Hypertext Knowledge Workbench*

Max Völkel

FZI Forschungszentrum Informatik
Universität Karlsruhe (TH), Germany
voelkel@fzi.de, http://www.fzi.de/ipe/

Abstract. This paper presents a tool for semantic personal knowledge management called *Hypertext Knowledge Workbench* (HKW), an editor and browser for semantic personal knowledge models. The tool is designed to be used by a single person to manage her personal notes about any topic that seems relevant. Existing wikis and semantic wikis represent content as pages with a title and content. Hypertext-based Knowledge Workbench (HKW) offers a more powerful yet simple to use conceptual model and allows entering and using knowledge in different degrees of granularity and formality.

1 Introduction

In 1958, Peter F. Drucker [8] 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.

Frand and Hixon [10] were among the first to use the term Personal Knowledge Management (PKM) in an academic context, followed by [2, 18]. Higgison [12] defines personal knowledge 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".

Polanyi [24] makes a distinction between explicit knowledge encoded in artefacts such as books or web pages, and tacit knowledge which resides in the individual. Concerning *explicitness* of knowledge, Nonaka and Takeuchi [20] distinguish two kinds of explicitness: explicit and tacit. Later works [6, 19] conclude that external and internal (tacit) knowledge are two extremes on a spectrum. Maurer [17] 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. In PKM, we often deal 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

* Acknowledgments: Research reported in this paper has been financed by the EU in the Social Semantic Desktop project NEPOMUK (IST-FP6-027705, http://nepomuk.semanticdesktop.org).

again to re-activate the knowledge in the individuals 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 such a 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 the latest source code management tool called coffee". In HKW, knowledge can be represented and organised within each note as well as by connections among notes.

HKW is intended to serve as a personal log book for all kinds of knowledge. E. g. it can be used to record ideas, bookmarks, text modules for or from documents, contact data of persons, notes on sport exercises, cocktail recipes, places to travel to, list of friends who enjoy Chinese cuisine, favoured artists and their interrelationships, nice restaurants, etc. These items often have relations that cannot be represented in specialised tools, e. g. friends living in the cities where I know a nice restaurant; text modules uses in which documents?; idea resulted in which publication?; who knows more about this topic?; who has a record from this rare music artist? The user can decide whether she represents e. g. a relation between a person and a music artist as a tag, a link, typed link or just in natural language. HKW is designed to relief the user from deciding "Will it be worth the effort to take this note?" and "Where do I file this?" Authoring in HKW is "pay as you go", with very low initial costs yet expressive modelling abilities in the same tool. A user can record any thought at low costs and *might* develop it later into more structured, more linked, more important notes. HKW can represent structured knowledge from any domain, because the relations and types *can* be changed and extended by the user at runtime. Informal knowledge represented as plain text or structured text needs no defined types and relations.

1.1 Existing Approaches

Existing approaches to personal note management are **paper-based approaches** such as sticky-notes, paper notebooks, and a "Zettelkasten" [16]. The paper-based approaches are hard to automate, e. g. in a Zettelkasten one has to traverse the links from note card to note card manually. 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.

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 [3] point out, there is also a need to organize notes so that a note is even found if the user was just querying for a related note or browsing to a certain folder or category. On their desktop, users manage their content in files; these files are organized in folders. 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.

Software tools designed for knowledge management offer ways to create links between notes. Such tools include blogs, wikis, PIM tools, mind map tools, and desktop wikis. These tools offer better usability for quickly creating structured content and linking it with exiting notes. However, if a knowledge base grows to thousands of notes, mere store and retrieval reaches its limits. E. g. which of your recipes were tasty when you tried but do not contain too much chilli, because Dirk does not like chilli? You want to go to Heidelberg – what was the name of the nice restaurant you went to together with Claudia last summer? Or imagine you keep a diary where you occasionally note your running times. Suddenly you weight more than the year before. Did you went running less often than the year before? Most real-world use-cases are not restricted to single domains, rather it is typical to have links across multiple domains (here: recipes, people, travel, sports).

