptet Elisabeth, dass das Kind bei der Geburt gestorben sei. Sie verspricht Frank, dass sie nicht in die Schule zurückkehren wird. Sie will ein neues Leben beginnen. Clemens sagt seinem Sohn Heiko, dass er und Vera sich scheiden lassen wollen. Währenddessen schreibt Elisabeth ihren Abschiedsbrief und versucht sich das Leben zu nehmen.

Bei den Richters ist Familienkrach angesagt. Vater Clemens packt noch am gleichen Abend seine Koffer und verlässt das Haus. Derweil macht sich Peter Becker Sorgen um seine Lehrerin Elisabeth Meinhart, die nicht ans Telefon geht. Als er sie zu Hause besuchen will, trifft vor der Tür Elke Opitz. Sie sehen Licht und nehmen Gasgeruch wahr. Sie finden Elisabeth bewusstlos auf dem Boden und können noch rechtzeitig Hilfe holen. Patrick Graf stellt sich bei Clemens Richter in der Agentur Löpelmann vor. Er will, bevor er mit seinem Vater nach Amerika geht, Berufserfahrung sammeln. Clemens sagt, dass er in der Poststelle der Werbeagentur anfangen kann. Claudia, Clemens Sekretärin, betritt sein Büro. Die beiden haben eine Affäre. Claudia drängt Clemens den Anwalt, wegen seiner Scheidung mit Vera, endlich anzurufen. Währenddessen liest Vera ihrem Sohn Heiko todtraurig einen 15 Jahre alten Liebesbrief von Clemens vor. Peter Becker wird von seinem Vater (Erwin Becker) verdächtigt 50 Mark aus der Spardose gestohlen zu haben. Peter streitet dies allerdings ab. Darauf hin schlägt ihn sein Vater, während seine Mutter (Gerda Becker) in der Tür steht und nichts unternimmt um ihrem Sohn zu helfen.

Patrick verspricht Tina, sie durch die Connections seines Vaters zu einem Top-Model zu machen. Heiko Richter kümmert sich um seine Mutter, die Depressionen hat. Tina taucht auf und wenig später steht Peter mit blutüberströmtem Gesicht vor der Tür. Völlig außer sich erzählt er, dass er sich mit seinem Vater geschlagen und ihn wahrscheinlich umgebracht habe. Nachdem Peter bei Dr. Frank Ullrich verarztet wurde und Heiko den Beckers einen Besuch abgestattet hat, zieht Peter für ein paar Tage ins Haus der Richters ein.

Elke Opitz trifft ihre ehemalige Mitschülerin Julia Backhoff, die bei einem Hostessen-Service arbeitet. Als Lilo Gottschick, Julias Chefin, dazukommt,

prophezeit sie Elke, dass sie in weniger als zwei Wochen ebenfalls bei ihr anfangen wird. Vera Richter hat sich beim Joggen den Knöchel verstaucht. Peter, der sich im Garten nützlich gemacht hat, massiert die lädierte Stelle. Zwischen den beiden knistert es gewaltig. Vera allerdings will, dass Peter wieder auszieht.

Am Frühstückstisch der Richters sagt Peter Becker, dass er ausziehen will. Heiko ist dagegen und streitet sich mit seiner Mutter Vera darüber. Da taucht Elisabeth Meinhart auf und will Peter sprechen. Sie beschuldigt ihn, ihren Abschiedsbrief an sich genommen zu haben, als er in ihrer Wohnung war. Er wird sarkastisch und meint, dann könne er sie ja damit erpressen. Tief verletzt geht Frau Meinhart. Wenig später erfährt sie von Frank, dass er den Brief gefunden hat und ihn in ihrer Wohnung versteckt hat. Elke verabredet sich mit Julia. Sie will Näheres über deren Job wissen.

Elisabeth Meinhart will Kontakt zu Peter Becker aufnehmen, der mittlerweile bei den Richters keine Bleibe mehr hat. Sie bittet ihre Bekannte Frau Gerlach, mit dem Jungen Kontakt aufzunehmen. Als sie bei den Beckers auftaucht, wird sie sofort erkannt, denn Frau Gerlach war es, die damals die illegale Adoption von Peter eingefädelt hatte. Weil Vater Becker eine Erbschaft wittert, will er sich wieder mit seinem Adoptiv-Sohn vertragen. Peter wird ein Job als Animateur in einem Ferienklub in Aussicht gestellt. Ein willkommener Lichtblick in seiner derzeitigen Lebenslage ... Unterdessen trifft sich Heiko mit Claudia, um ihr die Affäre mit Clemens auszureden.

Peter arbeite mit Benni im Yachthafen. Weder die Beckers noch Frau Meinhart und Frau Gerlach wissen, wo sich der Junge aufhält. Erst durch Frank Ulrich erfährt Frau Meinhart, wo Peter steckt. Doch auch er kann nicht wissen, dass die Jugendlichen Probleme mit der Polizei haben. Unterdessen übt Clemens' Sekretärin und Geliebte Claudia zunehmend Druck auf ihren Boss aus. Sie will den gleichen luxuriösen Lebensstandard wie Vera haben.

Clemens Richter erfährt von der Polizei, dass sein Sohn Heiko samt Freunden vorläufig festgenommen ist. Auf der Wache trifft er auf Erwin Becker, der seinen Sohn Peter wegen der vermeintlichen Erbschaft umwirbt. Peter nimmt das Angebot an, nach Hause zurückzukehren. Vera Richter hat derweil im Alkohol einen treuen Freund gefunden. Sie trägt ihre Probleme auf dem Rücken ihres Sohnes Heiko aus. Bei den Köhlers hängt der Haussegen aber ebenfalls schief. A. R. Daniel steigt in dieser Folge bei GZSZ ein. Patrick bekommt bei ihm eine Chance, als sein Assistent zu arbeiten.

Denise Köhler ist verbittert über Bertrams Forderung, sie solle ihr Kind abtreiben lassen. Sie sieht sich vor die Wahl gestellt, entweder ihr Baby oder ihren Mann zu verlieren. Sie beschließt, bei Dr. Frank Ulrich Rat zu suchen. Er will eine Abtreibung verhindern. In der Praxis befindet sich auch der Werbemacher A. R. Daniel, der wegen Rückenschmerzen behandelt werden will. Doch Frank Ulrich muss ein Nierenleiden diagnostizieren und empfiehlt dem Erfolgsmenschen, seinen Lebenswandel radikal umzustellen. Unterdessen muss Peter Becker feststellen, dass beim Einwohnermeldeamt keine Geburtsurkunde von ihm existiert. Er weiß nicht, wer er ist.

Der aufgebrachte Peter Becker stellt seine Eltern zur Rede. Nach hartem

Ringen gestehen sie ein, dass Peter nicht ihr leiblicher Sohn ist. Nachdem Gerda eine Fehlgeburt erlitten hatte, wurde ihnen angeboten, gegen viel Geld einen Jungen bei sich aufzunehmen. Peter ist fest entschlossen, seine wahren Eltern zu finden. Unterdessen gedeiht die Beziehung zwischen Frank Ulrich und Patricia Koller. Franks Schwester Tina ist eifersüchtig.

Peter zieht wieder zu den Richters. Vera ist zwar nicht begeistert, aber sie will sich nicht mit ihrem Sohn Heiko anlegen. Die Freunde machen sich auf die Suche nach Peters wahren Eltern. Der einzige Anhaltspunkt ist die Anschrift einer Frau Wirt, die das Baby-Tauschgeschäft damals angeblich eingefädelt hat. Frank ist mit Patricia zum Golfspielen verabredet. Tina ist sauer, weil er sie deswegen versetzt.

Frank hat Patricia Koller zum Abendessen eingeladen. Tina fühlt sich verpflichtet, das Dinner vorzubereiten. Sie ist beleidigt, als Frank mit fertigem Essen vom Party-Service nach Hause kommt. Auch der Abend verläuft nicht nach Tinas Wünschen. Ihr Freund Heiko unterhält sich mit Patricia und Tina fühlt sich überflüssig. Elke trifft Michael Döring. Sie erklärt ihm, dass sie wegen seiner Nachstellungen die Schule verlassen hat.

Clemens kommt nach Hause und sieht die halb nackte Claudia im Clinch mit Patrick Graf. Er missversteht die Situation und beschimpft seine Freundin als Schlampe. Vera bricht betrunken zusammen. Peter bringt sie ins Bett. Claudia besucht Vera, um von ihr die Zustimmung zu einer schnellen Scheidung zu bekommen. Sie will Clemens endlich ganz für sich haben.

Frank, Heiko und Patricia machen einen Sonntagsausflug. Tina bleibt zu Hause, sie hat ihre Eifersucht gegenüber Franks neuer Freundin noch immer nicht im Griff. Elke trifft sich wieder mit Michael Döring, lässt ihn aber nicht zu weit gehen. Sie hofft, ihn in wenigen Monaten heiraten zu können. Peter besucht Elisabeth. Sie söhnen sich miteinander aus, und die Lehrerin verspricht ihrem ehemaligen Schüler, ihm bei der Jobsuche behilflich zu sein. Bei Ullrichs ist der Haussegen wieder hergestellt. Tina und Patricia verstehen sich gut. In die gemütliche Atmosphäre platzt Denise herein. Sie bittet Frank, ihr Kind abzutreiben.

Clemens will sich von Claudia nicht zu einer Scheidung von seiner Frau zwingen lassen. Vera will sich nur dann von Clemens scheiden lassen, wenn sie das Haus behalten darf. Tina bekommt einen Job als Empfangsdame bei Sports Unlimited - Sport total. Die Firma entpuppt sich als Pornoladen, dessen Chef Herr Smythe ist.

Elisabeth lässt sich widerwillig überreden, mit Jessica und deren Freund Charlie in den Club der einsamen Herzen zu gehen. Heiko, Tina und Peter sind immer noch auf der Suche nach Frau Wirt. Sie müssen erfahren, dass sie vor zwei Jahren gestorben ist.

Elisabeth hat im Tanzclub den Zahnarzt Joseph Mader kennengelernt, der ihr sehr sympathisch ist. Clemens will mit seinem Sohn Heiko ein kameradschaftlich-väterliches Gespräch führen. Der Versuch geht in die Hose. Heiko will sich nichts mehr sagen lassen.

Denise hat ihr Kind von einem Kurpfuscher abtreiben lassen. Es geht ihr gesundheitlich nicht gut. Daraufhin verspricht Bertram, seine Abreise nach Kenia zu verschieben. Elke besucht Michael Döring, weist ihn aber zurück,

als er mehr von ihr will als nur reden. Nachdem Elke weinend das Weite gesucht hat, ruft Döring eine Frau an, die wenig später auf der Matte steht: Es ist Lilo Gottschick.

Lilo Gottschick ist Michael Döring hörig. Sie droht, ihn umzubringen, wenn er jemals eine andere Frau anfassen sollte. Daniel entdeckt Tina auf einem Foto in Clemens' Büro. Er vermittelt ihr einen Vorstellungstermin für einen Werbespot. Vor der Agentur Löpelmann wird Tina von einem Wagen von oben bis unten mit Schmutz bespritzt. Sie giftet den Fahrer an, ohne zu ahnen, dass es der große Löpelmann höchstpersönlich ist.

Oswald Löpelmann verunsichert Tina so, dass sie das Vorsprechen vermasselt. Die ganze Werbekampagne, die von Clemens und Daniel entwickelt wurde, passt dem großen Löpelmann nicht. Als er sich bei Vera und Clemens zum Essen einlädt, müssen die beiden einen ganzen Abend lang trautes Heim spielen. Peter bekommt von Gerd Spengler die Chance, die Cafeteria auf Vordermann zu bringen. Seine erste Tat: Er feuert Marina.

Elisabeth eröffnet Joseph Madigan, den sie im Klub kennengelernt hat, dass sie sich nie mehr fest binden möchte. Sie gibt ihrer Beziehung keine Chance. Joseph muss das akzeptieren. Heiko ist wütend, dass Tina in der Agentur seines Vaters für einen Werbespot vorgesprochen hat, denn er ist der Meinung, Clemens wollte sich bloß bei Tina einschmeicheln. Tina ärgert sich über ihren Freund. Trotzdem besuchen sie zusammen Denise und Bertram Köhler, bei denen die Stimmung auf dem Nullpunkt ist.

Clemens und Vera streiten sich mit ihrem Sohn. Heiko hat keine Lust, für Herrn Löpelmann gute Miene zum bösen Spiel zu machen und ihm eine heile Familienwelt vorzugaukeln. Er droht, Löpelmann beim gemeinsamen Essen alles zu erzählen. Elke ist in Lilo Gottschicks Büro gelandet. Die raffinierte Geschäftsfrau redet mit Engelszungen auf das naive Mädchen ein.

Elke darf sich auf Lilos Kosten neue Klamotten kaufen. Aber wie soll sie Elisabeth und den anderen erklären, woher sie das Geld für die Kleider hat? Lilo beruhigt sie, Elke solle sich tagsüber einen Alibijob suchen. Den findet diese auch, und zwar in Peters Cafeteria. Der will seinen Laden zu einem Renner machen und stellt die Speisekarte auch in der nahe gelegenen Agentur Löpelmann vor.

...

Clemens muss jetzt mit Tina arbeiten, die völlig unerfahren ist. Er ist sauer und behandelt das Mädchen schlecht. Das macht Claudia wütend. Sie ist sowieso gekränkt, weil Clemens die letzte Nacht bei seiner Frau Vera verbracht hat. Patrick lädt Tina und Heiko großzügig zum Abendessen ein, um ihren Einstand ins Berufsleben zu feiern. Im Restaurant kommt es zum Streit.

Patrick bemerkt, dass Daniel entgegen Löpelmanns Anweisungen für einen gewissen Drew McPherson arbeitet. Er versucht, damit eine Beförderung von Daniel zu erpressen. Elke Opitz zieht bei Elisabeth ein. Von Lilo erhält sie einen neuen Auftrag: Elke soll zusammen mit dem Callgirl Julia zwei Kunden zum Abendessen begleiten. Lilo versichert, dass alles ganz seriös zugeht.

