A Version Model for Supporting Adaptation of Web Pages

Rodrigo Giacomini Moro*
Centro de Processamento de Dados - UFSM
Santa Maria, RS, Brasil
rgmoro@cpd.ufsm.br

Renata de Matos Galante†
Instituto de Informática - UFRGS
Porto Alegre, RS, Brazil
galante@inf.ufrgs.br

Carlos Alberto Heuser
Instituto de Informática - UFRGS
Porto Alegre, RS, Brazil
heuser@inf.ufrgs.br

ABSTRACT

Maintenance of large Web sites is a complex task, similar in some sense to software maintenance. Content should be separated from the formatting rules, allowing independent development and maintenance of both parts. As in software maintenance, version management is important in order to separate stable versions from versions under development. Further a version model that allows alternative versions may be used to support adaptation and personalization of Web content and formatting. To fulfill this request, this paper presents a Web server infrastructure to accomplish version control and to support adaptation of Web pages in a way that is transparent to the user. A version model that separates page content and formatting in distinct components is provided. A configuration is also proposed and it is responsible for generating dynamic pages.

Categories and Subject Descriptors

H.5.4 [**Document and Text Editing**]: Version Control; I.7.1 [**Hypertext/Hypermedia**]: Architectures

General Terms

Management, Design

Keywords

Versions, Configurations, Hyperdocument Model, Web Adaptation

1. INTRODUCTION

In order to attract visitors, the content of Web sites must be maintained up-to-date and must be friendly presented. This results are an increase on the number of pages and in the complexity of a Web site. Maintenance of large Web sites becomes harder, and software tools that support site development and maintenance are needed. A tool that supports Web site maintenance is called a *Web Content Management System*.

Separation between page components (for instance, content, formatting, images) is an important feature of a Web Content Management System because it simplifies page maintenance to both humans and software tools. This separation may be (and usually is [18]) provided by storing page content in XML documents and page formatting in some stylesheet language like XSLT. Translation from XML to HTML format using XSLT is a common way of presenting XML document in a standard Web browser.

In many aspects, a Web site is similar to software [2]: both have components that can be of many types and they are maintained by teams of many developers. As in software management, in Web site management the versioning of components is a key aspect. Software configuration systems [10] usually store all states of program components (e.g. modules, header files, etc.). The reasons for managing multiple versions are manifold [28, 3, 17]: reuse of versions, maintenance of versions already delivered to customers, storage of back-up versions, or support of change management. Storage of component history is also a key feature in Web Content Management, because it allows the Web site manager to roll-back to a preliminary state (version) if some problem is found in the current state of the document (or document component, including images and other multimedia resources).

*Work was done while the author was at Instituto de Informática - UFRGS.

†The first author is partially funded by National Council for Scientific and Technological Development - CNPq, with the special program CTPETRO. Grant number:502009/2003-9.

WIDM'04, November 12–13, 2004, Washington, DC, USA.

Another site requirement that is becoming common is page *adaptation* or *personalization* [4] depending on user profiles. Site adaptation or personalization allows the Web server to provide content and formatting customized for a specific class of users. A Web site with adaptation may have an important advantage over its competitors in attracting visitors. For example, pages with content that is specific to some period of time (like promotions or holidays) can be automatically published. Besides the period of time, adaptation may consider the user or the device that is requesting a page and use this information to select the best version of each page component. Adaptation may also consider current server work load and provide a less resource intensive page [1] if a server overload condition is detected.

In a standard Web server the construction of adaptive pages is usually achieved by running some type of program or script created specifically to this end. Although this is a very flexible approach, it is error prone and requires skilled developers.

When a page component has *alternative* versions, it is possible to combine versions of components to construct alternative versions of a page, thus providing support to the adaptation of pages. Most software configuration systems have a *configuration process*, that chooses a version for each software component and generates a complete program (configuration). A similar configuration process can also be implemented for Web pages, if the Web server provides a versioning system that implements the concept of alternative versions of pages together with a configuration process.

This paper presents a Web server infrastructure to accomplish version control and to support adaptation of Web pages in a way that is transparent to the user. A version model that separates page content from formatting rules is proposed. This model is based on WebDAV [9], the standard protocol used for Web versioning. A configuration process is also proposed and it is responsible for generating dynamic pages. Besides, the model provides resource properties and version selection criteria leading to an automatic configuration process.

The rest of this paper is structured as follows. Section 2 shows an overview of the proposed Web server infrastructure for supporting adaptation of Web pages. The proposed version model as well as the automatic configuration process are described in Section 3. An example of an application is given in Section 4, in order to illustrate the different concepts previously presented in Section 3. Section 5 provides a comparison with work done by other authors while Section 6 discusses future work and summarizes the main ideas of the paper.

