Modelling Higher-Level Thought Structures – Method & Tool[1]

Max VÖLKEL, Heiko HALLER, Andreas ABECKER
FZI Forschungszentrum Informatik,
Haid-und-Neu-Straße 10-14, 76131 Karlsruhe, Germany
Email: {voelkel,haller,abecker}@fzi.de Web: http://www.fzi.de

Knowledge articulation costs are the bottleneck for efficient Personal Knowledge Management (PKM). Current tools either allow to few structures and hence have to rely only on keyword searches in plain text, allow no associative browsing, and cannot infer new knowledge. Semantic modelling tools on the other hands are too cumbersome to use and force the user to formalise everything all the time – this is too costly in PKM usage.

Conceptual Data Structures (CDS) are what is found to be the largest common denominator of information structures used in common knowledge artefacts. CDS allow step-wise and gradual formalisation and representing the spectrum from informal notes up to formal ontologies.

This paper describes the CDS data model and ontology in detail and shows how CDS can largely be implemented with existing semantic web technologies.

1. Introduction

There is a wealth of methods and tools that facilitate our every-day *Personal Knowledge Management* (PKM). They range from hand-written paper notes to personal semantic wikis and from mere text processing or spreadsheet applications over special outlining tools to nice and colourful graphical mind-mapping applications. They all have one thing in common: They help the user externalise knowledge in a more or less structured way.

According to Nonaka and Takeuchi [1], externalisation is the articulation of tacit knowledge that resides in someone's mind and is often only vague, into explicit knowledge that can be communicated. The act of externalisation is one of the four conversions in their widespread model of the knowledge creating process, and it is the step that must come before any piece of knowledge can be externally stored or even processed by a computer.

Although explicit, externalised knowledge still varies largely in its degree of formalisation. It can be anything between a loose collection of keywords, a weakly structured text, an informal graph or hypertext up to highly structured knowledge representations and fully formalised ones like an ontology.

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. On the other hand, more structure requires a higher effort during creation in the first place. For many uses 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"). CDS allows to express and use these semi-formal semantics—as well as no or fully formalised

[1] This material is based upon works supported by the European Commission under the Nepomuk project FP6-027705.

semantics. Gradual transitions from no formal semantics up to fully formalised semantics are thus possible. This is the core feature of the CDS model.

2. Objectives

In this paper we present the motivation for and ideas behind *Conceptual Data Structures* (CDS), a lean vocabulary for incremental recording and step-wise formalisation of personal knowledge, first described in [2]. We found these structures to be inherent to most knowledge artefacts ranging from vague paper notes to highly structured documents. CDS is suitable for representing knowledge in various degrees of formalisation in a uniform fashion, allowing gradual migration.

One can distinguish between domain-specific data models such as the data model of e.g. Microsoft Outlook, which limits the user to speak about persons and their addresses. There is neither a way to state the relation to other persons, nor to represent music collections or relations from persons to other objects. This is exactly the characteristic of domain-speficic tool: They usually support data modelling in a given domain well but do not allow to extend the model or link to other objects in other data models. Domain-free data models such as the file system or the model behind mind map applications allow to model any domain, but not in a structured way. It is, e.g. not possible to export a set of persons created in e.g. a Mind Mapping application to an address book application. CDS is intended to unify dominant domain-free data models and allow at the same time to represent structured, domain-specific data in such a way that domain-specific semantics can be used for inference, search and export.

There are two levels of semantics that CDS intends to support. First, formal semantics can be used by the computer to infer new knowledge. This works only for semantics expressed within formal languages such as OWL or RDFS for which reasoning engines are already available. Second, many ad-hoc structures used in documents (e.g. colour and formatting) and mind maps (e.g. attached icons) carry semantics for humans but can not (yet?) be used for logical inference—unless mapped to existing formal construct. It is the aim of CDS to let the user profit from formal annotations more easily. CDS makes mapping of user-semantics to formal semantics easier by allowing the user to use less expressive, vaguer semantics. E.g. sometimes it is easy to say that two items are related but it is hard to say how. The CDS vocabulary is described in detail in Sec. 4.

2.1 Related Work:

Existing knowledge representation languages are very general (RDF, RDFS, Topic Maps, OWL, Conceptual Graphs) and feature few semantic relations suited to directly model and structure personal knowledge. Dealing with them requires quite a technical mindset, e.g. thinking about the distinction between literals, blank nodes and URIs (RDF, OWL). Existing vocabularies and ontologies (SKOS, IBIS) are too domain-specific to model arbitrary personal knowledge.