Elisabeth liegt nach ihrem Zusammenbruch im Krankenhaus. Dr. Frank Ullrich betreut sie. Elke verbringt den Abend mit Julia Backhoff und den beiden Kunden im Carltons. Als ihre Kollegin mit ihrem Begleiter abzieht, bleibt Elke zurück. Ihr Kunde gibt seinem Unmut über ihr Verhalten lautstark Ausdruck. Am nächsten Tag spricht Elke mit Julia und kündigt ihren Entschluss an, bei Lilo zu kündigen.

Tina ist so wütend über Heikos Einmischung, dass sie noch im Büro mit ihm Schluss macht. Es kommt zu einem lautstarken Streit im Beisein von Patrick Graf. Heiko besucht Elisabeth im Krankenhaus und erzählt ihr von seinen Sorgen. Sie gibt ihm den Rat, Tina am Abend zu Hause zu besuchen.

Frank überrascht seine Schwester Tina mit ihrem Freund Heiko im Bett. Seine Moralpredigt prallt an dem Mädchen ab. Heiko versteht die Welt nicht mehr, als Tina wiederholt, dass zwischen ihnen Schluss sei. Elke Opitz will bei Lilo Gottschick kündigen. Die Agenturchefin geht scheinheilig auf Elkes Wünsche ein, besteht aber darauf, dass die 14-tägige Kündigungsfrist eingehalten wird.

Julia Backhoff hat im Parkcafé Peter kennen gelernt. Ihr wird klar, dass sie mit ihrem Job Schwierigkeiten haben wird, eine normale Beziehung zu führen. Claudia drängt Clemens, seine Frau und Heiko aus dem Haus zu werfen, damit es verkauft werden kann.

Elke erzählt Elisabeth, dass sie Michael Döring liebt und ihn heiraten will, sobald er frei ist. Ihre ehemalige Lehrerin warnt sie davor, sich überstürzt in eine Ehe mit dem älteren Mann zu stürzen. Heiko leidet unter der Trennung von Tina und lässt seine Mutter und Peter Becker seine schlechte Laune spüren. Schließlich bittet er Frank Ullrich um Vermittlung.

A.5.3. Retrieval questions

(1a) Wer waren die Hauptdarsteller im Film “Der weiße Hai”?
(1b) In welchen zwei Filmen spielt der Weisse Hai mit?
(2a) Zu welcher Familie gehört der Weiße Hai?
(2b) Zu welcher Gattung gehört der Weisse Hai?
(3a) Wie groß ist ein durchschnittlicher erwachsener weißer Hai?
(3b) Wie groß ist ein durchschnittlicher erwachsener Riesenhai?
(4a) Welches Gewicht hat der durchschnittliche Weiße Hai?
(4b) Wie schwer kann der Riesenhai werden?
(5a) Wo lebt der weiße Hai?
(5b) Kommt der Weiße Hai im Mittelmeer vor?

(6a) Von welcher Pflanzenfamilie kommt Minze?
(6b) Welche Gewürze sind Nachtschattengewächse?
(7a) Was kann kombiniert werden mit Gewürzen, die zu Rindfleisch passen?
(7b) What spices does Szechuan pepper blend well with?
(8a) Mit welchen Gewürzen werden Fleischgerichte gewürzt?
(8b) What spices are commonly used to season Fish?
(9a) Von welchen Pflanzenfamilien kommen viele Gewürze?
(9b) Welche Gewürze werden aus Pflanzenwurzeln gewonnen?
(10a) Welche Gewürze werden in europäischer Küche verwendet?
(10b) Welche Gewürze sind typisch für die asiatische Küche?
(11a) Für was benutzt man Muskat?
(11b) Für was benutzt man thymian?
(12a) Zu welchen Gerichten passt Oregano?
(12b) Für was benutzt man Estragon
(13a) Welche Gewürze passen zu Ingwer?
(13b) Was kann mit Knoblauch kombiniert werden ?
(14a) Welche Gewürze schmecken scharf?
(14b) Welche Gewürze schmecken süß?
(15a) Welche Gewürze werden sowohl für süße als auch für salzige Speisen verwendet?
(15b) Welche Gewürze werden sowohl für süße als auch für saure Speisen verwendet?

(16a) Wer zählt zu den Patienten von Tinas Bruder?
(16b) Wessen Vater arbeitet bei der Agentur Löpelmann?
(17a) Welche Firmen gibt es?
(17b) Welche Personen gibt es ?
(18a) Wer hat geschwister
(18b) Wer ist wessen Chef ?
(19a) Gibt es Personen A und B so, dass A B kennt aber nicht umgekehrt?
(19b) Gibt es Schüler/innen, die noch nicht gearbeitet haben?
(20a) Gibt es eine Gruppe von mindestens vier Personen die alle schon einmal Steit miteinander hatten?
(20b) Gibt es zwei Gruppen von Personen so, dass keine Person der einen Gruppe eine Person der anderen Gruppe kennt?

(21a) Als was arbeitet Clemens?
(21b) Wer arbeitet am Yachthafen ?
(22a) Wen kennt Heiko?
(22b) Wen kennt Vera Richter?
(23a) Mit wem ist Erwin verheiratet?
(23b) Mit wem ist Vera verheiratet und wie heißt ihr Sohn?
(24a) wer steht schlecht zu heiko
(24b) Wer steht schlecht zu Vera?
(25a) Welche Personen heißen Richter mit Nachnamen?
(25b) Welche Personen heißen Becker mit Nachnamen?

A.5.4. Retrieval schedule

Scenario	Participant				
	1	2	3	4	5
A, first half	CDS	CDS	CDS	source	CDS
A, second half	CDS	source	CDS	CDS	CDS
B, first half	SMW	CDS	SMW	SMW	CDS
B, second half	CDS	SMW	CDS	CDS	SMW
C, first half	SMW	SMW	CDS	CDS	SMW
C, second half	CDS	CDS	SMW	SMW	CDS

Table A.2.: Assignment of tools per retrieval task per participant

Question pair	Participant 1	2	3	4	5
	Scenario A				
1	a	a	a	b	b
2	a	a	b	b	a
3	a	b	b	a	a
4	b	b	b	a	a
5	b	a	a	b	b
	Tool change				
1	b	b	b	a	a
2	b	b	a	a	b
3	b	a	a	b	b
4	a	a	a	b	b
5	a	b	b	a	a

Question pair	Participant 1	2	3	4	5
	Scenario B				
6	b	b	a	a	a
7	b	a	b	a	a
8	a	b	b	a	b
9	a	b	b	b	a
10	b	a	a	b	b
11	b	a	b	a	b
12	a	b	a	a	b
13	a	a	b	b	a
14	b	b	a	b	a
15	a	a	a	b	b
	Tool change				
6	a	a	b	b	b
7	a	b	a	b	b
8	b	a	a	b	a
9	b	a	a	a	b
10	a	b	b	a	a
11	a	b	a	b	a
12	b	a	b	b	a
13	b	b	a	a	b
14	a	a	b	a	b
15	b	b	b	a	a

Question pair	Participant 1	2	3	4	5
	Scenario C				
16	b	b	a	a	a
17	b	a	b	a	a
18	a	b	b	a	b
19	a	b	b	b	a
20	b	a	a	b	b
21	b	a	b	a	b
22	a	b	a	a	b
23	a	a	b	b	a
24	b	b	a	b	a
25	a	a	a	b	b
	Tool change				
16	a	a	b	b	b
17	a	b	a	b	b
18	b	a	a	b	a
19	b	a	a	a	b
20	a	b	b	a	a
21	a	b	a	b	a
22	b	a	b	b	a
23	b	b	a	a	b
24	a	a	b	a	b
25	b	b	b	a	a

Table A.3.: Distribution of questions pairs for scenarios A, B and C (left to right)

A.5.5. User study results

Participant	Scenario	Tool	e_1	e_{10}	e_{100}	Effectiveness[1]	Efficiency[2]
p_1	B	CDS	0.20	1.59	5.11	0.80	0.66
p_1	B	SMW	0.50	3.40	8.12	1.00	1.24
p_1	C	CDS	0.14	0.99	2.68	1.20	1.31
p_1	C	SMW	0.15	0.88	1.66	1.33	1.49
p_2	B	CDS	0.27	2.06	6.02	0.66	0.33
p_2	B	SMW	0.53	3.05	5.87	0.86	0.78
p_2	C	CDS	0.09	0.63	1.59	1.53	0.83
p_2	C	SMW	0.26	1.70	3.87	1.59	1.16
p_3	B	CDS	1.18	7.63	16.89	0.33	0.25
p_3	B	SMW	0.58	3.83	8.75	0.73	0.58
p_3	C	CDS	1.30	6.14	9.81	1.26	0.65
p_3	C	SMW	0.15	1.46	9.54	1.40	0.89
p_4	B	CDS	0.19	1.87	14.15	0.73	0.61
p_4	B	SMW	0.75	3.05	4.39	1.13	0.96
p_4	C	CDS	0.93	3.63	5.12	1.13	0.72
p_4	C	SMW	0.30	2.03	4.82	1.40	1.31
p_5	B	CDS	0.35	2.48	6.22	0.73	1.46
p_5	B	SMW	0.51	3.43	8.06	0.33	0.87
p_5	C	CDS	0.42	2.31	4.22	1.06	2.38
p_5	C	SMW	0.67	4.01	8.04	0.80	1.51
SD			0.35	1.73	3.87	0.37	0.50
SD CDS Tools			0.45	2.27	4.95	0.35	0.64
SD SMW			0.21	1.08	2.57	0.39	0.31
Mean			0.47	2.81	6.75	1.00	1.00
Mean CDS Tools			0.51	2.93	7.18	0.94	0.92
Mean SMW			0.44	2.68	6.31	1.06	1.08

[1] Normalised points p.

[2] Mean value of normalised e_1, e_{10} and e_{100}. Efficiency $= \frac{e_1+e_{10}+e_{100}}{3}$.

Table A.4.: Comparing efficiency of CDS Tools and SMW

Independent Variables					Dependent Variables		
P	S	O	T	Q	t_m	t_r	p
p_1	A		CDS	10	463.70	17.60	18.00
p_1	B	2	CDS	10	347.00	10.60	12.00
p_1	B	1	SMW	10	100.00	19.50	15.00
p_1	C	2	CDS	10	205.40	11.30	18.00
p_1	C	1	SMW	10	107.00	22.40	20.00
p_2	A_1	1	CDS	5	338.30	14.50	4.00
p_2	A_2	2	source[1]	5	927.30	5.80	12.00
p_2	B	1	CDS	10	356.20	32.70	10.00
p_2	B	2	SMW	10	246.00	13.70	13.00
p_2	C	2	CDS	10	488.80	18.10	23.00
p_2	C	1	SMW	10	332.00	15.30	24.00
p_3	A		CDS	10	230.80	20.20	22.00
p_3	B	2	CDS	10	317.10	15.70	5.00
p_3	B	1	SMW	10	212.00	22.20	11.00
p_3	C	1	CDS	10	419.30	25.30	19.00
p_3	C	2	SMW	10	392.00	16.80	21.00
p_4	A_1	1	source[1]	5	927.30	3.50	15.00
p_4	A_2	2	CDS	5	230.30	4.50	12.00
p_4	B	2	CDS	10	490.10	6.10	11.00
p_4	B	1	SMW	10	274.00	13.70	17.00
p_4	C	1	CDS	10	423.10	15.20	17.00
p_4	C	2	SMW	10	234.00	13.30	21.00
p_5	A		CDS	10	110.20	7.10	23.00
p_5	B	1	CDS	10	107.50	6.50	11.00
p_5	B	2	SMW	10	65.00	6.50	5.00
p_5	C	2	CDS	10	64.90	9.10	16.00
p_5	C	1	SMW	10	100.00	8.00	12.00
SD					220.18	7.06	5.66
Mean					315.16	13.90	15.07
Scenario B and C only							
SD					141.41	6.91	5.48
Mean					264.07	15.10	15.05
SD CDS Tools					150.29	8.49	5.31
Mean CDS Tools					321.94	15.06	14.20
SD SMW					110.39	5.35	5.80
Mean SMW					206.20	15.14	15.90

[1]For the Wikipedia pages the modelling effort was estimated as twice as long as the maximal time for a hand-created model in the user study. This is probably still much underestimated, given the large amount of time that went into the creation of the Wikipedia pages.

Table A.5.: Session data of comparative user study

Relation	Formality degree	Usage count	Usage in percent
[is related to]	0	106	5.43 %
[links to]	1	224	11.47 %
[is similar to]	1	8	0.41 %
[has detail]	2	688[1]	35.23 %
[is alias for]	2	126	6.45 %
[is same as]	2	12	0.61 %
[comes before]	2	2	0.10 %
[annotates]	2	0	0.00 %
[replaces]	2	0	0.00 %
[has subtype]	3	536[2]	27.44 %
[has part]	3	77	3.94 %
[is tag of]	3	19	0.97 %
[has instance]	4	155	7.94 %
Sum		1953	100.00 %

[1]Of which 134 have been created in HKW, 550 in iMapping, and 4 in QuiKey. This is no surprise, as iMapping creates one such Statement for every Item in a map to link it to its parent Item.

[2]Of which 266 have been created in HKW, 113 in iMapping, and 148 in QuiKey. 79 Statements are from NameItem to NameItem and 457 are from Relation to Relation.

Table A.6.: Usage of built-in relations in user-created statements

A.5.6. Relation and concept hierarchies

This section shows all 15 [has subtype] hierarchies created by the 5 participants of the comparative user study. The different scenarios have noticeable effects: For scenario C no participant created any concept hierarchies. Social networks seem to demand more sophisticated RELATION hierarchies instead.

The distribution of [has subtype]-STATEMENTS per participant is p_1: 69, p_2: 125, p_3: 131, p_4: 177, and p_5: 34. This shows that every individual participant used the [has subtype] RELATION.

Out of the 536 STATEMENTS, there are 79 [has subtype]-STATEMENTS between NAMEITEMS, which represent concept hierarchies.

Out of the 5 participants, 3 created [has subtype]-STATEMENTS between NAMEITEMS[4]

Legend:

- For hierarchies with no concept hierarchies, the root node is the top-RELATION [is related to]. For the other graphs, the concept hierarchy has the root concept "cds:item", even though there is no top-concept in CDS. All lines decending "cds:Item" or [is related to] have [has subtype]-semantics.
- Circled graph items represent NAMEITEMS and regular graph items represent RELATIONS.
- RELATIONS rendered in bold are built-in RELATIONS.