2. THE WEB CONTENT SERVER ARCHITECTURE

This section presents an outline of a content server architecture that provides version control and adaptation pages. The users of the server are of two types: the *Web site manager* (or simply designer) and the *Web site visitor* (or simply user).

The general architecture of the server is depicted in Figure 1, in which the modules are presented in a bottom-up order. The *proposed infrastructure* is an extension to the WebDAV data model [9], the versioning standard for the Web. WebDAV (*Web-based Distributed Authoring and Versioning*) is

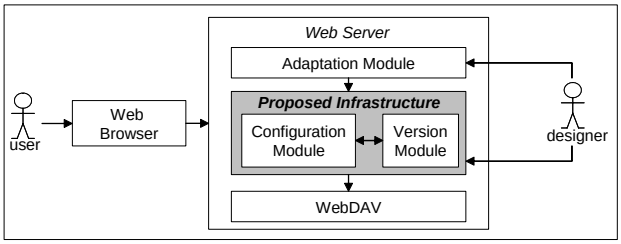

Figure 1: Content server architecture

a network protocol that allows a user to update a site remotely and to control versions of the files (resources) stored at the site. By adopting an accepted standard, we aim at simplifying the implementation of our model.

Actually, the proposed infrastructure can be split into two parts: *version* and *configuration* modules. The *Version Module* implements the WebDAV data model, coordinating storage modules used to store versions content and storing resource properties. The *Configuration Module* implements all concepts of our extension to the WebDAV model, maintaining consistency of associations between model concepts and targets of references.

The *Adaptation Module* handles information that are used to adapt pages and intervenes in the configuration process by providing additional version selection criteria to the configuration process. In this paper, the process of adaptation is not discussed. We just present the infra-structure that is needed to the adaptation module.

Finally, while the the *Web Browser* represents the applications used by user to see (adapted) pages, the *designer* can access the server and manage the resource repository.

In the following sections we detail our model and we show how the requirements explained here can be satisfied.

3. VERSION MODEL

This section presents the version model for supporting adaptation of Web pages. The main feature of the version model is to provide alternative versions of content and formatting components. Figure 2 shows (through an UML class diagram) the main concepts of the proposed version model.

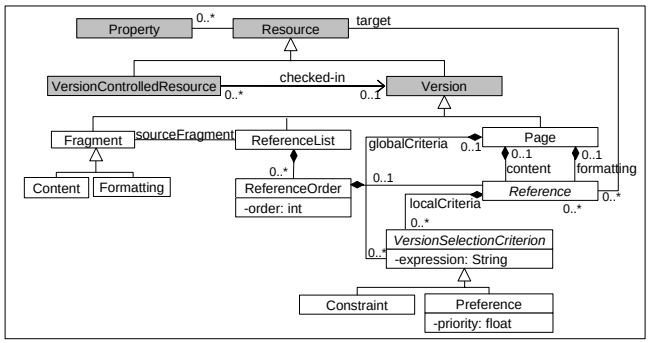

Figure 2: UML class diagram of the simplified model

The proposed model is an extension to the WebDAV data model. The WebDAV concepts, `Resource`, `Property`, `VersionControlledResource` and `Version` are depcited with a filled background in Figure 2. A `Resource` is a network data object or service that can be identified by an URI (*Uniform Resource Identifier*). Resources can have associ-

ated descriptive information (metadata), given in form of a name/value pair, represented by the `Property` class in Figure 2. A `VersionControlledResource` is a resource that has all its states controlled by the server, i.e. when the state of a version-controlled resource is modified, the previous state is not overwritten and the new state is also stored. Each state of a version-controlled resource is called `Version` resource. Versions are logically arranged as a graph, where each version is a node and arcs are derivation relations between versions (this association as others concepts of WebDAV are not shown in Figure 2 due to space restrictions). Each version-controlled resource has a version considered as its *current version*. Each current version is automatically maintained as the most recently created and it is indicated by a `checked-in` association in Figure 2. By specializing versions we can also specialize version-controlled resources. Hereafter, all concepts of the proposed model that are resources were specialized from the `Version` class.

The proposed model is based on references that relate the page components. The `Page` class is a resource that represents a Web page to the designers. The `Reference` class relates its containing resource (source) to its `target` resource (association in Figure 2). The page has two references, one to its `content`, which is an XML document that stores the page content, and other to its `formatting`, which is an XSLT stylesheet (or some other presentation format) that contains the formatting rules for the page. They appear as associations in the UML model. This way, the page content and formatting can be independently handled, simplifying both development and maintenance.