3. Business Case Description

The usual way to use CDS is: A user creates a number of text items, comparable to brainstorming with sticky notes on a whiteboard. Then she groups these snippets and connects them with arrows. Later she specifies these relationships by labelling the arrows. After a while, she might see that some items share common characteristics and assigns them to one or more types such as “Person”, “Idea” or “ToDo”. These types can be exploited for search, e. g. “give me all Persons in Karlsruhe”. Arcs are classified in a similar fashion and can be typed with Relations such as “knows” or “part of”.

3.1 Document Creation

Our society is heavily based on communication by documents, ranging from books to emails. But in practice, few documents are written directly in a text editor in a linear one-pass fashion. Instead, 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).

For each level of formalisation, different tools are suitable. But since no single tool is optimal for all stages, external knowledge—especially while being articulated and formalised—often goes a long way, traversing different media and tools while subsequently approaching its final structure.

The process of writing a document, as we have it in mind, could be like this: Text snippets are created and structured in a textual interface (like a Semantic Wiki). The structuring could take place by simple wiki syntax (e.g. for nested lists etc.) and a semantically enriched link syntax, to explicitly state CDS and other typed relations where desired by the advanced user. But even without these, simple document structures imply many structural decisions: e.g. order and hierarchy of each single line of text have been assigned. As a next step, the CDS data could be restructured and refined in a graphical user interface.

3.2 Personal Information Management (PIM)

PIM usually means managing contact information, appointments, to-do items and notes. In fact, CDS grew out of frustration encountered with the existing PIM toolscape and the inability to relate, link and describe PIM items. For example, one can currently not even relate a number of contacts, appointments and tasks to a project. In particular, plain text notes are in most PIM tools merely an unstructured, unrelated set of memo items. Wikis offer better linking abilities but come at the cost of not being able to represent specific types of items, such as appointment or contact data. Semantic wikis meet both requirements but are still to cumbersome to use, i.e. it is hard to get an overview of the emerging structure and refactoring is very costly.

4. Methodology

We analysed the structures inherent to common knowledge artefacts like documents, paper notes, hypertexts; structures exposed by common information management tools like file browsers (e.g. the Windows Explorer), mind-maps, concept maps, note-taking applications, and more advanced PKM tools like e.g. Haystack [3]; structures built-in to programming languages (e.g. Java Collection Framework) and content languages (e.g. XML, RDF and HTML); popular Web 2.0 sites for collaborative information organisation (e.g. del.icio.us[2] and flickr[3]).

We found that there is a relatively small set of types of structural relations between information items, which occur very often in all of these paradigms. We grouped these relation types according to different dimensions that they describe. Finally the more specific ones of these relations were subsumed under the more general ones depending on how generic they are. This hierarchy of relations is the basis of the CDS ontology.

[2] http://del.icio.us
[3] http://flickr.com

5. Technology Description

CDS consists of two parts: A data model (analogous to RDF) and the CDS core ontology expressed within the CDS data model. Data expressed within the CDS data model and aligned with the CDS core ontology is called a user model.

5.1 Requirements

The CDS data model is based mostly on the OMG meta-modelling standard with a mapping to the W3C standard *RDF* (Resource Description Framework [4]) in mind.
Thereby our main constraints were: a) The data model must allow to annotate (and therefore address) all of its elements, in order not to limit expressivity. b) The data model must allow giving things human-usable names. The success of wikis and their page naming scheme shows the importance of naming. c) As we cannot expect users to work on one modelling layer only, the data model has to allow meta-modelling. d) The CDS data model should profit from existing technology, hence the CDS data model maps to RDF will be integrated into the NEPOMUK[4] social semantic desktop project. This will allow the user to annotate, link and classify not only personal notes but also any desktop resource.

5.2 CDS data model

In CDS, the central object is the *Item*. Every Item has a *Unique Resource Identifier* (URI). This allows to uniquely address any entity. Items are like the nodes of a graph. They can have content, which is usually a short snippet of text. *Statements* are the arcs in such a graph and—as they are subtypes of items—can also have content. *Items* and *Statements* allow to represent node-and-arc diagrams, e.g. like concept maps. A *Model* contains a number of *Items*. *Items* may occur in several *Models*. A *NameItem* is an *Item*, whose content actually serves as its name, which means it has to be unique within each *Model* where it is used. URIs are too inconvenient to be handled by the user. The shift from using an *Item* with the content "X" to using a *NameItem* with the content (here: name) "X" can be seen as a first step toward formalisation: The user is building his or her own vocabulary.
The next formalisation step is possible by assigning *types* to *Items* by referring to any *NameItem* with the built-in relation *hasType*.