[4] One of them had worked on QuiKey before and created all [has subtype]-STATEMENTS in QuiKey. The other two participants used only iMapping and HKW. QuiKey might have a too steep learning curve.

Scenario A

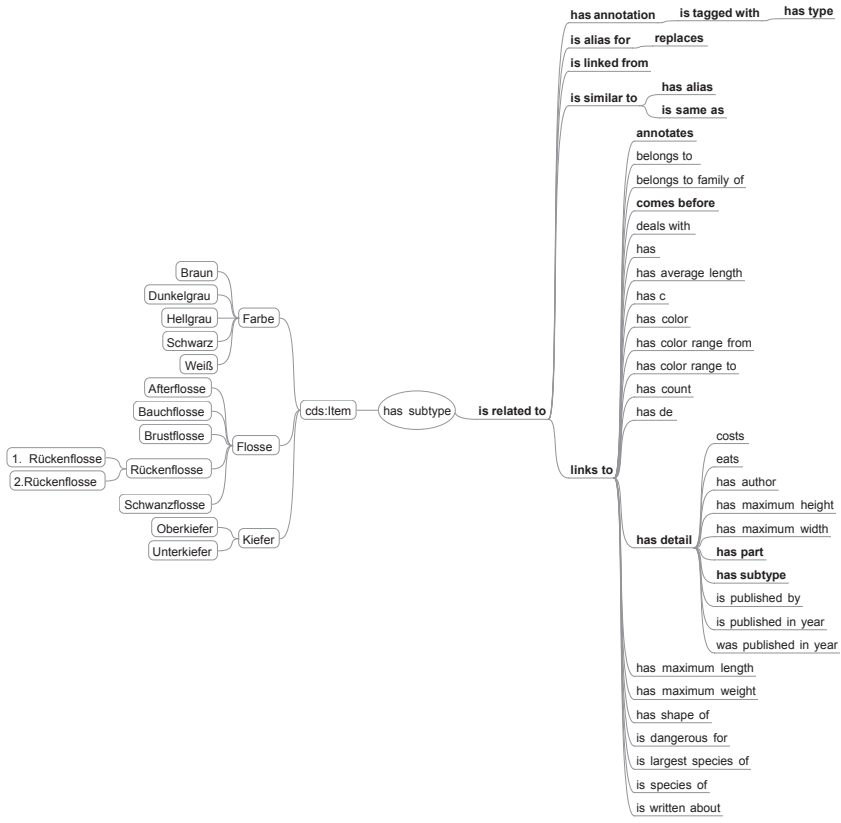


Figure A.1.: The [has subtype] hierarchy of participant 1 in scenario A

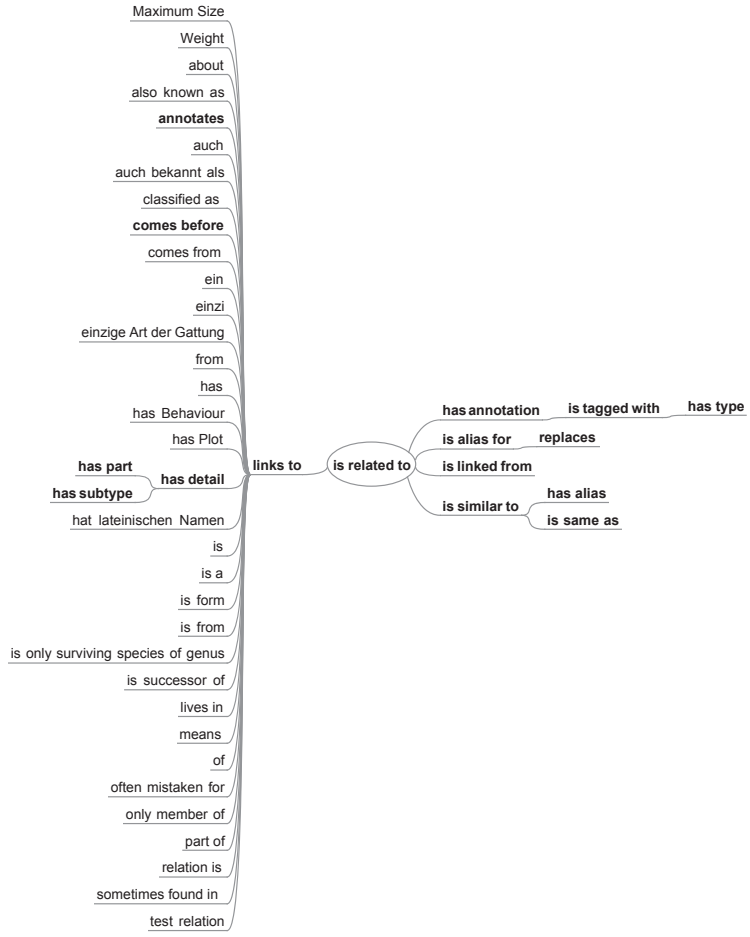


Figure A.2.: The [has subtype] hierarchy of participant 2 in scenario A

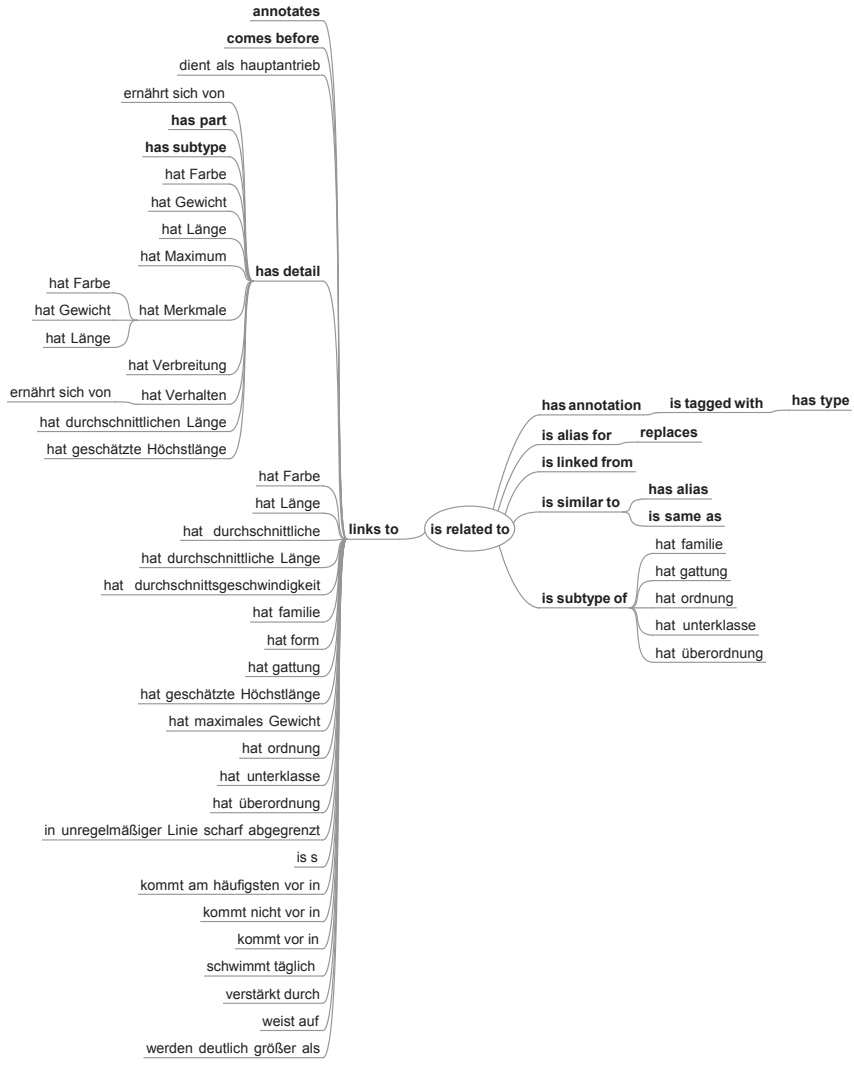


Figure A.3.: The [has subtype] hierarchy of participant 3 in scenario A

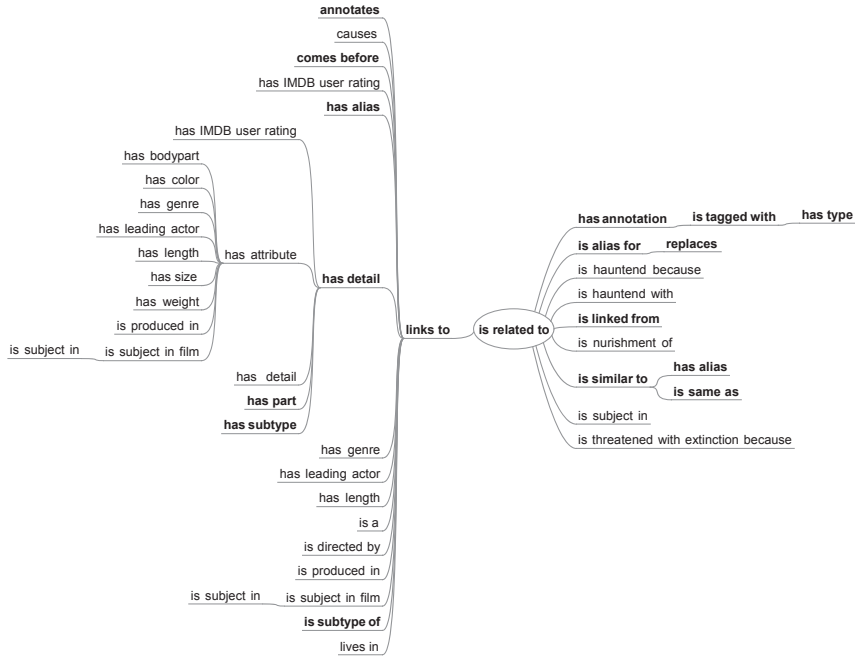


Figure A.4.: The [has subtype] hierarchy of participant 4 in scenario A

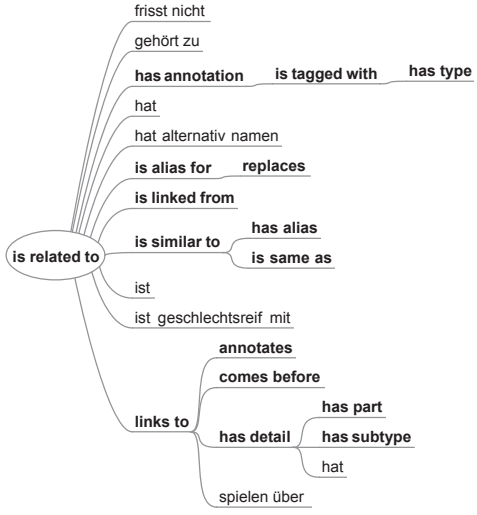


Figure A.5.: The [has subtype] hierarchy of participant 5 in scenario A

Scenario B

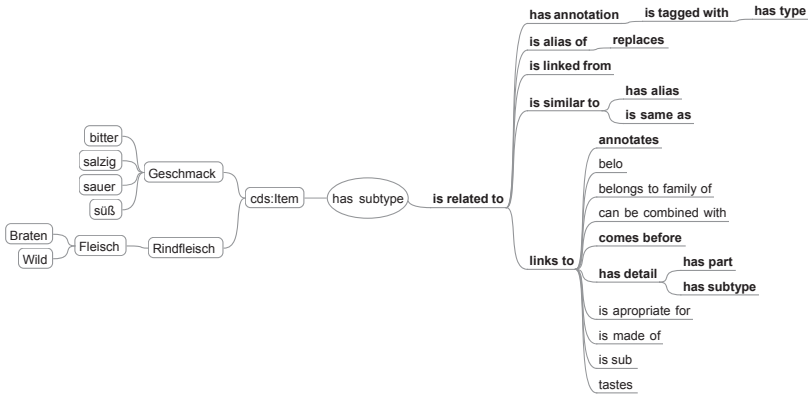
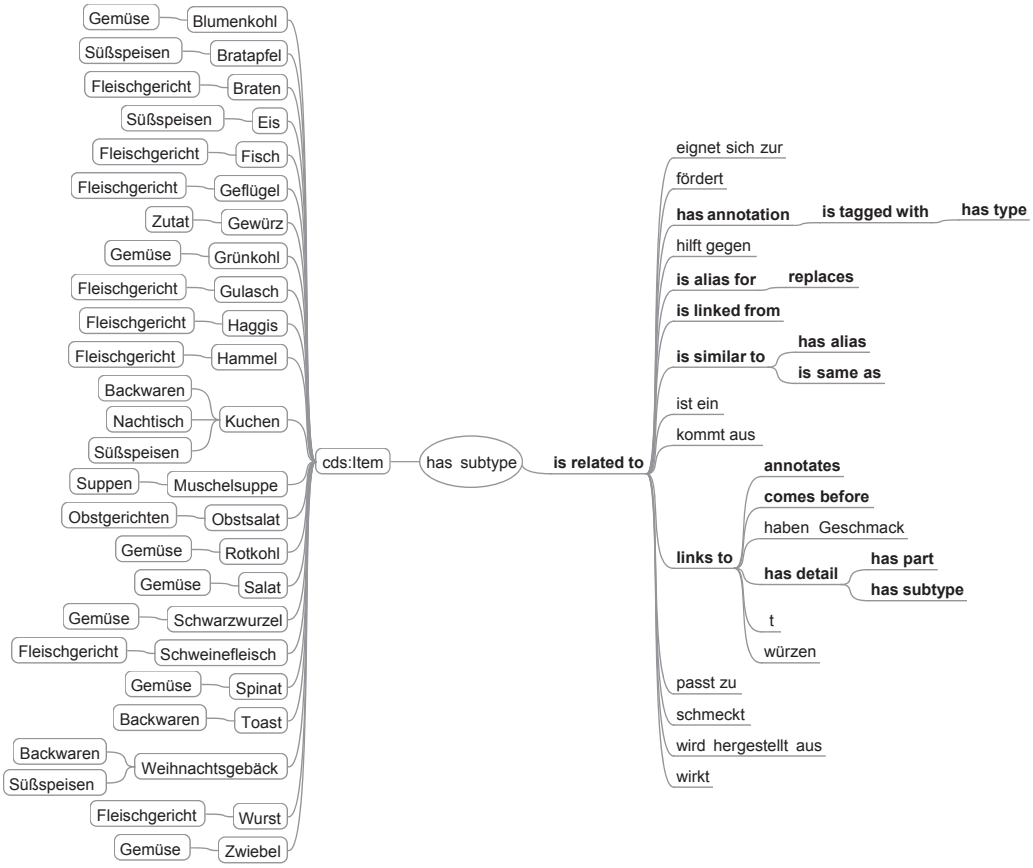


Figure A.6.: The [has subtype] hierarchy of participant 1 in scenario B



Most parts of the hierarchy seem to have been created in the opposite direction, e. g., in reality “Grünkohl” is a subtype of “Gemüse” rather than the other way round.

