Modelling Versioned Hypertext Documents*

Mária Bieliková and Pavol Návrat

Slovak University of Technology, Dept. of Computer Science and Engineering,
Ilkovičova 3, 812 19 Bratislava, Slovakia

Abstract. Versioning of hypertext documents is in many aspects very similar to versioning of software systems (and their components). In the paper we concentrate on an analysis of similarities and differences between them with the intention of possibly finding in the area of software configuration management a starting point for a new method of version control in hypertext systems. Then, we have proposed a model of a hypertext document which takes into account the perspective of its permanent change. Hypertext documents are modelled by two kinds of nodes in an AND/OR graph. The model forms a basis for building a configuration.

Keywords: hypertext document, hypertext document model, version, configuration.

1 Introduction

Version control has been identified as one of the critical research areas in the hypertext field [8]. It is important especially for hypertext documents published on the web.

Methods of support to versioning hypertext systems have been subject of intensive research for some time [9, 12, 14, 7]. However, the problem has quite naturally been related by many to versioning software systems which is being studied in software engineering.

Software configuration management (SCM) is a very active research area of software engineering today. In spite of many contributions to finding a way how to manage (large) software systems which evolve, researching a framework for unified version model which integrates extensional and intensional versioning, state-based and change-based versioning, revisions and variants, etc. remains on the agenda [4, 15].

Although we can consider a hypertext document as a software system (and consequently apply principles of SCM to versioning and configuration management of hypertext documents) there are several specific features of hypertext documents which deserve attention when efficient CM is to be implemented. We will discuss them later in the paper.

It becomes increasingly important to make explicit the structure and the relations between parts of the document because there is often a need to build a *configuration* of the (part of) document as a whole.

Documents are typically highly interrelated and often have an implicit structure (e.g., order of chapters, content of document, index, etc.). Therefore the process of building a hypertext document configuration is itself a complex one. Bookkeeping of attributes and relations of thousands of objects alone, not to speak of the frequency of their changes is a task which can best be handled by a computer. A support from a computer should further be sought in freeing the author(s) of document from the burden of a too detailed configuration specification. Instead, the author should have means to write higher level requirements which specify the configuration implicitly. Ultimately, this leads to employing relevant knowledge which would be represented explicitly and used by the computer. This can be considered as an approach to automating the above mentioned part of the hypertext documents management.

Any progress in automating is hard to imagine without further formalisation in describing the objects and processes. In this paper, we discuss the problem of modelling a versioned hypertext document. A model is used to express its structure, respecting in our case the viewpoint of building the hypertext document configuration. We have adopted the *AND/OR* graph model used in software configuration management [1]. Semantics of the model is specified according to specific properties of hypertext documents.

* The work reported here was partially supported by Slovak Science Grant Agency, grant No. G1/4289/97.

2 Hypertext documents vs. software systems

A hypertext document is in contrast to a traditional text (as a book) nonsequential (nonlinear), i.e. there is no single order that determines the sequence in which the text is to be read.

When reflecting on hypertext documents, two perspectives are especially worth mentioning: (1) *user perspective* where a navigational character of the hypertext document is important (for a document to be a hypertext, it must allow the users to take control over a set of links among units of information interactively [11]); (2) *developer perspective* where a life cycle, an architecture, a model, etc. are important features.

Our main interest is to capture the developer perspective. From this point of view a hypertext document has many similarities with the notion of software as it is traditionally understood:

- both consist of many components (nodes in the hypertext) which may undergo changes;
- both usually consist of many content types of components (e.g., in a WWW site the contents can range from HTML pages to Java programs, or sound files);
- components of both of them can be either static i.e., source (known in advance) or dynamic i.e., computed, or derived i.e., generated by the system;
- components of both of them are interrelated in several various ways (here, e.g. composition and dependency relationships in a software system specialize to links in a corresponding hypertext document). Relationships can be in both of them represented either explicitly in the sense that their instances have been marked in the document, or implicitly when the instances can be inferred from the contents of the document;
- both of them are under computer control;
- both of them are very often created and maintained by teams, so it may be desirable to maintain different versions;
- the components are in both of them managed using file systems, relational databases, and object-oriented databases;
- both are often developed in teams by multiple developers (e.g., when they are large).