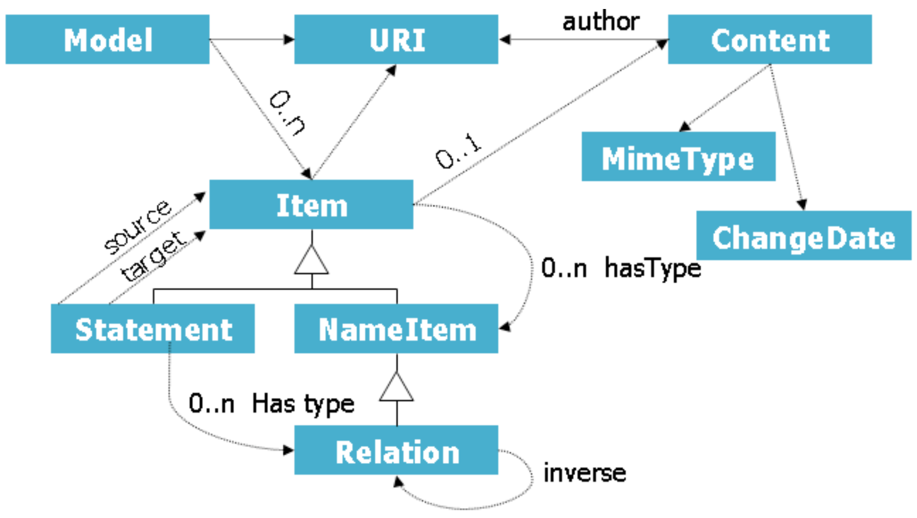

Figure 1: Complete CDS data model

[4] http://nepomuk.semanticdesktop.org

Fig. 1 shows the CDS data model. It has a number of extensions to allow real-world usage. We explain now each element in detail.

A *Model* is simply a container to group *Items*. It is important to have this concept in order to model different world views or simply different use cases. A *Model* is much like a file or database. A *Model* also has a URI, to make it easier to re-use CDS *Models* in RDF ModelSets, as defined by the SPARQL[5] specification or Named Graphs [5].

Content varies greatly in size and type. CDS must allow small content such as single words, sentences or short notes up to full-blown formatted documents. CDS must also allow modelling not only text but also other file types such as images or office file formats. The need for this mix of text and binary content can be observed in emails or wiki pages, which both can have binary attachments. CDS represents content as a bit sequence (not visible in the figure) and a mime-type, to allow decoding the bit sequence. Mime-types are an established means for web pages and emails to describe the semantics of a bit sequence.

NameItems are particular kinds of items where the mime-type of the content is always "text/plain", which means pure text. The content of a *NameItem* can easily be entered a human (possibly using an auto-completion mechanism). *NameItems* thus model a particular kind of content: A human-usable name. The CDS data model demands that all *NameItems* have exactly one *Content* (which is in fact the name) and that no two *NameItems* can have the same content (names are unique). Both constraints do not hold for normal *Items*: They can be empty or two items can have the same content. Note that *NameItems* represent really only the name itself. A wiki page, e.g. can be modelled as two things: A *NameItem* to hold the wiki page title and an *Item* to hold the wiki page content. A *Statement* connects the two parts.

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 is similar to wikis.

Comparing CDS with nodes and arcs, an arc is modelled as a *Statement*. Each *Statement* links a source *Item* with a target *Item*. Additionally, each *Statement* can have a type. CDS uses *Relations* as arc types. *Relation* is a special kind if *NameItem*. Each *Relation* has always an inverse *Relation*, to make it easier to browse and query the knowledge base.

Note that users can make *Statements* about *Items*, *NameItems*, *Relations* and other *Statements*. This allows full meta-modelling, which we exploit when designing the CDS ontology.

5.3 Mapping the CDS data model to RDF

CDS is implemented using the "*semantic web content repository" (swecr)*[6]. Swecr consists of three core components: An RDF store, a binary store (BinStore) and a full text index. The BinStore is much like a piece of WWW infrastructure, but running on the local desktop. It maps URIs to binary streams and mime-types. Additionally, the BinStore records author and creation date of the content.

A CDS model is stored partially in BinStore (*Content* of *Items*) and partially in RDF (*Items*, *NameItems*, *Relations* and *Statements*). Metadata of Content is also stored in RDF.

An Item is modelled as an rdfs:Resource, using the URI present in the CDS data model as the URI of the resource. As NameItems are expected to have rather short pieces of content, we store the content of NameItems also within the RDF model. We use two rdfs:Classes to model the types *Item* and *NameItem*. To represent *Statements*, we use the reification mechanism of RDF.