It is obvious that a system can support searching and browsing better, the more structured, more formal the content is. However, requiring to formalise *all* content is too expensive. Therefore plain ontology editors are not good PKM tools. A system should let the user decide at each step how much formalisation effort to put into the system. For numerical knowledge, spreadsheet applications such as Microsoft Excel, are an example for the *ability* to formalise knowledge. A user can either use a spreadsheet just like a sheet of paper or link different cells into complex formulas. In a similar way, semantic wikis allow the same flexibility for conceptual knowledge. A user *can*—but is not forced to—formalise link and page types.

Semantic wikis are designed and used not only for collaborative use but also for personal knowledge management (PKM) [21, 23]. Semantic wikis do 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.

The **Hypertext Knowledge Workbench (HKW)** is different from semantic wikis. HKW (a) is backed by a more flexible data model, (b) allows to create and *change* formal statements easily, and (c) integrates authoring, structuring and formalisation.

1.2 Outline

The remainder of this paper analyzes the conceptual model of wikis and semantic wikis and compare it to the conceptual model of HKW, called Conceptual Data Structures (CDS) (Sec. 2). In Sec. 3 we analyse the annotation abilities of CDS compared to semantic wikis. We report on the HKW user interface in Sec. 4. In Sec. 5 we compare CDS and HKW to related work before we conclude in Sec. 6.

2 Conceptual Models

A conceptual model is not to be confused with a technical data model. As an analogy, compare the technical level of the file system, which consists of nodes, blocks, node tables and file allocation tables, with the user-perceived model: A strict tree containing folders and files.

Wikis have a simple conceptual model: Each page has a name, which is a short string that can be typed on a keyboard and often 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.

In **semantic wikis**, e. g. Semantic MediaWiki [14] (SMW), the user can state link *types* and use a part of the page content itself as target of the semantic links. However, it is not possible to link to these content snippets with another link. The content snippets in SMW are not first-class entities.

The conceptual model of HKW — called **Conceptual Data Structures** (CDS) — is a generalisation of that model. CDS [26, 27] consists of two layers: The CDS data model and the CDS relation hierarchy. The semantics of CDS re-use some of the semantics of RDFS [11]. Basically CDS uses sub-classes and sub-properties, extended with inverse properties. The next two sections describe the CDS data model and relation hierarchy. The complete CDS framework has been implemented as Java API, available from `http://cds.xam.de`.

2.1 CDS Data Model

The CDS data model[1] is a technical data model to represent knowledge. However, most parts are *intended* to be directly exposed to the user as a conceptual model.

The conceptual model consists of six primitive types. Fig. 1 shows the technical data model with the conceptual parts shown in bold. We describe briefly how the conceptual model works from a user's perspective. This conceptual model is exposed to the user in HKW.

Model A *Model* can be opened or saved, just like other documents. A *Model* is a container for items. Such items might be items with content (i. e. *NameItems*, *ContentItems*, *Relations*, or *Statements*) or automatically appearing triples.
Under the hood: Each *Model* has a URI. In RDF, each model is represented as a Named Graph [4].

ContentItem These are simple text snippets, like the content of a wiki page or a sheet of paper. One can write anything into such a note. The system records automatically creation date, change date and author. One may use wiki syntax to format the note, or link to *NameItems* by referring to their

[1] It is the successor of the "Semantic Web Content Model" (SWCM) presented in [25]