Usually, the material in a hypertext document (e.g., WWW site) is authored by several teams. Since many documents can be integration points for various departments or functions, each of these departments use their own authoring teams to prepare the information for the hypertext documents. However, overall document has to present a consistent, navigable hypertext that is made up of material supplied by these teams. The problem is similar to the integration and testing step performed in software development. Teams may want to "install" or "stage" delivery of information in hypertext by swapping different configurations for different uses at different times.

Therefore collaborative development leads naturally to versioning. However, we do not discuss this issue in the paper. In [5] there is presented an interesting approach to modelling versions in a collaborative work by so called 'modal model' which takes into account a context of the use of versions.

The primary purpose of a software system development is to build executable software from its components (i.e., a configuration) which can be used for automated support of some task. The primary purpose of a hypertext document is to convey information by being browsed and consequently read (or heard, watched, etc, in case of hypermedia). In case of internet, or intranet hypertext documents there is a *new* interesting goal of browse (or download) specific configuration of the visited site.

Specific properties of hypertext documents enable the use of more specific structures and processes for configuration management. They are based on the characteristic properties and occurence of components and relationships between them:

- a structure of hypertext documents is more dynamic and subject to change than with most software systems. This means that hypertext document has great flexibility, which is normally an advantage but can also be a disadvantage [11]. Structure of software systems is typically restricted to a strict hierarchy which can be modelled by a tree or by an acyclic directed graph. A model of hypertext document may contain cycles;
- a hypertext document often requires that there are represented dependencies which are finer grained than in a traditional software;
- a hypertext document contains fewer types of relationships between components (nodes). Relationships are represented by links which often express explicit navigation ("activate this link to visit that related resource") or the position of a document within a series of documents.

3 Model of hypertext document

Solving various problems related to building hypertext document configurations requires describing the actual hypertext document in the simplest possible way, but still sufficiently rich to reflect the principal relations and properties which are decisive in the building process.

In spite of mentioned specific properties of a hypertext document we can with advantage use the analogy of the hypertext document with a software system. Generally, various kinds of graphs are being used to model software systems. The model is often provided by AND/OR graphs [13, 6, 1].

We attempt to describe a hypertext document with the specific purpose in mind, i.e. to be used during development and maintenance, and specifically in building the hypertext document configuration. Therefore, our model encompasses those parts of the document and those relations among them which are important for building a configuration. Due to many similar properties of software systems and hypertext documents we find the intertwinded AND/OR graphs suitable for modelling a hypertext document.

Note that we adopted the version oriented model (as an alternative to a change oriented one) where *explicit versions* of components are used to construct configurations. "Intensional vs. extensional versioning" is orthogonal to a model of the system [4], thus this is not a restriction for the proposed model.

3.1 Elements of the model

Throughout the rest of the paper we use the term *hypertext component* as any kind of identifiable entity put under configuration management control (i.e., hypertext nodes – elementary units as well as parts of a hypertext document – composite units). Creating a hypertext component version can be done in one of two possible ways. First, versions are created to represent alternative solutions of the same purpose. They differ in some attributes. Such 'parallel' versions, or variants, are frequently results of different specializations. Second, versions are created to represent improvements of previous ones, or as modifications caused by error correction, content enhancement, and/or adaptation to changes in an environment. Such 'serial' versions, or revisions, are frequently results of concretizations of the same variant. A *family* of hypertext components comprises all components which are versions of one another.

When defining a model of a hypertext document, relations between hypertext components should be considered. They can be either development-induced, for example *is_variant* and *has_revision*, or navigational, i.e. hypertext *links*. We identify several types of links based on the location of the source and destination: *intra-component* links are links with both the source and destination located in the same component; *inter-family* links are links with both the source and destination located in different components within different families; *inter-document* links are links with both the source and destination located in different documents. We distinguish also *implicit* links which mirror the predefined structure of the document such as *next, previous, home*, etc.

We found useful to consider *variant* as a set of hypertext components. This conceptual design choice does not impose any serious limitations in most cases. On the contrary, it provides a considerable flexibility to the configuration management process. It offers a useful abstraction that should simplify the process. In order to describe variants, we define a binary relation *is_variant* which determines a set of hypertext components with the same (1) navigational relations, (2) variant attributes and (3) constraints (in the sense of combining components to configurations) within a given family.