[5] http://www.w3.org/TR/rdf-sparql-query/
[6] http://wiki.ontoworld.org/wiki/Swecr

As an example, "Claudia" wants to express the facts that "Dirk" works at "SAP" and that "SAP" is located in "Karlsruhe". In CDS, she could use NameItems to represent Dirk, SAP and Karlsruhe, as they are unique concepts for here. Fig. 3 shows a CDS model representing this example. The same model, encoded in RDF is shown in Fig. 4.

```
[Dirk]             [works at]         [SAP].
[SAP]              [is located in]    [Karlsruhe].
[works at]         cds:hasInverse     [employs].
[is located in]    cds:hasInverse     [is location of].
```

Fig. 3: A simple CDS user mode)

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"

Make two *Statements*:

<s1>	rdf:type	rdfs:Statement
<s1>	rdf:subject	<dirk>
<s1>	rdf:property	<wa>
<s1>	rdf:object	<sap>
<s2>	rdf:type	rdfs:Statement
<s2>	rdf:subject	<sap>
<s2>	rdf:property	<locIn>
<s2>	rdf:object	<ka>

Define *Relations* and their inverse *Relations*:

<wa>	rdf:type	cds:Relation
<wa>	cds:hasContent	"works at"
<wa>	cds:hasInverse	<emp>
<emp>	rdf:type	cds:Relation
<emp>	cds:hasContent	"employs"

<locIn>	rdf:type	cds:Relation
<locIn>	cds:hasContent	"is located in"
<locIn>	cds:hasInverse	<locOf>
<locOf>	rdf:type	cds:Relation
<locOf>	cds:hasContent	"is location of"

Fig. 4: A CDS model represented in RDF (N3 Syntax)

5.4 CDS ontology

Whereas the CDS *data model* allows modelling content snippets (*Items*) and arbitrary relations between them, the CDS *ontology* defines a simple schema language to classify, relate and describe relations. It is designed to allow for soft migration from unstructured to structured knowledge. The user is free to create any *Relation* or *Item* types he needs. CDS only demands from the user to classify all relations according to the CDS ontology. This is not really a constraint however since unspecified relations can safely be made a sub-relation of the top-level relation cds:related. The CDS ontology is a taxonomy of *Relations*, each lower-level *Relation* implies the higher-level *Relations*, just like in RDF Schema.

Every *Item* that has any kind of *Relation* to any other node is at least "cds:related". If one *Item* A is an alias of another *Item* B, then all *Statements* about A are considered to be *Statements* about *Item* B. If one *Item* is a replacement for another one, then an occurrence of A's content is replaced at edit-time with B's content, e.g. when editing text.

CDS relations are based on four axes, which we consider to be the core dimensions for information structuring:

(1) **cds:hasTarget** and its inverse **cds:hasSource** model generic, directed linking. This can be found in WWW hyperlinks, references in documents, or links in the file system. The semantics of a link are pretty generic: A link refers to a target *Item*.

(2) **cds:hasAfter** and its inverse **cds:hasBefore** model any kind of ordering relation. 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.

(3) **cds:hasDetail** and its inverse **cds:hasContext** represent any kind of hierarchy and nesting. Hierarchies are very common information structures present in documents, organisational charts, file systems, and user interfaces. For both relations (2) and (3) the transitive closure is calculated and used in search queries.

(4) **cds:hasAnnotation** and its inverse **cds:hasAnnotationMember** models annotations of *Items*. An annotation is typically a statement *about* an item—taking a meta perspective In CDS, Web 2.0-style tagging is considered a special form of annotation. Classifying an item with cds:hasType is a special case of tagging. Tagging has no formal semantics, but types are inherited via the cds:hasSubType-*Relation*—which is in turn a special form of cds:hasDetail.

The complete CDS *Relation* ontology is depicted in Fig .5.

