Concept-based Management of Technical Knowledge

Klaus Wimmer and Klaus Meusel

Siemens AG
Otto-Hahn-Ring 6
D-81739 Munich
Klaus.wimmer@mchp.siemens.de

Abstract

Knowledge about technical systems deserves our attention for several reasons. It shapes industry's core processes which depend on knowledge. It is an asset of great value because it is used by those who engineer, operate or maintain a system.
This paper explains key characteristics of technical knowledge, the reason why it is difficult to manage and what we can do about it. Our experiences e.g. with documentation of complex communication systems and extensive use of SGML let us believe that management of technical information should rely more on conceptual structure and semantic markup. This will affect the way we design DTDs and use ontologies.

Characteristics of Technical Information

A prominent feature of technical information is it's complexity and we are greatly challenged to reduce it. An example will illustrate this case: the documentation used by technicians to operate and maintain a communication network. We generated such documentation as a technical information base of 20000 HTML-files and 100000 links among them. The major part of it describes service tasks and procedures; the rest describes functions, components, interfaces, dependencies, versions and variants of network elements. This information is carefully managed throughout its lifecycle. The authoring process, for instance, is supported by tools which prescribe how to structure information. To enforce structure we use five SGML-DTDs each of which consists of about 300 element- and attribute-types. All descriptions are stored as SGML-instances in databases; generators select and assemble the information pertinent to a specific version of a product and produce various publication forms. One such publication form is HTML as mentioned before.
So, it seems obvious that the complexity of technical knowledge stems from it's volume and structure. There is however still another dimension of complexity which is of a conceptual nature. To explain the point we will sketch a piece of information flow. Before product information (e.g. the description how to operate a network) arrives at the customer's site, it flows through several departments of the manufacturer of network gear. Typically, the information flow from engineering to service departments „transports" numerous aspects of a product: how can the product be installed, configured or diagnosed; how can the product interface to those foreign systems which constitute the customer environment; other views concern the states or dynamics of hardware or software. Last not least some views concern a product's legal and economic properties.
The views and concepts which structure technical knowledge constitute the „conceptual interface" when information flows from one department into another. Such views and concepts must be known by the receiver of information to understand the sender. In our domain we deal with an estimated 10.000 significant concepts and many more technical terms. Communications is like most other fields of technology. It is rich in terms and concepts which, however, have never been systematically collected.
But terms and concepts could be put to use immediately, for instance to build a portal to access the information on our family of products. Vertical portals may become part of the web technology catering to the broadening clientele of technical information.

Conceptual Deficit

That concepts can play a significant role is not a recent insight. Imagine *F* to be an important feature of a product, e.g. one of its interfaces, functions or components. *F* is described many times e.g. when it is designed, specified, implemented, tested, released and operated. It would mean a major step forward, if all these descriptions of *F*—they all belong to the concept of *F*—were interrelated. This would allow us to trace the evolution of feature *F*. Before we release *F*, we could look up earlier descriptions and check how it was implemented and tested and how the customer required it to be in the first place. Before we modify *F*, we could find all its versions and variants and thus the constraints that need to be accounted for. Before we ship *F*, we could find and select the information technicians must kow who will service *F*. Unfortunately such relations are hard to find automatically today, primarily because authors have much freedom in naming and describing *F*. In practice this means we use almost no

„conceptual markup" like <feature> *some description* </feature> when we talk about any kind of feature, or like <least-cost-routing> *some description* </least-cost-routing> when we talk about a specific feature. We use semantic markup only in a few important cases e.g. when we describe tasks or procedures. In these cases we pay a price because the semantic element type <procedure> is in turn made up from some 80 element-types. Many of them define the form of text or graphics. Only a few element-types mark the essential aspects of a procedure every technician must know like the steps, preconditions and effects of a procedure.

Clearly, cost is a limiting factor to semantic markup in our case as in all other cases we know of. No wonder, today's knowledge management techniques care little about concepts or even technical terminology, and the tools we use to handle information handle its form rather than its content.