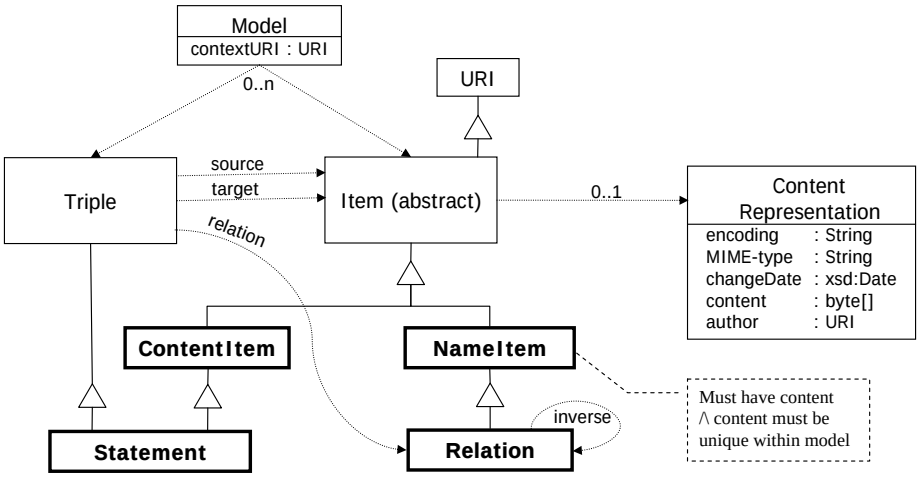

Fig. 1. CDS data model

name. Like a wiki page, a *ContentItems* content size can range from very short to very long. It may also be the case that a *ContentItem* has no content. *Under the hood:* Each item (i. e. *NameItems*, *ContentItems*, *Relations*, and *Statements*) has a unique URI to reference it. Even if the content of an item changes, the references remain the same. No content can appear outside of *Items*. Each piece of content is thus addressable, which makes it easier to record metadata and introduce versioning. Currently there is no versioning of content. The representation of content is modelled after resources on the web (c. f. [9]). All metadata is represented in Resource Description Framework (RDF), binary content is stored in a separate content repository. RDF can store binary data only inconveniently as `xsd:base64Binary` types. Current triple stores are not designed to handle large byte streams.

NameItem A *NameItem* is just a name. It is like the title of a wiki page or the name of a file within a folder. Like in a wiki, a *NameItem* must be unique within a model. *NameItems* do not have any content (besides their name) but it is easy to create links to other items. *NameItems* allows a user to jump directly to certain entities in the knowledge model, similar to navigating to a known wiki page. Other items can be reached indirectly through search or browse actions.

Under the hood: Representing names as first class citizens is handy to allow a user to rename e. g. a *NameItem* without having to change each *Statement* using it. A *NameItem* has three restrictions on its content: First, a *NameItem* has always exactly one content attached to it. Second, a *NameItem* may have only simple textual content, i. e. no line breaks and no wiki syntax. The content of a *NameItem* can easily be entered a human (possibly using an auto-completion mechanism). The MIME-type of the content is always “text/plain”. Third, this textual content must be *unique* within a

knowledge model: No two *NameItems* can have different URIs and the same content. Formally, for two *NameItems* n_1 and n_2 the following holds:

$$n_1.content = n_2.content \Leftrightarrow n_1 = n_2 \tag{1}$$

All these constraints do not hold for *ContentItems*: They can be empty or two items can have the same content. There may also be *ContentItems* having the same content as a *NameItem*.

Relation A *Relation* describes the way two items are linked to each other. There are many pre-defined relations, but one can create new relations as needed. The built-in relation hierarchy is described in the next section. Each relation always has an inverse relation defined, so one can view each link from both sides. E. g. "Dirk knows Claudia" is the same as "Claudia is known by Dirk", if "knows" has the inverse "is known by".

Under the hood: A *Relation* is a special kind of *NameItem*. This implies there can not be two different relations having the same name. Each *Relation* p has a *mandatory inverse* Relation $-p$. The inverse of the inverse of a *Relation* p is again p:

$$-(-p) = p \tag{2}$$

In CDS each statement of the form (s, p, o) can additionally be rendered as $(o, -p, s)$ with $-p$ being the inverse of p.

Triple A CDS *Triple* is like a semantic link in a wiki. It connects any two items and denotes the type of the link by a *Relation*. Triples appear in the user interface e. g. as the result of queries using inferencing.

Under the hood: Triples are not items. They have no metadata or content attached and a user has first to promote the triple to a *Statement* before it can be annotated.

Statement A *Statement* is both a *Triple* and an *Item*. As such, it also has a creation date, an author and may even be annotated or have textual content. Annotating statements is useful to state the source of knowledge e. g. in discussion systems.