Figure A.7.: The [has subtype] hierarchy of participant 2 in scenario B

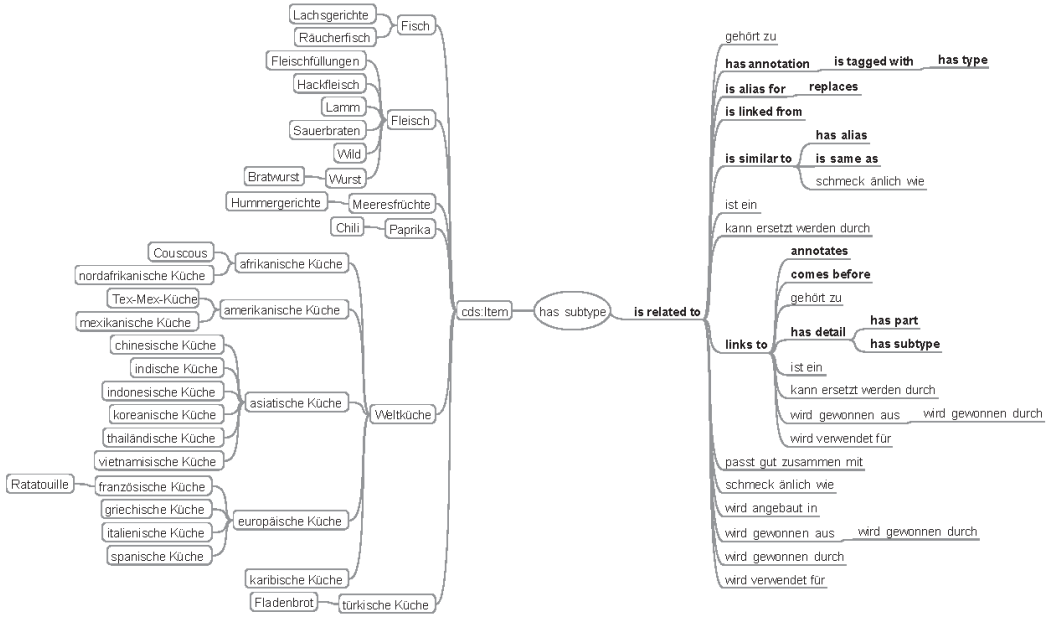


Figure A.8.: The [has subtype] hierarchy of participant 4 in scenario B

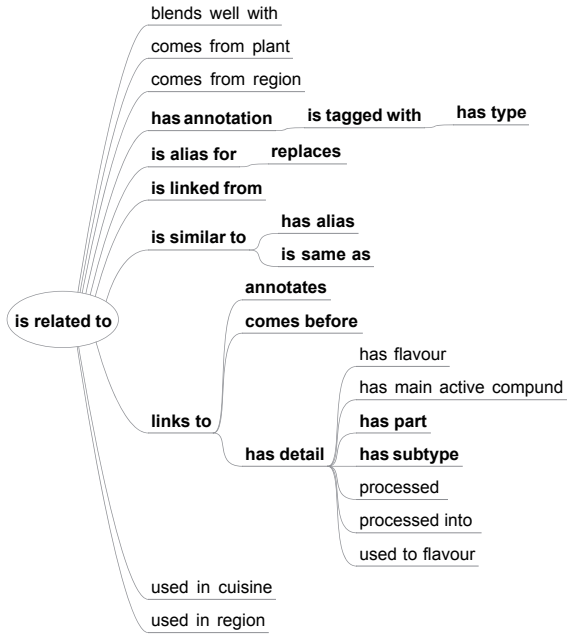


Figure A.9.: The [has subtype] hierarchy of participant 3 in scenario B

Scenario C

No single concept hierarchy STATEMENT ([has subtype]-STATEMENTS between NAMEITEMS) was created in scenario C by any participant.