Let us note that the distribution of hypertext components to variants depends on a decision which properties are considered as variant properties and as revision properties, i.e. unique properties of the actual hypertext component. Decision about distributing attributes is left open in our approach because it depends on the project, its size, problem domain, etc. Typical recommendations applicable in many cases are to consider as variant attributes the following properties: specific characteristics of the document being presented, characteristics of the development environment (language, formalism for text formating), etc. This means in our terminology that their change leads to a new variant. Properties related to the development process such as state, change description, author, date, time are often considered as revision attributes, i.e. their change leads to a new revision.

One consequence of our design decision of taking variants to be sets of components is that from the two kinds of versions of components, only revisions are left to represent actual single hypertext components (e.g., nodes of a hypertext network).

As an example, let us present a part of a hypertext document which includes versions of (some of) its components. The example is taken from Maria's home page where – among other things – the

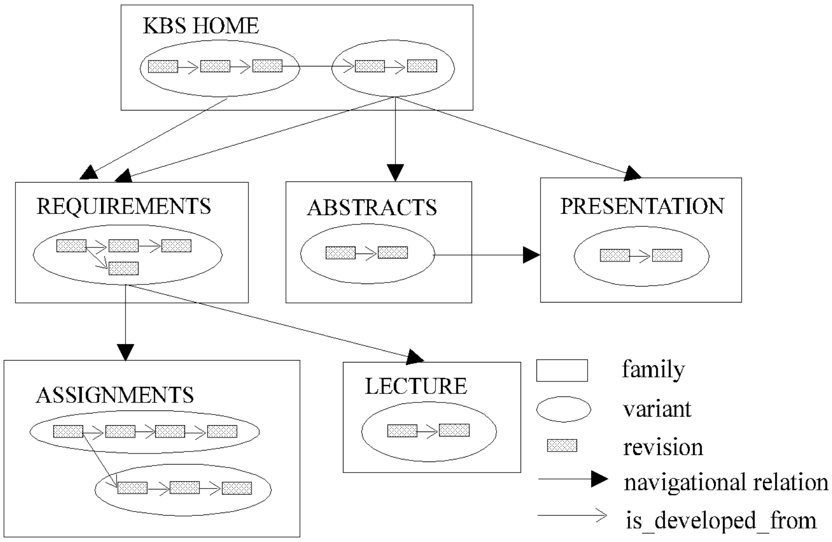

Fig. 1. An example hypertext document: partial hierarchy of elements.

subject Knowledge Based Systems is presented. Hypertext components are shown in Figure 1 along with navigational relations between them (implicit links and intra-component links are not illustrated).

Let us reflect the concept of variants once more by viewing of this figure. Assuming a branch in the version tree of REQUIREMENTS family resulted just from changing the author and the attribute "author" is considered a revision attribute, this family consists of just a single variant. Although there is a branch in the version tree, it does not give rise to another variant.
Consider now the ASSIGNMENTS family. Assuming a branch resulted from changing a graphic mode and the attribute "graphic" is considered a variant attribute, the family consists of two variants.
Finally, a change of the navigational relation in the family KBS HOME gives rise to another variant even if there is no branch.

3.2 AND/OR graph model

The concepts introduced above will let us to formulate a model of a hypertext document which supports the process of configuration building. In the case of a software system a configuration is often defined as a collection of components tailored to a specific purpose. This definition can be adopted to a hypertext document, too.

Our method of modelling a hypertext document H is to describe it by an oriented graph $M_H = (N, E)$, with nodes representing reference to families and variants in such a way that these two kinds of nodes alternate on every path and every maximal connected subgraph has at least one root.

Any element of $E, (e_1, e_2) \in E$, called an edge, is of one from among the two mutually exclusive kinds. Either $e_1 \in VARIANT_H$ (a set of variants of a hypertext document H) and $e_2 \in F_S$ (a set of family names of a hypertext document H); in this case, the node e_1 (variant) is called the AND-node. Or $e_1 \in F_S, e_2 \in VARIANT_H$; in this case, the node e_1 (reference to family) is called the OR-node. Revisions are covered in the model through AND-nodes which represent variants, i.e. sets of revisions.