There are two ways to create *Statements*. First, a user can use the name of a *NameItem* in the text of a *ContentItem*. Then the system automatically creates a statement, where the originating *ContentItem* is recorded as the author. This allows the user to back-trace the origin of a statement. Second, the user can directly create a *Statement* between any kind of item.

Under the hood: Statements are represented on RDF via a kind of reification. Different from RDF, each CDS *Statement does* entail the ground triple (s, p, o). For every *Statement* (s, p, o), the inverse *Statement* $(o, -p, s)$ is inferred, where $-p$ is the inverse of p. That is:

$$\forall s, p, o : (s, p, o) \mapsto (o, -p, s) \tag{3}$$

Note that the URI of the *Statement* does not influence the asserted facts. It is possible that different statements with the same URI assert the same facts but e. g. having different annotations. *Statements* with the *same* URI must have the same content, i. e. the same source, relation and target.

2.2 CDS Relation Hierarchy

On top of this conceptual data model, CDS defines a hierarchy of relations.

The CDS built-in relations have been selected after an analysis of a number of existing information structures in applications used for PKM. The core relation types deal with order, hierarchy, different forms of annotation (i. e. free-text annotations, tagging, and formal typing), and generic hyper-links. As the relation hierarchy is represented in the CDS data model, the user can (and should) extend it in CDS-based tools such as HKW.

The relation hierarchy itself is represented by the built-in relation `cds:hasSubRelation`. Each lower-level *Relation* implies the higher-level *Relations*, just like in RDF Schema (RDFS). The complete relation hierarchy is described in [28].

3 Semantic Annotations

In semantic wikis, users can usually either state the type of link or embed metadata about the current page using special syntax constructs. Some wikis (i. e. SemperWiki) allow embedding arbitrary RDF statements on a page). There is a high variance between the capabilities of semantic wikis to create semantic data. [22] compares some popular semantic wikis with respect to their ability to create annotations. We now analyse HKW with the dimensions given in [22]: Attribution, granularity, representation distinction, terminology reuse, object type, and context.

Attribution Most wikis attribute their annotations to the page where the user is editing the wiki syntax. In HKW (and CDS) links are external to the items. Like in other semantic wikis it is possible to create links between entities via syntax constructs. Internally, such syntax constructs are parsed and result in generated statements. However, only HKW allows creating links which do no originate from a wiki page. This allows e. g. easily importing an existing ontology into the knowledge base without the need to append generated wiki syntax to existing text. Furthermore, each *Statement* in HKW is an *Item* itself and each *Item* in HKW has an author and a creation date. This allows recording the provenance of *Statements* conveniently. The downside of this extended flexibility is versioning. Existing semantic wikis where all semantic statements originate in wiki syntax, the page-based wiki versioning is re-used. In HKW, this is not possible. In fact, HKW has currently no versioning. In the future, we plan to add two kinds of versioning: Item-level versioning for the textual content and model-level versioning for the semantic statements.

Granularity HKW allows creating *ContentItems* of varying size, ranging from single words to full documents, just like the page content of a wiki page. However, different from wiki pages, *ContentItems* have no name, therefore it is cognitively easier to create a large set of them: Imagine an author of a long document would have to name each paragraph in the document individually!

In wikis, every entity that one wants to link to must be written on its own wiki page.

In HKW, the amount, size and relation between *NameItems* and *ContentItems* can be chosen by the user. Therefore HKW can be used to mimic a classic wiki (with *NameItem*-*ContentItem* pairs), but can also be used in other ways, i. e. linking *ContentItems* with *ContentItems* and *NameItems* with *NameItems*.

Representation Distinction There have been long debates in mailing lists and workshops over the role of URIs that are used to locate information resources on the web and to denote abstract concepts. For practical everyday personal knowledge management tasks this distinction does not matter much. The individual users create their Items with URIs bound to a personal unique namespace, so there is no danger of accidental overlap. As we separate the *NameItems* from the *ContentItems*, they have different URIs. *NameItems* may contain only a short string. Line breaks and formatting are not allowed. This reduces *NameItems* more or less to labels (but unique ones). So there is the ambiguity whether one talks about the *NameItem* or the concept denoted by the *NameItem*. However, the same ambiguity is in our everyday life: Do we talk about the *name* "Dirk Hageman" or the *person* "Dirk Hageman" when we say "Dirk Hageman"? For pragmatic reasons HKW does not distinguish these two cases in the data model. Note that the information-resource-like *ContentItems* are distinguished from the name-like *NameItems*.