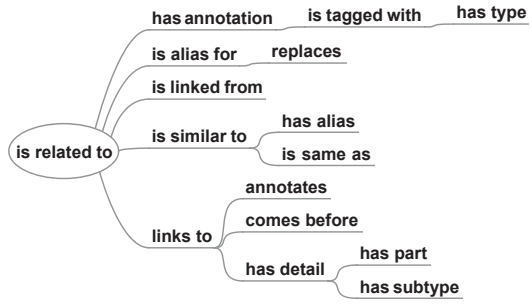


Figure A.10.: The [has subtype] hierarchy of participant 5 in scenario B

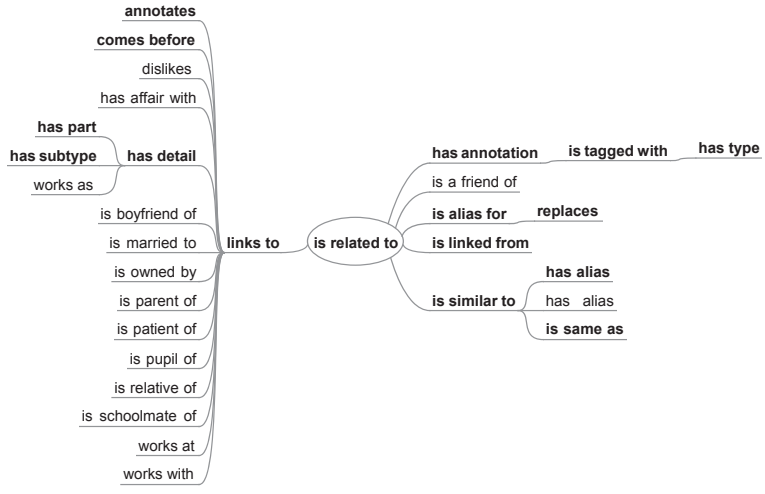


Figure A.11.: The [has subtype] hierarchy of participant 1 in scenario C

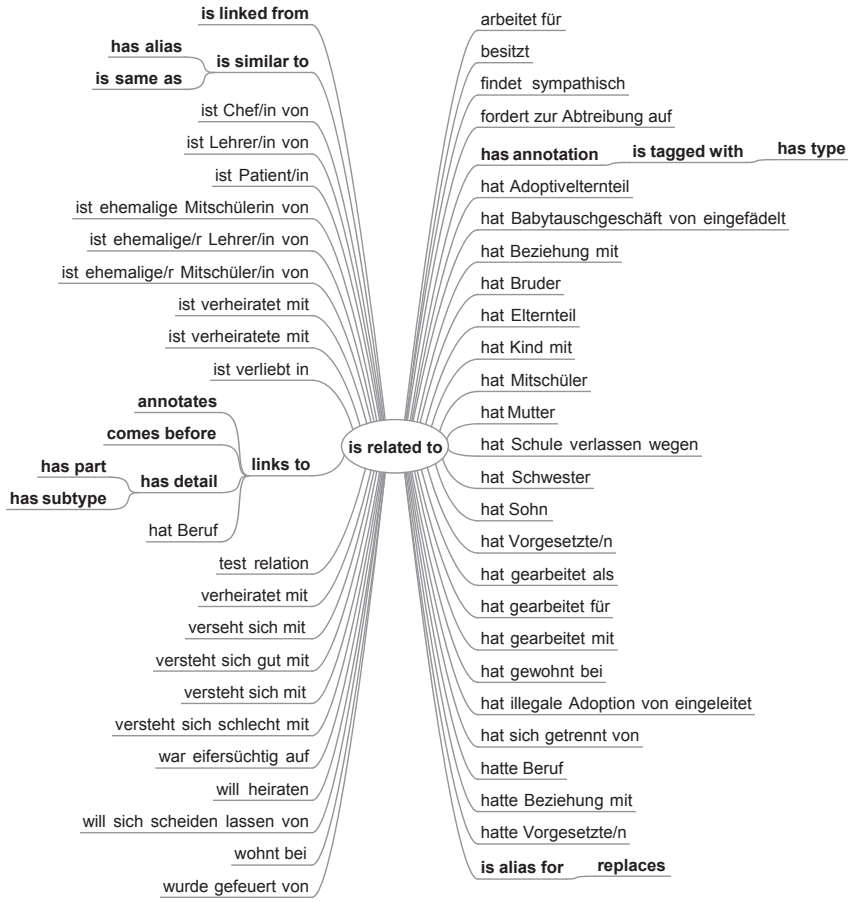


Figure A.12.: The [has subtype] hierarchy of participant 2 in scenario C

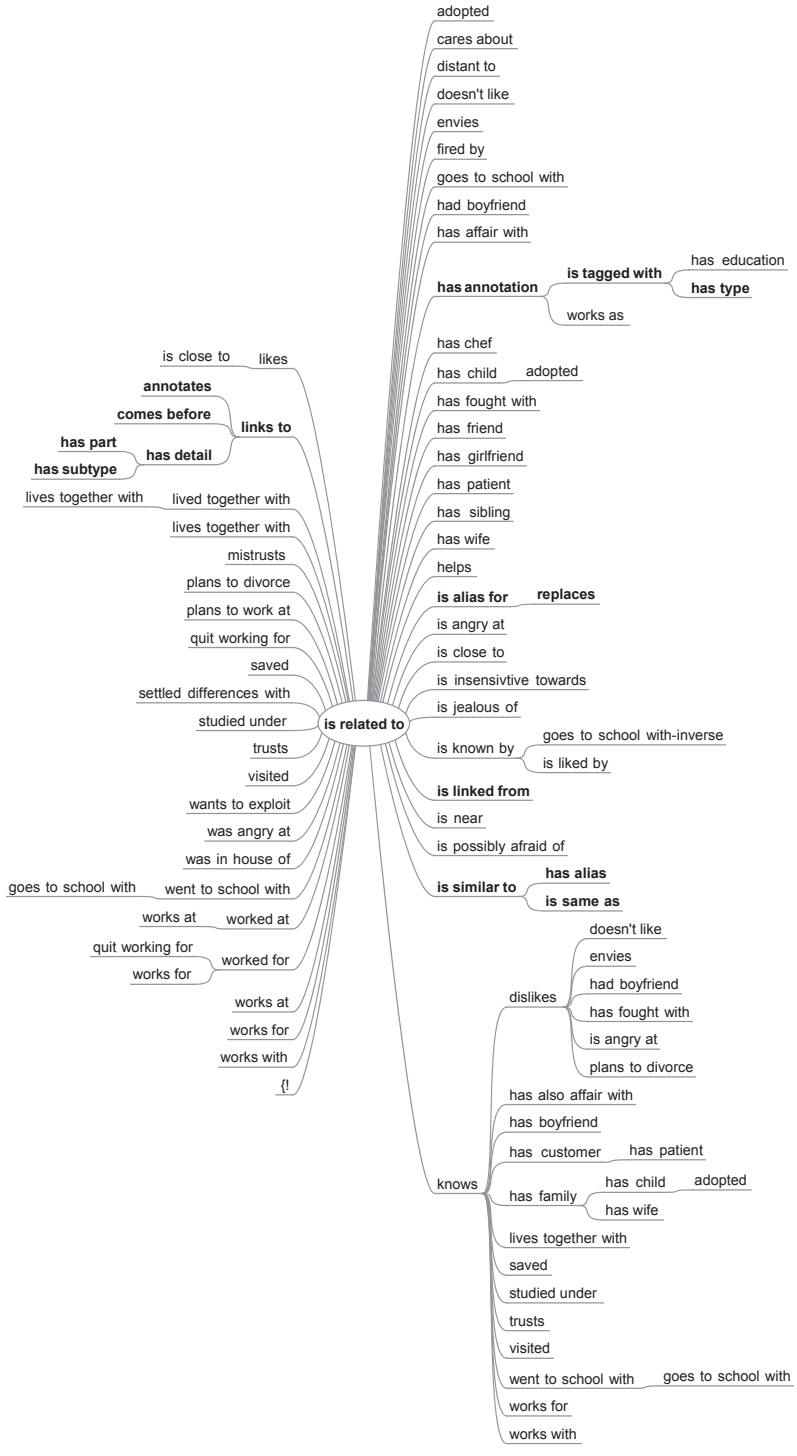


Figure A.13.: The [has subtype] hierarchy of participant 3 in scenario C

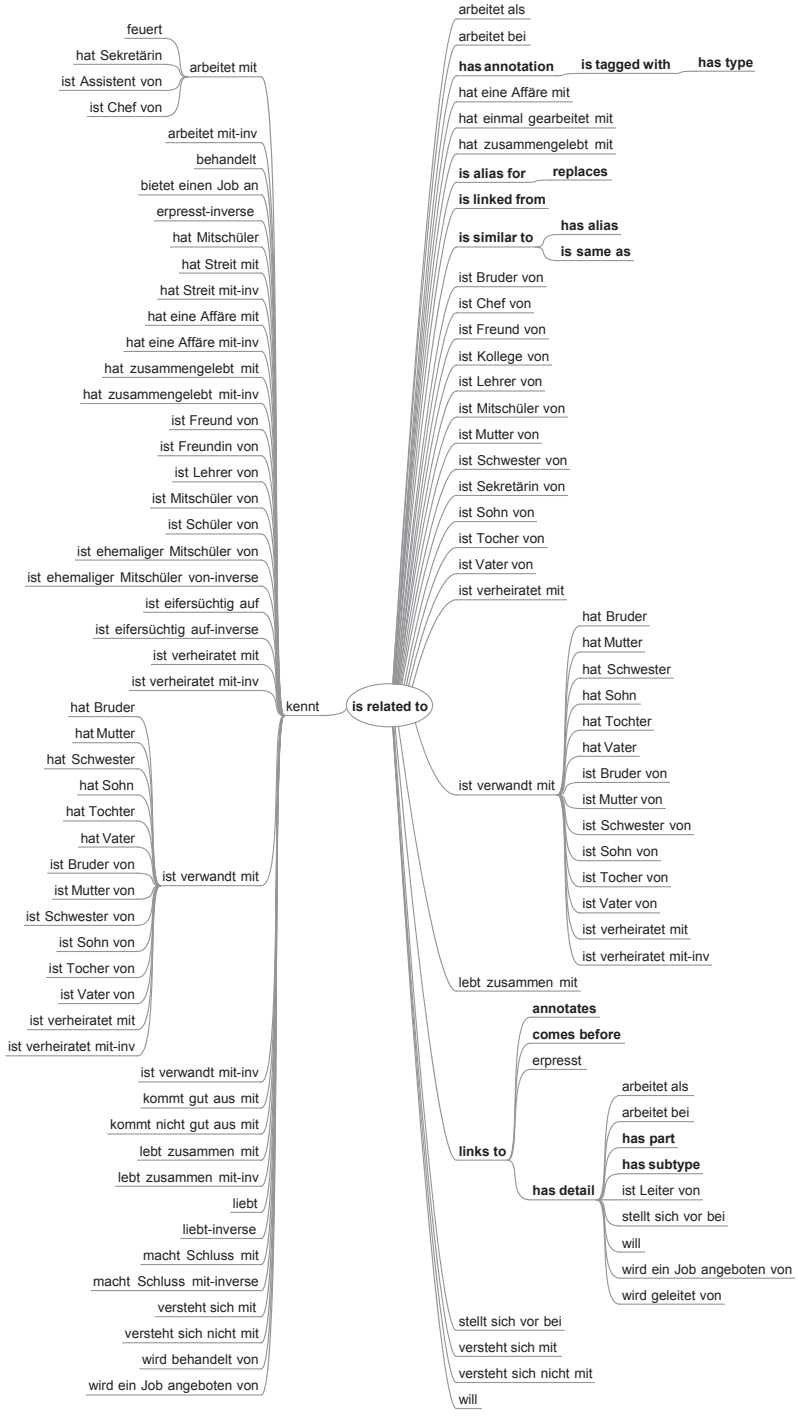


Figure A.14.: The [has subtype] hierarchy of participant 4 in scenario C

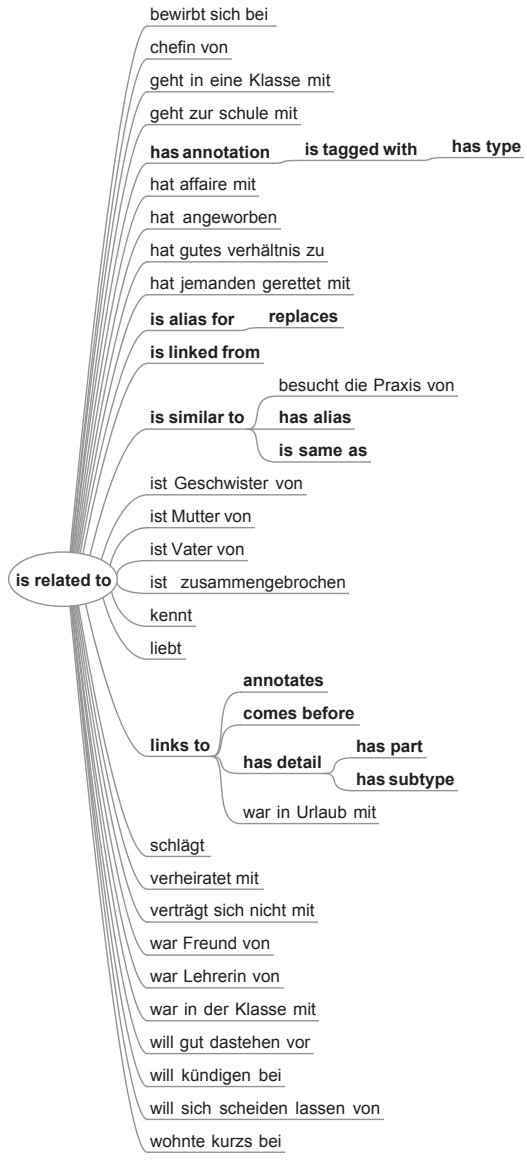


Figure A.15.: The [has subtype] hierarchy of participant 5 in scenario C

Bibliography

Abecker, A. (2004). *Business-Process Oriented Knowledge Management: Concepts, Methods and Tools*. Ph. D. thesis, Universität Karlsruhe (TH), Institut AIFB, D-76131 Karlsruhe.

Abecker, A., A. Bernardi, S. Ntioudis, G. Mentzas, R. Herterich, C. Houy, S. Müller, and M. Legal (2001). The decor toolbox for workflow-embedded organizational memory access. In *ICEIS (1)*, pp. 225–232.

Abecker, A. and L. van Elst (2004, January). Ontologies for knowledge management. In S. Staab and R. Studer (Eds.), *Handbook on Ontologies*, International Handbooks on Information Systems, pp. 435–454. Springer.

Adar, E., D. R. Karger, and L. A. Stein (1999). Haystack: Per-user information environments. In *CIKM*, pp. 413–422. ACM.

Allan, N., P. Heisig, P. Iske, D. Kelleher, M. Mekhilef, R. Oertel, A. J. Olesen, and M. V. L. (ES) (2004). European guide to good practice in knowledge management. workshop agreement CWA 14924-1:2004 E, European Committee for Standardization, rue de Stassart, 36 B-1050 Brussels.

Alley, L. R. (1999). Diverting a crisis in global human and economic development: A new transitional model for lifelong continuous learning and personal knowledge management. *Higher Education In Europe 24*(2), 187–195.

Altheim, M. and S. McCarron (2001, May). XHTML^{TM} 1.1: Module-based xhtml. Recommendation, W3C. W3C Recommendation 31 May 2001.

Alvarado, C., J. Teevan, M. S. Ackerman, and D. Karger (2003, April). Surviving the information explosion: How people find their electronic information. Technical Report AIM-2003-006, MIT AI Lab.

Anderson, J. R. (1980). *Cognitive psychology and its implications*. New York: Freeman, San Francisco, USA.

Angele, J., D. Fensel, D. Landes, and S. Neubert (1995). Modellbasiertes und inkrementelles Knowledge Engineering: der MIKE-Ansatz. *KI 9*(1), 16–21.

Avery, S., R. Brooks, J. Brown, P. Dorsey, and M. O'Conner (2001). Personal knowledge management: Framework for integration and partnerships. In *Proc. of ASCUE Conf.*

Baeza-Yates, R. and B. Ribeiro-Neto (1999, May). *Modern Information Retrieval* (1st ed.). Addison Wesley.

Barreau, D. and B. Nardi (1995). Finding and reminding: File organization from the desktop. *SIGCHI Bulletin 27*(3), 39–43.

Barth, S. (2004). *Knowledge Management Tools and Techniques: Practitioners and Experts Evaluate KM Solutions*, Chapter 28: Self-Organization: Taking a Personal Approach to KM. Butterworth-Heinemann.

Bates, M. (2002). Speculations on browsing, directed searching, and linking in relation to the bradford distribution. In *Emerging frameworks and methods: Proceedings of the Fourth International Conference on Conceptions of Library and Information Science (CoLIS 4)*, Greenwood Village, CO, pp. 137–150. Libraries Unlimited.

Beckett, D. and T. Berners-Lee (2008, January). Turtle - terse RDF triple language. Team submission, W3C.

Bederson, B. B. (2004, September). Interfaces for staying in the flow. *Ubiquity 5*(27), 1–1.

Berners-Lee, T., R. Fielding, and L. Masinter (2005, January). Uniform resource identifier (URI): Generic syntax. Request for Comments 3986, Network Working Group, IETF.

Bernstein, M. (2006, October). Shadows in the cave: hypertext transformations. In *Proc. of 2006 Symposium on Interactive Visual Information Collections and Activity (IVICA)*.

Bernstein, M., M. V. Kleek, D. R. Karger, and monica mc schraefel (2008, September). Information scraps: How and why information eludes our personal information management tools. *ACM Trans. Inf. Syst. 26*(4), 1–46.

Bernstein, M., M. V. Kleek, monica mc schraefel, and D. R. Karger (2007). Management of personal information scraps. In M. B. Rosson and D. J. Gilmore (Eds.), *CHI Extended Abstracts*, pp. 2285–2290. ACM.

Bernstein, M., M. V. Kleek, monica mc schraefel, and D. R. Karger (2008, April). Evolution and evaluation of an information scrap manager. Workshop on Personal Information Management at CHI 2008, April 5-6, 2008, Florence, Italy.

Bettoni, M. C., R. Ottiger, R. Todesco, and K. Zwimpfer (1998). Knowport: A personal knowledge portfolio tool. In U. Reimer (Ed.), *PAKM*, Volume 13 of *CEUR Workshop Proceedings*. CEUR-WS.org. `http://ceur-ws.org/Vol-13/paper6.ps` (accessed 06.01.2010).

Bilotti, M. W., P. Ogilvie, J. Callan, and E. Nyberg (2007). Structured retrieval for question answering. In *SIGIR '07: Proceedings of the 30th Annual International ACM SIGIR Conference on Research and Development in Information Retrieval*, New York, NY, USA, pp. 351–358. ACM Press.

Blackler, F. (1995). Knowledge, knowledge work and organizations: An overview and interpretation. *Organization Studies 16*(6), 1021–1046.

Blandford, A. E. and T. R. G. Green (2001). Group and individual time management tools: What you get is not what you need. *Personal Ubiquitous Comput. 5*(4), 213–230.

Bloehdorn, S., O. Görlitz, S. Schenk, and M. Völkel (2006, September). Tagfs - tag semantics for hierarchical file systems. In *Proceedings of the 6th International Conference on Knowledge Management (I-KNOW 06), Graz, Austria, September 6-8, 2006*.

Boardman, R. (2004). *Improving Tool Support for Personal Information Management*. Ph. D. thesis, Imperial College, London.

Boehm, B. W., J. R. Brown, H. Kaspar, M. Lipow, G. J. MacLeod, and M. J. Merritt (1978). *Characteristics of Software Quality*. New York, NY: North-Holland Publishing Company.

Boettger, M. (2005). PKM and "cues to knowledge". http://www.25uhr.de/weblog/index.php?itemid=70 (accessed 05.01.2010).

Bontas, E. P., C. Tempich, and Y. Sure (2006). Ontocom: A cost estimation model for ontology engineering. In I. Cruz et al. (Eds.), *Proceedings of the 5th International Semantic Web Conference (ISWC 2006)*, Volume 4273 of *Lecture Notes in Computer Science (LNCS)*, pp. 625–639. Springer-Verlag Berlin Heidelberg.

Boose, J. H. (1990). Knowledge acquisition tools, methods, and mediating representations. In *Proceedings of the First Japanese Knowledge Acquisition for Knowledge-Based Systems Workshop: JKAW-90, Ohmsha, Ltd: Japan.*

Bradshaw, S., M. Light, and D. Eichmann (2006). (bee)dancing on the boundary between pim and gim.

Braganza, A. and G. J. Mollenkramer (2002). Anatomy of a failed knowledge management initiative: Lessons from pharmacorp's experiences. *Knowledge and Process Management 9(1)*, 23–33.

Braun, S., A. Schmidt, and V. Zacharias (2009). Mit Social Semantic Bookmarking zur nützlichen Ontologie. *i-com - Zeitschrift für interaktive und kooperative Medien 8*.

Bruce, H. (2005). Personal, anticipated information need. *Information Research 3*(10). paper 232, available at `http://InformationR.net/ir/10-3/paper232.html` (accessed 19.01.2010).

Buckland, M. K. (1997). What is a "document"? *J. Am. Soc. Inf. Sci. 48*(9), 804–809.

Bush, V. (1945). As we may think. *Atlantic Monthly 176*, 101–108.

Buzan, T. (1991, January). *Use Both Sides of Your Brain: New Mind-Mapping Techniques, Third Edition (Plume)*. Plume, New York.

Cai, Y., X. L. Dong, A. Halevy, J. M. Liu, and J. Madhavan (2005). Personal information management with SEMEX. In *SIGMOD '05: Proceedings of the 2005 ACM SIGMOD International Conference on Management of Data*, New York, NY, USA, pp. 921–923. ACM Press.

Caldwell, F. (2002, November). Personal knowledge networks emerge with grassroots KM. Research Note SPA-18-8059, Gartner Research.

Carroll, J. J., C. Bizer, P. Hayes, and P. Stickler (2004, 05). Named graphs, provenance and trust. Technical report, HP.

Chaffee, J. and S. Gauch (2000). Personal ontologies for web navigation. In *CIKM '00: Proceedings of the ninth International Conference on Information and Knowledge Management*, New York, NY, USA, pp. 227–234. ACM.

Chen, P. P.-S. S. (1976). The entity-relationship model: Toward a unified view of data. *ACM Transactions on Database Systems 1*(1), 9–36.

Clemente Laboreo, D. (2007, 9). Diskussionssysteme. Studienarbeit, Universität Karlsruhe. `http://www.danielclemente.com/disk/` (accessed 05.01.2010).

Cleverdon, C. W., L. Mills, and M. Keen (1966). Factors determining the performance of indexing systems. Technical report, ASLIB Cranfield Research Project, Cranfield.

Conklin, J. and M. L. Begeman (1988). gIBIS: a hypertext tool for exploratory policy discussion. In *CSCW '88: Proceedings of the 1988 ACM Conference on Computer-Supported Cooperative Work*, New York, NY, USA, pp. 140–152. ACM Press.

Cutrell, E., S. T. Dumais, and J. Teevan (2006). Searching to eliminate personal information management. *Communications of the ACM 49*(1), 58–64.

Dan Brickley, R. G. (2004, February). RDF vocabulary description language 1.0: RDF Schema. `http://www.w3.org/TR/rdf-schema/` (accessed 05.01.2010).

Dau, F. and J. H. Correia (2006). *Formal Concept Analysis*, Chapter Two Instances of Peirce's Reduction Thesis, pp. 105–118. Springer Berlin / Heidelberg.

Davenport, T. H. (2005, September). *Thinking for a Living: How to Get Better Performances And Results from Knowledge Workers*. Harvard Business School Press, Boston, MA, USA.

Davies, S., S. Allen, J. Raphaelson, E. Meng, J. Engleman, R. King, and C. Lewis (2006). Popcorn: the personal knowledge base. In J. M. Carroll, S. Bødker, and J. Coughlin (Eds.), *Conference on Designing Interactive Systems*, pp. 150–159. ACM.