We remark that the binary relation originating at AND node stands for navigational relations (relating variants to families) Implicit links and intra-component links are not captured by the model. The relation originating at OR node mirrors *has_variant* relation. In case when composite nodes are incorporated in the model the former relation can represent also the composition relationship.

The requirement that a model of a hypertext document should have at least one root is motivated by the fact that the model should serve the purpose of building a hypertext document configuration. When there is no root in a model, then it is not possible to determine which components are to be selected for a configuration.

Actually, this requirement is not a restriction in our case as a document commonly has an entry point to start reading from. Moreover, the hypertext document model captures the notion of a definite hypertext document. In the case of WWW hypertext documents we do not have ambition to model the whole hypertext network. Rather, a WWW network is to be modelled as a collection of hypertext

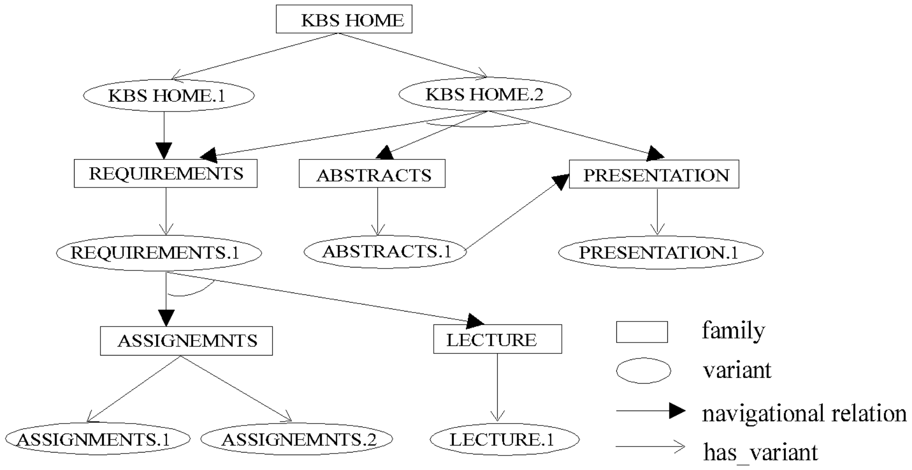

Fig. 2. Model of the hypertext document from Figure 1 represented as an AND/OR graph.

documents all of which are treated independently from the others. This may seem as a restriction and thus a disadvantage but as a consequence it highly simplifies the model. Inter-document (external) links are not represented in the model, so their change does not give rise to a new variant. Moreover, such a model is typically formed by an acyclic AND/OR graph.

The example hypertext document depicted in Figure 1 can be expressed by an AND/OR graph in Figure 2. For the sake of simplicity, variants are given names which are derived from the name of the corresponding family by suffixing it with a natural number. Formal definition of such a model is presented in [1] where modelling a software system is considered. The difference lies in the interpretation of specific parts of the model which were described above.

Note that in such a model of a hypertext document, versions of links are represented by a new variant in the family of components. A hypertext component need not to be a file. For example, in many cases it is advantageous to consider a page (in a WWW site) together with all graphical objects included in it as a hypertext document node.

3.3 Building of a hypertext document configuration

When building a configuration, for each family already included in a configuration there must be selected at least one variant. For each variant already included in a configuration, there must be included all the families related by architectural relations to that variant. Taking into account that a software component is determined completely only after a revision has been selected, the resulting configuration is built by selecting precisely one revision for each selected variant.

Note that not precisely one variant for each family included in a hypertext document configuration is to be selected but instead *at least one*. This is a consequence of the specific characteristics of the hypertext document and of our defintion of its model. Variants can for example represent sets of revisions in different languages (Slovak, English, etc.). Sometimes there is a requirement to have the document written in several languages so more than one variant for a particular family should be incorporated into the configuration.

There can be built several different configurations from a model of a hypertext document, usually based on different required purposes of the desired configuration. There can be desired a configuration for the end user, a configuration for further development, etc. Such configurations can be specified by different configuration requirements.

In order to build a configuration, our method that was designed originally for software systems can be used [2] together with a programming technique of implementing search of AND/OR graphs with constraints [3]. This technique uses markings to maintain consistency and identification the reason for a deadend. It attempts to find a place in the graph where the search for an alternative solution should be resumed.