Terminology Reuse In HKW, the user is usually not confronted with URIs, so she cannot directly re-use existing URIs. There are two options around this: One is to create explicitly an *Item* with a given URI, another one is to import an existing ontology as a set of *NameItems*. The ontology needs either to have unique labels or labels have to be changed to become unique at import time.

Object Type Most semantic wikis link either to other wiki pages or literal values. In HKW, there is no such distinction. All textual content is addressable by URIs. So the object type is neither page nor literal but *Item*.

In the future, we will integrate the CDS API with the NEPOMUK backbone. This will allow the user to link any semantic desktop item with any other semantic desktop or CDS *Item* and vice versa.

Context As the annotations in HKW are stored as first-class citizens, provenance and context can be stored. E. g. for each *Item* the author and creation date are automatically recorded. In addition to that, each *Statement* can be annotated further by the user. Note that none of the semantic wikis analysed in [22] had a way to record context.

4 User Interface

Fig. 4 shows a screen-shot of the HKW GUI[2] focusing on the *NameItem* "Dirk Hageman". The screen-shot shows the auto-completion list after entering the letter "c". The screen is divided into seven colored areas. Below the "Dirk" item, HKW shows the *Items* related via the relation `cds:hasDetail`. E. g. the statement "Dirk Hageman"–"born in"–"Offenburg" is rendered here. This tells the user that 'born in" is a `cds:hasSubRelation` of `cds:hasDetail`. The inverse relation of `cds:hasDetail` is `cds:hasContext`. *Items* related to the selected *Item* via `cds:hasContext` are rendered above the "Dirk" item. The other colored boxes represent other CDS core relations. The GUI shows relations always in their most specific box. Items are only rendered in different boxes at the same time if the user assigned multiple super-relations to a relation. Behind the word

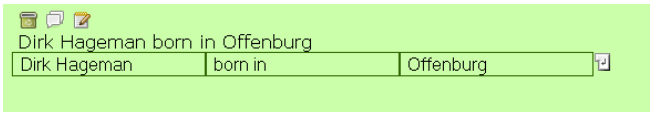

Fig. 2. Statement Widget

"Offenburg" there are icons allowing the user to navigate to the *Statement* "Dirk Hageman"–"born in"–"Offenburg". In a *Statement* view (c. f. Fig. 2), the *Statement* can be changed. E. g. the user can change the *Relation* or create a new source or target. Auto-linking is supported wherever possible. Most actions in