Davies, S. C. (2005). *The efficacy of personal knowledge bases for materializing mental impressions*. Ph. D. thesis, Boulder, CO, USA.

Decker, S. et al. (2000). The semantic web: The roles of XML and RDF. *IEEE Internet Computing 4*(5), 63–74.

Decker, S. and M. R. Frank (2004). The networked semantic desktop. In C. Bussler, S. Decker, D. Schwabe, and O. Pastor (Eds.), *WWW Workshop on Application Design, Development and Implementation Issues in the Semantic Web*, Volume 105 of *CEUR Workshop Proceedings ISSN 1613-0073*. CEUR-WS.org. `http://ceur-ws.org/Vol-105/DeckerFrank.pdf` (accessed 06.01.2010).

Decker, S., J. Park, D. Quan, and L. Sauermann (Eds.) (2005). *The Semantic Desktop – Next Generation Information Management & Collaboration Infrastructure*, Galway, Ireland.

DeRose, S. J., D. G. Durand, E. Mylonas, and A. H. Renear (1997). What is text, really? *SIGDOC Asterisk J. Comput. Doc. 21*(3), 1–24. Reprint from 1990.

Despres, C. and D. Chauvel (2000, October). *Knowledge Horizons: the present and promise of Knowledge Management*. Butterworth-Heinemann.

Dienel, H.-L. (2006). *Technografie. Zur Mikrosoziologie der Technik*, Chapter Schreiben, Zeichnen, Erinnern. Persönliches Wissensmanagement im Ingenieurberuf seit 1850, pp. 397–425. Frankfurt/New York: Campus Verlag.

Dittrich, J.-P. and M. A. V. Salles (2006). iDM: a unified and versatile data model for personal dataspace management. In *VLDB '06: Proceedings of the 32nd International Conference on Very Large Data Bases*, pp. 367–378. VLDB Endowment.

Dittrich, J.-P., M. A. V. Salles, D. Kossmann, and L. Blunschi (2005). iMeMex: escapes from the personal information jungle. In *VLDB '05: Proceedings of the 31st International Conference on Very Large Data Bases*, pp. 1306–1309. VLDB Endowment.

Dix, A., A. Katifori, A. Poggi, T. Catarci, Y. Ioannidis, G. Lepouras, and M. Mora (2007, December). From information to interaction: in pursuit of task-centred information management. In F. Borri, A. Launaro, and C. Thanos (Eds.), *Proc. of the Second DELOS Conference on Digital Libraries, 5-7 December 2007, Tirrenia, Pisa, ITALY*, Volume 4877 of *Lecture Notes in Computer Science*. DELOS Network of Excellence.

Dörner, D. (2003). *Die Logik des Mißlingens. Strategisches Denken in komplexen Situationen.* Rowohlt Tb.

Doyle, J. (1987). *A truth maintenance system*, Chapter 20, pp. 259–279. San Francisco, CA, USA: Morgan Kaufmann Publishers Inc.

Drucker, P. (1977). Managing oneself. *Harvard Business Review 2*, 65–74.

Drucker, P. F. (1985, February). *Management: Tasks, responsibilities, practices (Harper & Row management library)*. Harper & Row.

Drucker, P. F. (1999). Knowledge-worker productivity: The biggest challenge. *California Management Review 41*(2), 79–94.

Durusau, P. and S. Newcomb (2005, 02). Topic maps reference model. ISO committee draft, ISO 13250: Topic Maps.

Elkins, J. (1999). *The Domain of Images*. Cornell University Press.

Elsweiler, D. and I. Ruthven (2007). Towards task-based personal information management evaluations. In *SIGIR '07: Proceedings of the 30th Annual International ACM SIGIR Conference on Research and Development in Information Retrieval*, New York, NY, USA, pp. 23–30. ACM Press.

Engelbart, D. (1963). *A Conceptual Framework for the Augmentation of Man's Intellect*, pp. 1–29. Spartan Books: Washington, DC: London. Technical Report version: `http://sloan.stanford.edu/mousesite/EngelbartPapers/B5_F18_ConceptFrameworkInd.html` (accessed 05.01.2010).

Esselborn-Krumbiegel, H. (2002, November). *Von der Idee zum Text. Eine Anleitung zum wissenschaftlichen Schreiben.* Utb. 2. Auflage.

Esser, K. B. (1998). *Ein Modell zur Verknüpfung des persönlichen Gedächtnisses mit externen Informationsspeichern*. Ph. D. thesis, Freie Universität Berlin.

Ewenstein, B. and J. K. Whyte (2007). Picture this: visual representations as "artifacts of knowing". *Building Research and Information 35*(1), 81–89.

Feldman, S., J. Duhl, J. R. Marobella, and A. Crawford (2005). The hidden costs of information work. Technical report, IDC.

Fellbaum (1998, May). *WordNet: An Electronic Lexical Database (Language, Speech, and Communication)*. The MIT Press.

Fielding, R. T. (2000). *Architectural styles and the design of network-based software architectures*. Ph. D. thesis, University of California, Irvine.

Folkens, F. and M. Spiliopoulou (2004). Towards an evaluation framework for knowledge management systems. In D. Karagiannis and U. Reimer (Eds.), *PAKM*, Volume 3336 of *Lecture Notes in Computer Science*, pp. 23–34. Springer.

Franconi, E., M. Kifer, and W. May (Eds.) (2007). *The Semantic Web: Research and Applications, 4th European Semantic Web Conference, ESWC 2007, Innsbruck, Austria, June 3-7, 2007, Proceedings*, Volume 4519 of *Lecture Notes in Computer Science*. Springer.

Frand, J. and C. Hixon (1999, December). Personal knowledge management: Who, what, why, when, where, how? `http://www.anderson.ucla.edu/faculty/jason.frand/researcher/speeches/PKM.htm` (accessed 05.01.2010). working paper.

Frank, G. H. (1988). Reflections on notecards: seven issues for the next generation of hypermedia systems. *Communications of the ACM 31*(7), 836–852.

Frank, U. (2000). Modelle als Evaluationsobjekt: Einführung und Grundlegung. In I. Häntschel and L. Heinrich (Eds.), *Evaluation und Evaluationsforschung in der Wirtschaftsinformatik*, pp. 339–352. Oldenbourg.

Freed, N. and N. Borenstein (1996, November). Multipurpose Internet Mail Extensions (MIME) Part Two: Media Types. Technical Report 2046, Internet Engineering Task Force, Network Working Group. Updated by RFCs 2646, 3798, 5147.

Friedewald, M. (2000, March). *Der Computer als Werkzeug und Medium. Die geistigen und technischen Wurzeln des Personalcomputers* (Taschenbuch ed.). GNT-Verlag.

Geiß, R. (2008, November). *Graphersetzung mit Anwendungen im Übersetzerbau* . Ph. D. thesis, Universität Karlsruhe (TH), Germany.

Gerke, S. (2005). Interaktives Erkennen von Textstrukturen durch Maschinelles Lernen. Studienarbeit, Institute AIFB, University of Karlsruhe.

Glushko, R. (2006, September). 3. information organization {and,or,vs} search. Lecture Note.

Graça Pimentel, M. d., G. D. Abowd, and Y. Ishiguro (2000). Linking by interacting: a paradigm for authoring hypertext. In *HYPERTEXT '00: Proceedings of the eleventh ACM Conference on Hypertext and Hypermedia*, New York, NY, USA, pp. 39–48. ACM Press.

Grebner, O. (2009). *Using Unified Personal Information in Workspaces*. Ph. D. thesis, Fakultät für Wirtschaftswissenschaften, Institut AIFB.

Groth, K. . K. S. E. (2006). Combining personal and organisational information.

Groza, T., S. Handschuh, K. Möller, and S. Decker (2007). Salt - semantically annotated latex for scientific publications. See Franconi, Kifer, and May (2007), pp. 518–532.

Gruber, T. (1993). A translation approach to portable ontology specifications. *Knowledge Acquisition 5*, 199–220.

Gruber, T. R. (1989). *The acquisition of strategic knowledge*. San Diego, CA, USA: Academic Press Professional, Inc.

Halasz, F. and M. Schwartz (1990, January). The dexter hypertext reference model. In *NIST Hypertext Standardization Workshop*.

Halasz, F. G., T. P. Moran, and R. H. Trigg (1987). Notecards in a nutshell. In *CHI '87: Proceedings of the SIGCHI/GI Conference on Human Factors in Computing Systems and Graphics Interface*, New York, NY, USA, pp. 45–52. ACM.

Haller, H. (2003). Mappingverfahren zur Wissensorganisation. `http://heikohaller.de/literatur/diplomarbeit/` (accessed 05.01.2010).

Haller, H. (2006, November). iMapping – a graphical approach to semi-structured knowledge modelling. In L. Rutledge (Ed.), *Proceedings of the The 3rd International Semantic Web User Interaction Workshop (SWUI2006)*. Poster presented at the The 3rd International Semantic Web User Interaction Workshop.

Haller, H. (2008). Quikey. In *Proceedings of the Workshop on Semantic Search at the 5th European Semantic Web Conference*, Volume 334, pp. 74–78.

Haller, H., A. Abecker, and M. Völkel (2010, June). A graphical workbench for knowledge workers. In *ICEIS 2010 – 12th International Conference on Enterprise Information Systems, Funchal, Madeira - Portugal, June 8-12, 2010*. To appear.

Haller, H., M. Völkel, and F. Kugel (2006, June). iMapping wikis - towards a graphical environment for semantic knowledge management. In S. Schaffert, M. Völkel, and S. Decker (Eds.), *Proceedings of the First Workshop on Semantic Wikis – From Wiki To Semantics, Budva, Montenegro*, Volume 360 of *CEUR Workshop Proceedings*. CEUR-WS.org.

Harth, A. and S. Decker (2005). Optimized index structures for querying RDF from the web. In *Proceedings of the Third Latin American Web Congress (LA-Web 2005), 1 October - 2 November 2005, Buenos Aires, Argentina*, pp. 71–80. IEEE Computer Society.

Hayes, P. (2004, Februar). RDF semantics. Recommendation, W3C.

Heitmann, B., E. Oren, and M. Völkel (2006). Revisiting and simplifying RDF. Technical Report 2006-15, DERI Galway.

Hennum, E. (2006, August). Representing discourse models in RDF. In *Extreme Markup Languages 2006 (R)*, Montréal, Québec.

Higgison, S. (2004, April). Your say: Personal knowledge management. *Inside Knowledge 7*(7).

Hitzler, P., M. Krötzsch, B. Parsia, P. F. Patel-Schneider, and S. Rudolph (2009, October). OWL 2 web ontology language:primer. W3C recommendation, W3C. http://www.w3.org/TR/owl2-primer/.

Holsapple, C. W. (2004). *Handbook on knowledge management*, Chapter Knowledge and Its Attributes, pp. 165–188. Springer, Berlin; Heidelberg.

Huhns, M. and L. Stephens (1999, Sep/Oct). Personal ontologies. *Internet Computing, IEEE 3*(5), 85–87.

Hussmann, H. (1997). *Formal foundations for software engineering methods*. Springer.

Immaneni, T. and K. Thirunarayan (2007). A unified approach to retrieving web documents and semantic web data. See Franconi et al. (2007), pp. 579–593.

Iske, P. and T. Boekhoff (2002). The value of knowledge doesn't exist. In D. Karagiannis and U. Reimer (Eds.), *PAKM*, Volume 2569 of *Lecture Notes in Computer Science*, pp. 632–638. Springer.

ISO (1991a). Ergonomic principles related to mental work-load. Technical Report ISO 10075:1991, ISO, Geneva.

ISO (1991b). Information processing – text and office systems – standard generalized markup language (SGML). Technical Report ISO 8879:1986, ISO, Geneva.

Jäschke, R., A. Hotho, C. Schmitz, B. Ganter, and G. Stumme (2008, February). Discovering shared conceptualizations in folksonomies. *Journal of Web Semantics 6*(1), 38–53.

Johnson, N. E. (1989). Mediating representations in knowledge elicitation. pp. 177–194.

Johnson-Laird, P. N. (1983). *Mental models: towards a cognitive science of language, inference, and consciousness*. Cambridge, MA, USA: Harvard University Press.

Jonassen, D. H. (2000). Toward a design theory of problem solving. *Educational Technology Research and Development 48*(4), 63–85.

Jones, W. and H. Bruce (Eds.) (2005, January). *A Report on the NSF-Sponsored Workshop on Personal Information Management, Seattle, WA, 2005*.

Jones, W., H. Bruce, and S. Dumais (2001). Keeping found things found on the web. In *CIKM '01: Proceedings of the tenth International Conference on Information and Knowledge Management*, New York, NY, USA, pp. 119–126. ACM Press.

Jones, W., A. J. Phuwanartnurak, R. Gill, and H. Bruce (2005). Don't take my folders away!: organizing personal information to get things done. In G. C. van der Veer and C. Gale (Eds.), *CHI Extended Abstracts*, pp. 1505–1508. ACM.

Jones, W., P. Pirolli, S. K. Card, R. Fidel, N. D. Gershon, P. Morville, B. A. Nardi, and D. M. Russell (2006). "it's about the information stupid!": why we need a separate field of human-information interaction. In G. M. Olson and R. Jeffries (Eds.), *CHI Extended Abstracts*, pp. 65–68. ACM.

Jones, W. P. and J. Teevan (2007, October). *Personal Information Management*. University of Washington Press.

Jüngst, K. L. (1992). *Lehren und Lernen mit Begriffsnetzdarstellungen. Zur Nutzung von concept-maps bei der Vermittlung fachspezifischer Begriffe in Schule, Hochschule, Aus- und Weiterbildung*. AFRA.

Kalnikaitė, V. and S. Whittaker (2008). Cueing digital memory: how and why do digital notes help us remember? In *BCS-HCI '08: Proceedings of the 22nd British HCI Group Annual Conference on HCI 2008*, Swinton, UK, UK, pp. 153–161. British Computer Society.

Khan, F. (1994, February). A survey of note-taking practices. Technical Report HPL-93-107, Hewlett-Packard.

Kidd, A. (1994). The marks are on the knowledge worker. In *CHI '94: Proc. of the SIGCHI conf. on Human factors in computing systems*, pp. 186–191. ACM Press.