Instruments for Handling Concepts

We wish we were better equipped to generate semantic markup and thus structure our voluminous information bases conceptually. But suitable taxonomies, glossaries, thesauri, ontologies and concept-based DTDs are not yet available for our specific technical domain. What a clever combination of such instruments could do for us is sketched in the following scenario.

Assume an author of service documentation uses a novel kind of DTD which does not force him/her to write sequentially i.e. a sequence of chapters, sections and paragraphs. Instead, the DTD would encourage the author to focus on content and create „topics". A topic in the service domain would describe a task, a procedure, an action or a component of a communication system. Each topic then leads to other related topics or views. A component, for instance, is described by it's functions, interfaces, structures or other related views. Initially, topics are created as independent modules of information. Later they are interrelated e.g. a step (topic) becomes *part-of* a procedure (topic) which becomes *part-of* a task (topic). The task is likewise related to a network component (topic) because the task *configures* the component. The component-topic may have already been related to those topics capturing its views.

In case the author feels left alone when he must describe an interface, he can consult his domain ontology to retrieve the concept „interface". The ontology collects and classifies the important domain concepts. The author can use them as „best-practice proposals" to capture the views of an interface such as input, output, thruput, protocols or standards. The ontology would „know" not only the generic concept named interface but also several specific kinds of interface concepts. The ontology also knows the technical terms in the glossary which can be used to name concepts. Last not least the ontology knows the kinds of relations which may apply between topics. For instance a task may diagnose, configure or repair a component. And of course, relations like „part-of" or „configure" are concepts as well.

By proceeding in this manner the author uses an ontology as an extension of a DTD. For the author each concept in the ontology is semantic markup „dragged" from the ontology to be dropped as a template into his document at the right place.

Note that while filling in the template, an author will use text and tables and graphics just like before. While some ontologies today are highly formal constructs, ontologies do not force us to become more formal or rigorous than what is implied by using SGML or XML. In our domain it would be costly and impractical e.g. to describe the mass of our procedures in such a way that allows for automatic reasoning or similar forms of computation. In other words, by semantic markup we mean conceptual markup only. Of course, reasoners can help us find out—as parsers do today—whether descriptions are conceptually valid or complete.

If a product and it's maintenance has been described by topics and their relations, we want to store this description in an information- or document- or object-management system. Later on we will add the variants of this description which correspond to new variants of the product. We use conceptual markup as schema information. As a result every piece of description has type-information which we can augment with meta- and style-information. In this manner we can use concepts as a means to enforce a certain degree of completeness and consistency of the „description modules" in our database. Remember that such an undertaking is highly problematic for traditional documents i.e. an often random sequence of sections, paragraphs, links and lists, where any such item is allowed to "talk about" anything.

The scenario already indicates that concepts play a role in all tools of our tool chain, which includes tools for authoring as well as document management and retrieval. A glossary combined with an ontology could become an integral part of any portal for accessing a (large) technical information base. The ontology explains the kinds of contents to be expected in an information base. And any concept may serve as a template for formulating precise queries. Last not least, our tool chain contains sophisticated tools for exchanging complex technical information between business partners. For instance, we have written several converters to transform an SGML-instance conforming to our DTD into a SGML-instance conforming to a partner-DTD. If both DTDs contain elaborate but differing element-types, like <procedure>, mapping one element-type onto the other presents an effort and a risk. We would be better off if an ontology contained a „master-concept" for <procedure> - an abstraction of the specific element-types we have to map. In general, if an ontology constitutes a collection of both popular and proven concepts in a domain, it can facilitate many forms of information processing such as transformation.

Ways to Proceed

These examples explain what concepts can do for us provided they come with the right tools. Tools are indispensable to create ontologies, DTDs and glossaries, tie them together and keep them up-to-date. These core instruments will be ready for use only if tailored to a specific domain and integrated into a specific tool chain for creating and managing information.