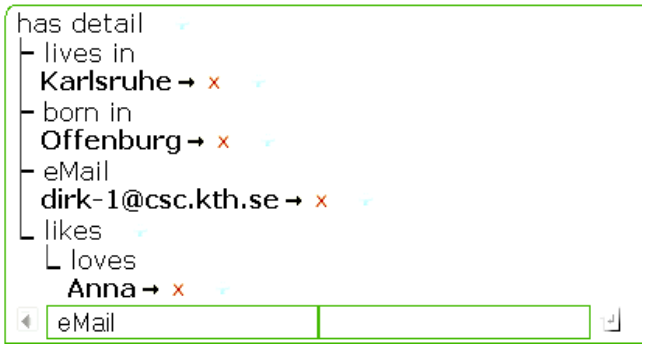

Fig. 3. Relation Tree Widget

HKW are performed in the *Relation Tree Widget* (c. f. Fig. 3). Each relation

[2] Try online or download from `http://wiki.ontoworld.org/wiki/CDS_Editor`

tree widget represents on of the CDS relations (detail, context, before, after, tag, type, annotation, annotation member, related, source, or target). The widget allows deleting existing statements by pressing the little red 'X'; creating new items, relations and corresponding statements. By pressing the blue plus icon next to an existing relation the widget expands and shows two form fields. One to enter a relation name, pre-filled with the relation where the blue plus was selected from and one form field to create a new item or select among the existing *NameItems*. The user is free to enter a different relation name into the relation field, again supported by auto-completion. At any time new relations can be created by simply typing in a new *Relation* name. The *Relation* is automatically a sub-relation of the main *Relation* of a box. I. e. creating a new *Relation* in the top right box ("has annotation") creates a sub-relation of "has annotation". Inverse *Relations* are automatically created and named "inverse of ...". The name can easily be change by the user in a single place. This allows creating new semantically interlinked items easily. If the user enters a longer text or uses line breaks, the system assumes the user creates a *ContentItem*. For short text, the system suggest existing *NameItems* or creates new ones. As a result, a user can always just start typing in the address bar, no matter whether a concept-like *NameItem* or a note-like longer *ContentItem* is going to be created.

The HKW prototype has been realised with the *Google Web Toolkit* (GWT), an open-source AJAX-enabled web user interface toolkit by Google Inc. Styling is done via *Cascading Style Sheets* (CSS). Due to CSS issues, the tool works currently only properly in the Firefox browser. This is not a problem as Firefox is available free of charge for all platforms. GWT applications are web-applications, which can run in any servlet container. The typical use case is to run the server locally on the desktop.

5 Related Work

A unified model for web content and semantic statements is presented in [13]. However, different from [13], the CDS model (i. e. *ContentItems*, *NameItems*, *Relations* and *Statements*) is specifically designed to be exposed to and understood by end-users. A model and system for a unified browsing and querying across document boundaries is presented in [7], but authoring is not considered. Systems similar to HKW include Artificial Memory [15] and Haystack [1]. Artificial Memory shares the idea of CDS to break documents up into small, interlinked parts to minimize redundancy and improve automated processing. But Artificial Memory does not allow the user to create non-structured sloppy entries. We believe letting the user decide how much effort to put into formalisation of a knowledge item is an important feature to keep the total cost of usage low. Haystack emphasizes rendering and linking of RDF-based entities, but lacks ways to author textual content intermingled with semantic facts.

6 Conclusions and Future Work

CDS lets the user express knowledge in the form of text (within an item), structure (structured text in items or structures between items) and formal statements (by using relations with defined semantics). In HKW, searches and navigation do not bring up long documents, but short fragments of text with its relations to other parts.

In the future, we will extend HKW to allow a user to convert a *ContentItem* structured with wiki syntax into a set of corresponding smaller *ContentItems*. This will lower the cost of creating *Items* even further. The reverse operation should also be possible: Merge a set of *Items* into a single *ContentItem*, as wiki syntax. This makes gradual formalisation of knowledge easier: First content is written into *ContentItems*, then these items are structured using wiki syntax, finally they are converted into many smaller *Items* that can further be annotated as needed.

Fig. 4. HKW prototype screen shot, focusing on *Dirk Hageman*

Bibliography

[1] Adar, E., Karger, D. R. and Stein, L. A. [1999], Haystack: Per-user information environments, *in* 'CIKM', ACM, pp. 413–422.

[2] Avery, S., Brooks, R., Brown, J., Dorsey, P. and O'Conner, M. [2001], Personal knowledge management: Framework for integration and partnerships, *in* 'Proc. of ASCUE Conf.'.

[3] Barreau, D. and Nardi, B. [1995], 'Finding and reminding: File organization from the desktop', *SIGCHI Bulletin* **27**(3), 39–43.

[4] Carroll, J. J., Bizer, C., Hayes, P. and Stickler, P. [2004], Named graphs, provenance and trust, Technical report, HP.

[5] Decker, S., Park, J., Quan, D. and Sauermann, L., eds [2005], *The Semantic Desktop – Next Generation Information Management & Collaboration Infrastructure*, Galway, Ireland.

[6] Despres, C. and Chauvel, D. [2000], *Knowledge Horizons: the present and promise of Knowledge Management*, Butterworth-Heinemann.

[7] Dittrich, J.-P. and Salles, M. A. V. [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', VLDB Endowment, pp. 367–378.