Kirschner, P. A., S. J. B. Shum, and C. S. Carr (Eds.) (2003). *Visualizing argumentation: software tools for collaborative and educational sense-making*. London, UK: Springer.

Klyne, G. and J. J. Carroll (2004, February). Resource description framework (RDF): Concepts and abstract syntax. `http://www.w3.org/TR/2004/REC-rdf-concepts-20040210/` (accessed 05.01.2010).

Krohn, T., M. C. Kindsmüller, and M. Herczeg (2008). myPIM: a graphical information management system for web resources. In *ICPW '08: Proceedings of the 3rd International Conference on the Pragmatic Web*, New York, NY, USA, pp. 3–12. ACM.

Krötzsch, M., S. Rudolph, and P. Hitzler (2008, October). ELP: Tractable rules for OWL 2. In A. Sheth, S. Staab, M. Dean, M. Paolucci, D. Maynard, T. Finin, and K. Thirunarayan (Eds.), *Proceedings of the 7th International Semantic Web Conference (ISWC 2008)*, Volume 5318 of *LNCS*, pp. 649–664. Springer.

Krötzsch, M., D. Vrandečić, M. Völkel, H. Haller, and R. Studer (2007). Semantic wikipedia. *Journal of Web Semantics 5*(4), 251–261.

Kunz, W. and H. W. J. Rittel (1970). Issues as elements of information systems. Technical report wp-131, University of California, Berkeley.

Lamming, M., P. Brown, K. Carter, M. Eldridge, M. Flynn, G. Louie, P. Robinson, and A. Sellen (1994). The Design of a Human Memory Prosthesis. *The Computer Journal 37*(3), 153–163.

Lethbridge, T. (1998). Metrics for concept-oriented knowledge bases. *International Journal of Software Engineering and Knowledge Engineering 8*(2), 161–188.

Lethbridge, T. C. (1991a, October). Creative knowledge acquisition: An analysis. In *Proc. 6th Knowledge Acquisition for Knowledge-Based Systems Workshop, Banff, Alberta*, pp. 12.1–12.20.

Lethbridge, T. C. (1991b). A model for informality in knowledge representation and acquisition (an extended abstract). In *Workshop on Informal Computing, Santa Cruz, Incremental Systems*, pp. 175–177.

Lethbridge, T. C. (1994). *Practical Techniques for Organizing and Measuring Knowledge*. Ph. D. thesis, University of Ottawa.

Leuf, B. and W. Cunningham (2001). *The wiki way: Quick collaboration on the web*. Addison-Wesley.

Lie, H. W. and B. Bos (1999). Cascading style sheets, level 1. Technical Report REC-CSS1-19990111, W3C. W3C Recommendation 17 Dec 1996, revised 11 Jan 1999.

Likert, R. (1932). A technique for the measurement of attitudes. *Archives of Psychology 22*(140), 1–55.

Löffler, K. and N. Fischer (1956). *Einführung in die Katalogkunde*. Anton Hiersemann, Stuttgart, Germany.

Lorie, R. A. (2001). Long term preservation of digital information. In *JCDL '01: Proceedings of the 1st ACM/IEEE-CS Joint Conference on Digital Libraries*, New York, NY, USA, pp. 346–352. ACM.

Ludwig, L. (2005, June). Semantic personal knowledge management. Technical Report D11.01_v0.01, DERI Galway, University Road, Galway, Ireland. Cached version available at `http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.123.7872&rep=rep1&type=pdf` (accessed 05.01.2010).

Luhmann, N. (1992). Kommunikation mit Zettelkästen. Ein Erfahrungsbericht. In A. Kieserling (Ed.), *Universität als Milieu*, Kleine Schriften, pp. 53–61. Haux Verlag, Bielefeld. ISBN 3-925471-13-8.

Maier, D., D. Archer, L. Delcambre, S. M. Hy, T. J. Annareddy, L. N. Cassel, D. Gangula, G. B. Teng, E. A. Fox, and U. Murthy (2006). Personal information enhancement for education.

Maier, R. and A. Schmidt (2007). Characterizing knowledge maturing: A conceptual process model for integrating e-learning and knowledge management. In N. Gronau (Ed.), *4th Conference Professional Knowledge Management - Experiences and Visions (WM '07), Potsdam*, Volume 1, Berlin, pp. 325–334. GITO.

Malone, T. W. (1983). How do people organize their desks?: Implications for the design of office information systems. *ACM Trans. Inf. Syst. 1*(1), 99–112.

Marshall, B., Y. Zhang, H. Chen, A. M. Lally, R. Shen, E. A. Fox, and L. N. Cassel (2003). Convergence of knowledge management and e-learning: The getsmartexperience. In *JCDL*, pp. 135–146. IEEE Computer Society.

Marshall, C. C. and A. J. B. Brush (2004). Exploring the relationship between personal and public annotations. In *JCDL '04: Proceedings of the 4th ACM/IEEE-CS Joint Conference on Digital Libraries*, New York, NY, USA, pp. 349–357. ACM.

Maturana, H. R. and F. J. Varela (1987). *Der Baum der Erkenntnis* (2. Aufl. ed.). Scherz.

Maurer, H. (1999). The heart of the problem: Knowledge management and knowledge transfer. In *Proc. ENABLE'99*, pp. 8–17. Espoo-Vantaa Institute of Technology.

Miles, A. and S. Bechhofer (2008, January). Skos simple knowledge organization system reference. Working draft, World Wide Web Consortium. `http://www.w3.org/TR/2009/REC-skos-reference-20090818/` (accessed 06.01.2010).

Miller, G. (1956). The magical number seven, plus or minus two: Some limits on our capacity for processing information. *Psychological Review 63*, 81–97.

Miller, R. C. (2002). *Lightweight Structure in Text*. Ph. D. thesis, School of Computer Science, Carnegie Mellon University.

Minack, E., L. Sauermann, G. Grimnes, C. Fluit, and J. Broekstra (2008). The Sesame LuceneSail: RDF queries with full-text search. Technical report, NEPOMUK. Available at `http://nepomuk.semanticdesktop.org/xwiki/bin/download/Main1/Publications/Minack%202008.pdf` (accessed 05.01.2010).

Mitchell, A. (2005, September). The rise of personal KM. *Inside Knowledge 9*(1).

Musen, M. A. (1989, March). Conceptual models of interactive knowledge acquisition tools. *Knowledge Acquisition 1*(1), 73–88.

National Library of Medicine (U.S.), Board of Regents (1987). *Long Range Plan*. U.S. Dept. of Health and Human Services, Public Health Service, National Institutes of Health.

Nelson, T. H. (1995). The heart of connection: hypermedia unified by transclusion. *Communications of the ACM 38*(8), 31–33.

NEPOMUK Consortium, M. Völkel, H. Haller, W. Bolinder, B. Davis, H. Edlund, K. Groth, R. Gudjonsdottir, M. Kotelnikov, P. Lannerö, S. Lundquist, M. Sogrin, Y. Sundblad, and B. Westerlund (2008, January). Conceptual data structure tools. Deliverable 1.2, NEPOMUK consortium. Copy available at `http://xam.de/2008/D1.2_v10_CDS-Tools.pdf` (accessed 05.01.2010).

Nevo, D., B. Furneaux, and Y. Wand (2008, December). Towards an evaluation framework for knowledge management systems. *Information Technology and Management 9*(4), 233–249.

Nissen, M. E. (2005, October). *Harnessing Knowledge Dynamics: Principled Organizational Knowing & Learning*. IRM Press.

Noga, M. L. and M. Völkel (2003, November). From web pages to web services with WAL. In *NCWS 2003*, Växjö, Sweden. Mathematical Modelling in Physics Engineering and Cognitive Science.

Noirhomme-Fraiture, M. and V. Serpe (1998). Visual representation of hypermedia links according to their types. In *AVI '98: Proceedings of the working conference on Advanced Visual Interfaces*, New York, NY, USA, pp. 146–155. ACM.

Nonaka, I. (1994, Februar). A dynamic theory of organizational knowledge creation. *Organization Science 5*(1), 14–37.

Nonaka, I. and N. Konno (1998). The concept of "ba": Building a foundation for knowledge creation. *California Management Review 40*(3), 40–54.

Nonaka, I. and H. Takeuchi (1995, May). *The Knowledge-Creating Company : How Japanese Companies Create the Dynamics of Innovation*. Oxford University Press.

North, K. (2002). *Wissensorientierte Unternehmensführung* (3rd ed.). Dr. Th. Gabler Verlag.

North, K. (2007, October). Produktive Wissensarbeit. In *5. Karlsruher Symposium für Wissensmanagement in Theorie und Praxis*. CD-ROM.

Novak, J. D. and A. J. Canas (2006, January). The theory underlying concept maps and how to construct them. Technical report, Institute for Human and Machine Cognition.

Noy, N. F., M. Sintek, S. Decker, M. Crubézy, R. W. Fergerson, and M. A. Musen (2001). Creating semantic web contents with Protégé-2000. *IEEE Intelligent Systems 16*(2), 60–71.

O'Hara, K. and A. Sellen (1997). A comparison of reading paper and on-line documents. In *CHI*, pp. 335–342.

OMG (2007). Unified modeling language: Superstructure, version 2.1.1. Technical Report formal/2007-02-03, Object Management Group.

Orchard, D., E. Maler, and S. DeRose (2001, June). XML linking language (XLink) version 1.0. W3C recommendation, W3C. http://www.w3.org/TR/2001/REC-xlink-20010627/.

Oren, E. (2005). SemperWiki: a semantic personal wiki. See Decker et al. (2005).

Oren, E. (2006). An overview of information management and knowledge work studies: Lessons for the semantic desktop. In S. Decker, J. Park, L. Sauermann, S. Auer, and S. Handschuh (Eds.), *Proc. of the Semantic Desktop and Social Semantic Collaboration Workshop*, Volume 202 of *CEUR Workshop Proceedings ISSN 1613-0073*. CEUR-ws.org. `http://CEUR-WS.org/Vol-202/SEMDESK2006_0010.pdf` (accessed 06.01.2010).

Oren, E., M. Völkel, J. G. Breslin, and S. Decker (2006, September). Semantic wikis for personal knowledge management. In *Database and Expert Systems Applications*, Volume 4080/2006, pp. 509–518. Springer Berlin / Heidelberg.

Paoli, J., J. Cowan, T. Bray, F. Yergeau, E. Maler, and C. M. Sperberg-McQueen (2006, August). Extensible markup language (XML) 1.1 (second edition). W3C recommendation, W3C. http://www.w3.org/TR/2006/REC-xml11-20060816.

Park, J. and A. Cheyer (2006). *Charting the Topic Maps Research and Applications Landscape*, Chapter Just for Me: Topic Maps and Ontologies, pp. 145 – 159. Secaucus, NJ, USA: Springer-Verlag New York, Inc.

Pédauque, R. T. (2003, July). Document: Form, sign and medium, as reformulated for electronic documents.

Peirce, C. S. (1931-1958). *Collected Papers of Charles Sanders Peirce, 8 vols*. Harvard University Press, Cambridge, Massachusetts.

Pemberton, S. (2000, January). XHTML^{TM} 1.0: The extensible hypertext markup language - a reformulation of HTML 4 in XML 1.0. first edition of a recommendation, W3C. W3C Recommendation 26 January 2000, revised 1 August 2002.

Peter, H., H. Sack, and C. Beckstein (2006). Tags and dependencies: an integrated view of document annotation. In *in Proc. of the 1st Semantic Authoring and Annotation Workshop (SAAW2006)*, Athens (GA), USA.

Phelps, T. A. and R. Wilensky (2000). Multivalent documents. *Communications of the ACM 43*(6), 82–90.

Pirolli, P. and S. K. Card (1995). Information foraging in information access environments. In *CHI*, pp. 51–58.

Polanyi, M. (1958). *Personal Knowledge: Towards a Post-Critical Philosophy*. London: Routledge & Kegan Paul Ltd.

Polanyi, M. (1966). *Tacit Dimension*. London: Routledge & Kegan Paul Ltd.

Polanyi, M. (1998, March). *Personal Knowledge*. Routledge.

Pollard, D. (2005, November). Personal knowledge management (PKM) – an update. Dave Pollard's environmental philosophy, creative works, business papers and essays (Blog). `http://blogs.salon.com/0002007/2005/11/23.html` (accessed 05.01.2010).

Pras, A. and J. Schoenwalder (2003, January). On the difference between information models and data models. Informational 3444, Network Working Group.

Probst, G., S. Raub, and K. Romhardt (2006). *Wissen Managen: Wie Unternehmen ihre wertvollste Ressource optimal nutzen* (5 ed.). Gabler Verlag.

Prud'Hommeaux, E., A. Seaborne, A. Seaborne, and E. Prud'hommeaux (2007, June). SPARQL query language for RDF. W3C recommendation, W3C. `http://www.w3.org/TR/rdf-sparql-query/` (accessed 05.01.2010).

Quan, D., D. Huynh, and D. R. Karger (2003). Haystack: A platform for authoring end user semantic web applications. In *Proc. of ISWC 2003*, pp. 738–753.

Raggett, D., A. L. Hors, and I. Jacobs (1999, December). HTML 4.01 specification. W3C recommendation, W3C. http://www.w3.org/TR/1999/REC-html401-19991224.

Reinmann, G. and M. J. Eppler (2007). *Wissenswege*. Huber, Bern.

Renear, A., E. Mylonas, and D. Durand (1993, January). Refining our notion of what text really is: The problem of overlapping hierarchies. online. A slightly edited version of this paper was published in 1996 in Research in Humanities Computing, Oxford University Press, Nancy Ide and Susan Hockey, eds.

Reynolds, D., S. Cayzer, I. Dickinson, and P. Shabajee (2001). Semantic web applications – analysis and selection. 12.1 Application Survey, SWAD-Europe project consortium (`http://www.w3.org/2001/sw/Europe/`, accessed 05.01.2010).

Rittel, H. and M. Webber (1973). Dilemmas in a general theory of planning. *Policy Sciences 4*, 155–169.

Rockley, A. (2002, October). *Managing Enterprise Content: A Unified Content Strategy*. New Riders Press.

Rubin, D. L., N. F. Noy, and M. A. Musen (2007, November). Protégé: A tool for managing and using terminology in radiology applications. *Journal of Digital Imaging 20*, 34–46.