The method takes into account the knowledge about the navigational relations between components, about selecting components (families) and also about selecting a variant and revision for each family. Selection of a variant and a revision can be accomplished by our method for version selection [10]. Our strategy of version selection is based on a sequence of heuristic functions which reduce the set of suitable versions. By changing the order in which the heuristic functions are applied we can vary the importance of the evaluation criterion which the given function embodies.

4 Conclusion

We have presented an *AND/OR* graph model of hypertext document. The model is based on the similarities of software systems and hypertext documents and on specific characteristics of hypertext documents. Main strengths of our approach to modelling hypertext documents for configuration management are (1) considerating of the conceptual distinction between variants and revisions, (2) considerating of navigational relations at the variant level, (3) abstarcting from implicit and intra-component links in the model which results in simpler model, and (4) allowing more than one hypertext component from a particular family to be in a configuration. In the case of web documents we propose an atomic hypertext document to be in most cases of greater granularity than a file.

Our way of modelling a hypertext document is limited by the fact that every change (leading to a new version) of variant attributes, constraints or navigational relations outside the component but within the hypertext document results in a new variant regardless to the real nature of the change.

The area of hypertext documents versioning and configuration building requires further research. Open problem is acquiring knowledge on the suitability of component versions. In case when attributes of components are not known for no matter what reason, methods of reverse engineering could be attempted to supply them.

The proposed model together with a method for configuration building could be incorporated into a hypertext system. At the moment we have started with prototyping for modelling versioned web pages. We concentrate on a developer perspective as was indicated in the paper. At the implementation level it is advantageous to use results from research in versioning databases.

References

1. M. Bieliková and P. Návrat. Modelling software systems in configuration management. *Applied Mathematics and Computer Science*, 5(4):751–764, 1995.
2. M. Bieliková and P. Návrat. A knowledge based method for building a software system configuration. *Knowledge Based Systems*, 9(1):61–65, 1996.
3. M. Bieliková and P. Návrat. A Prolog technique of implementing search of A/O graphs with constraints. *Computers and Artificial Intelligence*, 16(4):377–400, 1997.
4. R. Conradi and B. Westfechtel. Version models for software configuration management. *ACM Computing Surveys (to appear)*.
5. A. Dix, T. Rodden, and I. Sommerivlle. Modelling versions in collaborative work. *IEE Proc. Softw. Eng.*, 144(4):195–205, August 1997.
6. J. Estublier. Configuration management: the notion and the tools. In *Proc. Int. Workshop on Software Version and Configuration Control*, pages 38–61, Stuttgart, 1988.
7. A. Haake and D. Hiks. VerSE: Towards hypertext versioning styles. In *7th ACM Conference on Hypertext*, Online Proceedings (http://www.cs.unc.edu/~barman/HT96/), Washington DC, USA, March, 1996.
8. F.G. Halasz. Reflections on NoteCards: seven issues for the next generation of hypermedia systems. *Commun. ACM*, 31(7):836–852, July 1988.
9. C.Y. Lo. Comparison of two approaches of managing links in multiple versions of documents. In *Proc. of the Workshop on Versioning in Hypertext Systems*, Edinburgh, September, 1994.
10. P. Návrat and M. Bieliková. Knowledge controlled version selection in software configuration management. *Software - Concepts and Tools*, 17:40–48, 1996.
11. J. Nielsen. *Hypertext & hypermedia.* Academic Press, Boston, 1990.
12. L.F.G. Soares and M.A. Casanova. Nested composite nodes and version control in hypermedia systems. In *Proc. of the Workshop on Versioning in Hypertext Systems*, Edinburgh, September, 1994.
13. W.F. Tichy. A data model for programming support environments and its application. In B. Langefors, A.A. Verrijn-Stuart, and G. Bracchi, editors, *Trends in Information Systems*, pages 219–236. North Holland, 1986.
14. F. Vitali and D.G. Durand. Using versioning to support collaboration on the WWW. In *Fourth World Wide Web conference*, 1995.
15. A. Zeller and G. Snelting. Unified versioning through feature logic. *ACM Transactions on Software Engineering and Methodology*, 6(4):397–440, 1997.