In the past decade we have dealt with several issues we deemed instrumental for concept-based management of technical knowledge. A selection of them is listed below:

- we have investigated how to construct domain concepts (Wimmer and Wimmer 1992)
- we have built several DTDs (Meusel 1994)
- we have built a top-level domain ontology using the terminological representation features of FLEX, a 3rd generation system in the KL-ONE tradition (Quantz 1995)
- we have built domain concepts into several DTDs and used them in practice (Gandenberger and Hecht 1996)
- we have built several DTDs to capture those complementary points of view of a technical product which occur during the early phases of the product engineering process (Gandenberger 1997)
- we have investigated how to design an ontology for a technical domain and how to find a conceptual abstraction for DTDs (Wimmer 1998)
- we have developed methods and tools to train authors how to use an ontology while describing a technical product (Peter, forthcoming).

Our efforts have not yet resulted in an industry-strength solution to build an integrated concept-based tool-chain. Numerous questions such as the following still need to be answered – and that is part of the position we take in this paper:

- How can we create ontologies for a domain which is as broad, diverse and dynamic as the domain of communications? Which tools can help us distill concepts from textbooks, standards or product descriptions and what methods should these tools implement?
- Concept based DTDs and technical ontologies must serve many purposes and prove their worth in such diverse contexts as authoring, generation, transformation or retrieval of documents. Many such use-cases are not yet well understood. How can we better understand and define the design characteristics of our conceptual instruments?
- There has been, fortunately, progress such as in linking DTDs to ontologies (Erdmann and Studer 1999) which is a major step forward in DTD-design. Still we must learn how to construct a well tuned set of concept-based DTDs to describe a product. Each DTD should help to capture only the information relevant during a certain phase of the product's life-cycle. In other words how can we make DTDs capture diverse views of a product such that, if taken together, these views yield the „complete picture"?
- A DTD and an ontology can each be seen as a set of rules. Ontology-rules concern a technical domain and are thus general in nature while DTD-rules are more specific reflecting regulations for processes, products, tools and applications. Since rules tend to change the question is: how can we keep the evolution of ontologies and DTDs under tight control?
- Last not least we want our ontologies and DTDs to be easy to understand and to use. At present this is not the case. In particular, the web of generic concepts—the upper layer of our ontology—deserves our attention because it is used by a large clientele with diverse backgrounds. Thus, how can we provide training of conceptual skills to those who create or use editors, archives, generators, transformators, portals, or other concept-based tools?

Acknowledgment. We gratefully acknowledge our colleagues and Paul Madsen (Newbridge Inc.) for sharing their thoughts about DTD design with us.

References

Erdmann, M. and Studer, A. 1999. Ontologies as Conceptual Models for XML Documents. *12th Wokshop on Knowledge Acquisition Modeling and Management.* Banff Canada. (http://sern.ucalgary.ca/KSI/KAW99)

Gandenberger, S. and Hecht, A. 1996. Using SGML for Model-based Engineering Data. In *Proceedings of SGML'96.* Boston Ma., CGA.

Gandenberger, S. 1997. Using SGML in an Object Oriented Development Process. In *Proceedings of SGML/XML'97.* Washington DC, CGA.

Meusel, K. 1994. An SGML-based Architecture for O&M Manuals in the Telecommunications Industry. In *Proceedings of SGML'94.* Vienna Va., CGA.

Peter, K.-G. Entwicklung und Erprobung eines fallbasierten Lernprogramms zum Erlernen einer Modellierungsmethode für komplexe technische Systeme. *Dissertation.* Forthcoming.

Quantz, J. et.al. 1995. The FLEX System. *KIT-Report 124.* Berlin: Technical University.

Wimmer, K., and Wimmer, N. 1992. Conceptual modelling based on ontological principles. *Knowledge Acquisition* 4:387-406.

Wimmer, K. 1998. Design and Use of an Ontology for Complex Artifacts. In *AAAI Technical Report SS-97-06.* Menlo Park, Ca.: AAAI Press.