Rutledge, P.-A. and G. Bajaj (2006, December). *Special Edition Using Microsoft Office PowerPoint 2007*. Que.

Sauer, C., C. Smith, and T. Benz (2007). Wikicreole:: a common wiki markup. In *WikiSym '07: Proceedings of the 2007 international symposium on Wikis*, New York, NY, USA, pp. 131–142. ACM Press.

Sauermann, L. (2009, June). *The Gnowsis Semantic Desktop approach to Personal Information Management*. Ph. D. thesis, Fachbereich Informatik der Universität Kaiserslautern.

Sauermann, L., R. Cyganiak, and M. Völkel (2007, February). Cool URIs for the semantic web. Technical Memo TM-07-01, Deutsches Forschungszentrum für Künstliche Intelligenz GmbH, Germany.

Schmidt, A. (2009). *Situationsbewusste Informationsdienste für das arbeitsbegleitende Lernen*. Karlsruhe, Germany: Karlsruhe Institute of Technology / Universität Karlsruhe (TH). http://digbib.ubka.uni-karlsruhe.de/volltexte/1000012939.

Schnase, J. L., J. J. Leggett, D. L. Hicks, P. J. Nürnberg, and J. A. Sánchez (1993, June). Design and implementation of the HB1 hyperbase management system. *Electronic Publishing - Origination, Dissemination and Design (EPodd) 6*(1), 35–63.

Schnetzler, N. (2004). *Die Ideenmaschine : Methode statt Geistesblitz - wie Ideen industriell produziert werden* (2. Aufl. ed.). Wiley-VCH.

Schreiber (1993, May). *KADS: A Principled Approach to Knowledge-Based System Development (Knowledge-Based Systems)* (1 ed.). Academic Press.

Schreiber, G., H. Akkermans, A. Anjewierden, R. Dehoog, N. Shadbolt, W. Vandevelde, and B. Wielinga (1999, December). *Knowledge Engineering and Management: The CommonKADS Methodology*. The MIT Press.

Schreiber, G. and M. Dean (2004, February). OWL web ontology language reference. Recommendation, W3C. http://www.w3.org/TR/2004/REC-owl-ref-20040210/ (accessed 05.01.2010).

Schreiber, T. and K. Harbo (2004). Information literacy and personal knowledge management. In *12th Nordic Conference on Information and Documentation*, pp. 106–114.

Schütt, P. (2003). The post-nonaka knowledge management. *Journal of Universal Computer Science 9*(6), 451–462.

Scriven, M. (1991, August). *Evaluation Thesaurus* (4 ed.). Sage Publications, Inc.

Shipman, F. M. and R. J. McCall (1999). Incremental formalization with the hyper-object substrate. *ACM Trans. Inf. Syst. 17*(2), 199–227.

Shneiderman, B. (1989). *The Society of Text*, Chapter Reflections on authoring, editing, and managing hypertext, pp. 115–131. Cambridge, MA, USA: MIT Press.

Shneiderman, B. (1996, September). The eyes have it: A task by data type taxonomy for information visualizations. In *VL '96: Proceedings of the 1996 IEEE Symposium on Visual Languages*, Washington, DC, USA. IEEE Computer Society.

Shneiderman, B. (1998, May). *Designing the User Interface*. Addison Wesley.

Simon, H. A. (1981, May). Studying human intelligence by creating artificial intelligence. *American Scientist 69*, 300–309.

Sintek, M., L. van Elst, S. Scerri, and S. Handschuh (2007). Knowledge representation on the social semantic desktop: Named graphs, views, and roles in nrl. In W. M. Enrico Franconi, Michael Kifer (Ed.), *The Semantic Web: Research and Applications - Proceedings of 4th European Semantic Web Conference ESWC 2007*, LNAI, Springer, pp. 594–608. 4519.

Skuce, D. and T. C. Lethbridge (1995, April). Code4: a unified system for managing conceptual knowledge. *International Journal of HumanComputer Studies 42*(4), 413–451.

Sowa, J. F. (1976). Conceptual graphs for a data base interface. *IBM Journal of Research and Development 20*(4), 336–357.

Srinivasan, B. and J. Zeleznikow (Eds.) (1990). *Databases in the 1990s: Proceedings of the Australian Database Research Conference*. World Scientific.

Staab, S. and R. Studer (Eds.) (2009). *Handbook on Ontologies* (2nd ed. ed.). International Handbooks on Information Systems. Springer.

Staab, S., R. Studer, H.-P. Schnurr, and Y. Sure (2001, Jan-Feb). Knowledge processes and ontologies. *Intelligent Systems, IEEE 16*(1), 26–34.

Stachowiak, H. (1973). *Allgemeine Modelltheorie*. Springer, Wien.

Stankosky, M. (2005, February). *Creating the Discipline of Knowledge Management: The Latest in University Research*. Butterworth-Heinemann.

Studer, R., V. R. Benjamins, and D. Fensel (1998). Knowledge engineering: Principles and methods. *Data & Knowledge Engineering 25*(1-2), 161 – 197.

Sweller, J. (1988). Cognitive load during problem solving: Effects on learning. *Cognitive Science 12*(2), 257 – 285.

Taboada, M. and W. C. Mann (2006). Rhetorical structure theory: Looking back and moving ahead. *Discourse Studies*, 423–459.

Taylor, F. W. (1911). *The Principles of Scientific Management*. New York and London: Harper & Brothers.

Tazzoli, R., P. Castagna, and S. E. Campanini (2004). Towards a Semantic WikiWikiWeb. In *3rd International Semantic Web Conference (ISWC2004)*, Hiroshima, Japan.

Teevan, J., C. Alvarado, M. S. Ackerman, and D. R. Karger (2004). The perfect search engine is not enough: a study of orienteering behavior in directed search. In *CHI '04: Proc. of the SIGCHI conf. on Human factors in computing systems*, pp. 415–422. ACM Press.

Tergan, S.-O. (2005). Digital concept maps for managing knowledge and information. In S.-O. Tergan and T. Keller (Eds.), *Knowledge and Information Visualization*, Volume 3426 of *Lecture Notes in Computer Science*, pp. 185–204. Springer.

Thompson, H. S., C. M. Sperberg-McQueen, and S. Gao (2008, June). W3C XML schema definition language (XSD) 1.1 part 1: Structures. a WD in last call, W3C. http://www.w3.org/TR/2008/WD-xmlschema11-1-20080620/.

Toffoli, T. (2002, January). A knowledge home - personal knowledge structuring in a computer world. online. Draft 5.00.

Toms, E. G. (2000). Serendipitous information retrieval. In *DELOS Workshop: Information Seeking, Searching and Querying in Digital Libraries*.

Unicode (2007). *The Unicode Standard, Version 5.0.0*. Addison-Wesley, Boston, MA.

Uschold, M. (1996). Building ontologies: Towards a unified methodology. In *In 16th Annual Conf. of the British Computer Society Specialist Group on Expert Systems*, pp. 16–18.

van Harmelen, M. (2006). Personal learning environments. In *ICALT '06: Proceedings of the Sixth IEEE International Conference on Advanced Learning Technologies*, Washington, DC, USA, pp. 815–816. IEEE Computer Society.

Van Rijsbergen, C. J. (1979). *Information Retrieval, 2nd edition*. Dept. of Computer Science, University of Glasgow.

Varian, H. R. (1999). The economics of search. In *SIGIR '99: Proceedings of the 22nd Annual International ACM SIGIR Conference on Research and Development in Information Retrieval*, New York, NY, USA, pp. 1. ACM.

Vester, F. (2000). *Die Kunst vernetzt zu denken* (5. Aufl. ed.). DVA.

Völkel, M. (2005a, October). Semwiki - a restful distributed wiki architecture. In *Proceedings of the First International Symposium on Wikis*, San Diego, USA.

Völkel, M. (2005b, December). Writing the semantic web with java. Technical report, DERI Galway, Ireland. Copy available at `http://www.xam.de/2005/12_voelkel_semweb4j_DERISem05.pdf` (accessed 05.01.2010).

Völkel, M. (2006, May). RDFReactor – from ontologies to programmatic data access. In *Proc. of the Jena User Conference 2006*. HP Bristol.

Völkel, M. (2007a, March). From documents to knowledge models. In N. Gronau (Ed.), *Proc. of the Workshop on Productive Knowledge Work: Management and Technological Challenges (ProKW2007) at the 4th Conference Professional Knowledge Management, Potsdam*, Volume 2, Berlin. GITO.

Völkel, M. (2007b, September). A semantic web content model and repository. In *Proceedings of the 3rd International Conference on Semantic Technologies (I-SEMANTICS), Graz, Austria*, pp. 254–261.

Völkel, M. (2008, June). Hypertext knowledge workbench. In C. Lange (Ed.), *Proc. of the Third Workshop on Semantic Wikis – The Wiki Way of Semantics – Workshop at ESWC, Budva, Montenegro*.

Völkel, M. and A. Abecker (2008, June). Cost-benefit analysis for the design of personal knowledge management systems. In J. Cordeiro and J. Filipe (Eds.), *ICEIS 2008 - Proceedings of the Tenth International Conference on Enterprise Information Systems, Volume AIDSS, Barcelona, Spain, June 12-16, 2008*, pp. 95–105.

Völkel, M. and T. Groza (2006, October). SemVersion: An RDF-based ontology versioning system. In *Proceedings of IADIS International Conference on WWW/Internet*, Volume 1, Murcia, Spain, pp. 195–202. IADIS: IADIS.

Völkel, M. and H. Haller (2006, July). Conceptual data structures (CDS) – towards an ontology for semi-formal articulation of personal knowledge. In *Proc. of the 14th International Conference on Conceptual Structures 2006*, Aalborg University - Denmark.

Völkel, M. and H. Haller (2009). Conceptual data structures for personal knowledge management. *Online Information Review 33*(2), 298–315.

Völkel, M., H. Haller, and A. Abecker (2007, October). Modelling higher-level thought structures - method and tool. In *Proceedings of Workshop on Foundations and Applications of the Social Semantic Desktop at eChallenges*.

Völkel, M., M. Krötzsch, D. Vrandecic, H. Haller, and R. Studer (2006, May). Semantic Wikipedia. In *Proceedings of the 15th International Conference on World Wide Web, WWW 2006, Edinburgh, Scotland, May 23-26, 2006*.

Völkel, M. and E. Oren (2006, May). Towards a wiki interchange format (WIF). In M. Völkel and S. Schaffert (Eds.), *Proceedings of the First Workshop on Semantic Wikis – From Wiki To Semantics, ESWC, Budva,Montenegro*.

Völkel, M., E. Oren, and S. Schaffert (2008, June). *Personal Knowledge Management with Semantic Technologies*. Idea Group Inc.

Völkel, M. and S. Schaffert (Eds.) (2006, May). *Proceedings of the First Workshop on Semantic Wikis at ESWC, Budva, Montenegro*, Volume 206 of *CEUR Workshop Proceedings ISSN 1613-0073*. FZI Forschungszentrum Informatik Karlsruhe: CEUR-ws.org. `http://CEUR-WS.org/Vol-206` (accessed 06.01.2010).

Völkel, M. and Y. Sure (2005, November). RDFReactor - from ontologies to programmatic data access. Poster and Demo at International Semantic Web Conference (ISWC) 2005, Galway, Ireland.

Walsh, N. and L. Muellner (1999, October). *DocBook: The Definitive Guide (O'Reilly XML)*. O'Reilly Media, Inc.

Wiil, U. K. (2005). Hypermedia technology for knowledge workers: a vision of the future. In *HYPERTEXT '05: Proceedings of the sixteenth ACM Conference on Hypertext and Hypermedia*, New York, NY, USA, pp. 4–6. ACM Press.

Yergeau, F. (1998, January). UTF-8, a transformation format of ISO 10646. Technical Report 2279, Internet Engineering Task Force, Network Working Group. Obsoleted by RFC 3629.

Zhang, J. and D. A. Norman (1994). Representations in distributed cognitive tasks. *Cognitive Science 18*(1), 87 – 122.

Zipf, G. K. (1949). *Human Behaviour and the Principle of Least-Effort*. Cambridge MA: Addison-Wesley.

List of Figures

List of Tables

Glossary

CDS ContentItem (page 129) represents a piece of textual content. The text can be structured using STIF.

CDS data model ($\mathfrak{D}$, page 124) is a flexible yet simple data-model.

CDS Item (page 126) is an addressable entity in a knowledge model.

CDS NameItem (page 129) is both a symbol for the computer (via its URI) and a term of the user's vocabulary (via its content).

CDS Relation (page 131) is the name for the type of ITEMS that may be used as the relation of a TRIPLE.

CDS relation ontology ($\mathfrak{R}$, page 143) is a top-level relation ontology for personal knowledge models. It contains the most commonly used relations found in tools used for PKM tasks.

CDS Statement (page 132) is an ENTITY which connects ITEMS. Each STATEMENT plays a dual role as addressable ITEM and as a TRIPLE, connecting other ITEMS.

CDS Triple (page 130) is a labelled link. Depending on the used RELATION, it can as well represent a formal statement.

Conceptual Data Structures (CDS) (Ch. 4) is a meta-model for representing digital knowledge cues in a uniform fashion that are in different degrees of formalisation. It consists of three parts: A data-model ($\mathfrak{D}$), a relation ontology ($\mathfrak{R}$), and an interchange format for structured text (STIF).

Degree of formality (page 26); similar: formality continuum, hybrid formal-pragmatic specifications, degrees of formal and informal knowledge, formality spectrum

Knowledge cue (page 20); any kind of symbol, pattern or artefact, created with the intent to be used by its creator, to re-evoke a previously experienced mental state (activated knowledge), when viewed or used otherwise.

Personal (digital) knowledge model (page 23) is a digital artefact which represents a set of knowledge cues. The knowledge cues can vary in size, structuredness and degree of formality. A knowledge cue is either (a) a piece of content, containing plain text, semi-structured text, or arbitrary binary content such as images or desktop objects, or (b) a connection between other knowledge cues. Such connections can be unspecified relations, directed hyperlinks and formal statements.

Personal Knowledge Management (page 14) investigates the use of methods and tools to amplify the abilities of the individual to work better with knowledge.

Structured Text Interchange Format (STIF, page 153) allows representing and using structured text across different applications.

The End.