[8] Drucker, P. F. [1985], *Management: Tasks, responsibilities, practices (Harper & Row management library)*, Harper & Row.

[9] Fielding, R. T. [2000], Architectural styles and the design of network-based software architectures, PhD thesis, University of California, Irvine.

[10] Frand, J. and Hixon, C. [1999], 'Personal knowledge management : Who, what, why, when, where, how?', Speech. working paper.
URL: `http://www.anderson.ucla.edu/faculty/jason.frand/researcher/speeches/PKM.htm`

[11] Hayes, P. [2004], RDF semantics, Recommendation, W3C.
URL: `http://www.w3.org/TR/rdf-mt/`

[12] Higgison, S. [2005], 'Your say: Personal knowledge management', *Insight Knowledge* **7**(7).

[13] Immaneni, T. and Thirunarayan, K. [2007], A unified approach to retrieving web documents and semantic web data, *in* E. Franconi, M. Kifer and W. May, eds, 'ESWC', Vol. 4519 of *Lecture Notes in Computer Science*, Springer, pp. 579–593.

[14] Krötzsch, M., Vrandecic, D., Völkel, M., Haller, H. and Studer, R. [2007], 'Semantic wikipedia', *Journal of Web Semantics* . To appear.

[15] Ludwig, L. [2005], Semantic personal knowledge management, Technical Report D11.01_v0.01, DERI Galway.

[16] Luhmann, N. [1992], Kommunikation mit zettelkästen. ein erfahrungsbericht, *in* A. Kieserling, ed., 'Universität als Milieu', Kleine Schriften, Haux Verlag, Bielefeld, pp. 53–61. ISBN 3-925471-13-8.

[17] Maurer, H. [1999], The heart of the problem: Knowledge management and knowledge transfer, *in* 'Proc. ENABLE'99', Espoo-Vantaa Institute of Technology, pp. 8–17.

[18] Mitchell, A. [2005], 'The rise of personal km', *Inside Knowledge* **9**(1).

[19] Nonaka, I. and Konno, N. [1998], 'The concept of "ba": Building a foundation for knowledge creation', *California Management Review* **40**(3), 40–54.

[20] Nonaka, I. and Takeuchi, H. [1995], *The Knowledge-Creating Company : How Japanese Companies Create the Dynamics of Innovation*, Oxford University Press.

[21] Oren, E. [2005], SemperWiki: a semantic personal wiki, *in* [5].

[22] Oren, E., Delbru, R., Möller, K., Völkel, M. and Handschuh, S. [2006], Annotation and navigation in semantic wikis, *in* S. Schaffert and M. Völkel, eds, 'Proceedings of the First Workshop on Semantic Wikis - From Wiki to Semantics at the ESWC 2006'.

[23] Oren, E., Völkel, M., Breslin, J. G. and Decker, S. [2006], Semantic wikis for personal knowledge management, *in* 'Database and Expert Systems Applications', Vol. 4080/2006, Springer Berlin / Heidelberg, pp. 509–518.

[24] Polanyi, M. [1966], *Tacit Dimension*, Routledge & Kegan Paul Ltd, London.

[25] Völkel, M. [2007], A semantic web content model and repository, *in* 'Proceedings of the 3rd International Conference on Semantic Technologies'.
URL: *http://xam.de/2007/2007-05-voelkel-ISEMANTICS-swcm-CR.pdf*

[26] Völkel, M. and Haller, H. [2006], 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.

[27] Völkel, M., Haller, H. and Abecker, A. [2007], Modelling higher-level thought structures - method and tool, *in* 'Proceedings of Workshop on Foundations and Applications of the Social Semantic Desktop'.

[28] Völkel, M., Haller, H., Bolinder, W., Davis, B., Edlund, H., Groth, K., Gudjonsdottir, R., Kotelnikov, M., Lannerö, P., Lundquist, S., Sogrin, M., Sundblad, Y. and Westerlund, B. [2008], Conceptual data structure tools, Deliverable 1.2, nepomuk consortium.
URL: *http://nepomuk.semanticdesktop.org/xwiki/bin/download/IST/WebHome/D1.2_v10_CDS-Tools.pdf*