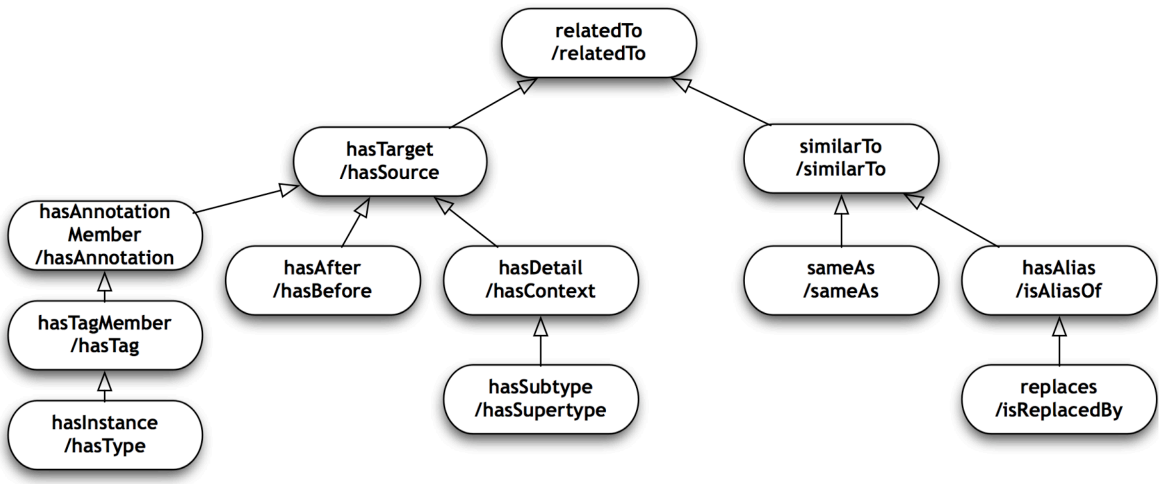

Figure 5: The CDS ontology

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.

As an example for CDS inferencing, we extend the example given in Fig. 3 with two more Statements that map Claudias Relation vocabulary to the CDS ontology:

```
    [works at]        cds:hasSuperType    cds:hasContext.
    [is located in]   cds:hasSuperType    cds:hasContext.
```

Now Claudia can pose the query "[Karlsruhe] cds:hasDetail ?" (or browse the *NameItem* [Karlsruhe]). The CDS model returns "Dirk" and "SAP" since both are (transitive) details of "Karlsruhe". A query for "[Karlsruhe] cds:hasDetail ?x AND ?x cds:hasType [Person]" would return only [Dirk]—if Claudia would also state that [Dirk] cds:hasType [Person].

5.6 Structured Text

We are currently defining a mime-type called *STIF* ("Structured Text Interchange Format")[7] which is based on the former *wiki interchange format* (WIF) [6].

STIF is meant as a simple markup language (in fact a subset of XHTML) allowing to represent *only structural*, but not visual features: headlines, list, tables, images and links are allowed, but font style, font size and color are not.

[7] http://wiki.ontoworld.org/wiki/STIF

A transformation from a STIF-text into a set of smaller, CDS-related Items is planned. The inverse direction will be used to assist the user in composing documents from a number of Items. E.g. a hierarchy of items can be mapped to document hierarchy (headlines, nested lists). Both processes ease the creation of structured content, as one can start to write structured documents, e.g. while setting in a meeting and later transforming the text into a more fine granular CDS model.

6. Conclusions and Summary

6.1 Achievements and next steps:

A CDS API and two authoring tools (a graphical desktop tool and a simple web editor) are currently implemented via swecr[8], using RDF. Existing RDF data can be browsed the CDS way, provided the RDF properties are mapped to CDS core relations. Next steps will include improvements in the mapping of terms to *Items,* integration into existing semantic desktop infrastructure, and implementation and evaluation of end-user tools.

6.2 Summary:

We have observed which structures people use in different tools and use cases and found that a small set of relations is very common across many kinds of knowledge artefacts. We believe, that the set of Conceptual Data Structures (CDS) presented in this paper can

(1) act as a formalism for recording, managing, and sharing personal knowledge,
(2) bridge the wide gap between informal or unstructured information and fully formal semantic models in Personal Knowledge Management (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.

References

[1] Nonaka, I. & Takeuchi, H. The Knowledge-Creating Company: How Japanese Companies Create the Dynamics of Innovation. Oxford University Press, 1995
[2] Max Völkel, Heiko Haller. Conceptual Data Structures (CDS)—Towards an Ontology for Semi-Formal Articulation of Personal Knowledge. In Proc. of the ICCS 2006. Aalborg University – Denmark, July 2006.
[3] David Karger; Karun Bakshi; David Huynh; Dennis Quan; Vineet Sinha: Haystack – A General Purpose Information Management Tool for End Users of Semistructured Data. CIDR Conference Paper, 2005. s.a. http://haystack.lcs.mit.edu/
[4] Resource Description Framework (RDF): http://www.w3.org/RDF/
[5] Carroll, J.J.; Bizer, C.; Hayes, P. & Stickler, P. Named Graphs, Provenance and Trust. HP, 2004.
[6] Max Völkel, Eyal Oren. Towards a Wiki Interchange Format (WIF). Proceedings of the First Workshop on Semantic Wikis—From Wiki To Semantics. 2006.

[8] http://swecr.semweb4j.org