In addition, since content and formatting can be included in other XML documents and XSLT stylesheets, we define the fragment concept, represented by the `Fragment` class in Figure 2. Similar to the page modelling, content and formatting are also separately kept in the fragments, as shown in Figure 2. Besides, a fragment may have references (in its content) to other fragments of the same type (content or formatting) in order to include them. The `sourceFragment` association in Figure 2 represents a list containing all the references of a source fragment.

When a hyperdocument having versions is used as a component of another one, references to it can be made in one of two ways: reference to a specific version, named *static reference*, or reference to the version-controlled, named *dynamic reference*. In a composed hyperdocument, in which components can contain versions, a *configuration process* selects exactly one version of each of its components. Dynamic references must be solved and the system must choose a version according to some pre-defined criteria.

The default behavior of the configuration process is to select the most recent version for each referred version-controlled resource. However, this behavior is not always appropriate as the configuration process does not try to select the best version. Thus, there should be means to intervene in this behavior. In order to solve this problem, designers can associate *version selection criteria* (the `VersionSelectionCriterion` class, in Figure 2) that guides the process choice of suitable versions for page configuration, similar to a query in a database system. Using version selection criteria it is possible to choose the most suitable versions even if new versions are created later. Versions are chosen based on their properties as specified by the version selection criteria. Besides, depending on its scope, a criterion can be local or global (`localCriteria` and `globalCriteria` associations in Figure 2). The local criteria are used just in one reference solution. Otherwise, global criteria are used to solve all references of a page request.

A criterion may also specify a *constraint* or a *preference*. For instance, as a component can include many others, it would be boring to a designer to specify the same criterion to each reference. In this case, the designer can specify a global criteria that will be considered in the resolution of all references. A constraint specifies a feature that the selected version must fulfill. This means the criterion is mandatory because it represents a consistency rule that must be satisfied (the `Constraint` class in Figure 2). A preference specifies features that are desirable in the selected version. This means the criterion is optional if it specifies just a preference that should be satisfied but is not required. The `Preference` class has an attribute `priority` that indicates its importance; it is used by the configuration process to establish the order of the preferences evaluation.

We also model a resource that stores the criteria separated from the fragments. This resource is called *reference list* (`ReferenceList` in figure 2). It has a list in which the elements (`ReferenceOrder`) are all the references of a source fragment (association *sourceFragment*). The order of the references (*order* attribute of `ReferenceList`) in the reference list is the same the in the source fragment. For each reference, the reference list also stores its version selection criteria. Thus, an author may edit the reference list to specify the version selection criteria for the references of the fragment.

A fragment has an association (called *referenceList*) with a version-controlled resource that is its reference list. When a fragment is created or updated, the Web server must automatically extract the references from it; the server must use these references to create the reference list (version-controlled resource). The reference list and the fragment must be associated by both associations *referenceList* and *sourceFragment*.

The association *referenceList* is used by the configuration process to locate the current version (association *checked-in*) of the reference list. The process needs the reference list in order to get the criteria to resolve the references. The association *sourceFragment* allows an author to locate the fragment whose references were extracted from. It cannot be done through the association *checked-in* because this association is unidirectional, and can not be accessed by the current version of the reference list.

The Web server (or a module that is in charge of adaptation) can specify global criteria in order to lead the configuration process in generating adapted pages. In this case, the criteria are provided to the process as arguments.

In addition, the Web server (or adaptation module) may provide to the process a set of environment properties. The *environment properties* represent information about the user needs, for instance, the preferred language. Thus, the criteria may use the environment properties to express the features that suitable versions should have. This way, the designer may also specify criteria that generate adapted pages.

4. DYNAMIC CONFIGURATION PROCESS

This section explains the proposed version model for supporting adaptation of web pages through a series of examples and illustrates how it can be used independently for supporting adaptation of Web page content and formatting. We consider as example the creation of a Web site that contains news, in which the following aspects of the page are adapted: (*i*) *content language*, there are two alternatives (English and Portuguese); and (*ii*) *page format* - there are two formats, one for presentation on a Web browser and other for presentation on a mobile phone.

The examples show several versions of the page. Example 1 shows the page separated in its content and formatting components. Examples 2 and 3 demonstrate the configuration process while Examples 4, 5 and 6 demonstrate the specification of a version selection criteria by a page designer. Each example represents the result of a step in the construction of the Web page, for which it is presented a figure that depicts the resources and versions of the page and its components.

The following conventions for graphical representation are adopted. Version-controlled resources are shown as large ellipses with its name on top (depending on the type of the component, the ellipses are filled with a different style). Inside ellipses, there are circles representing versions. References between resources are represented by solid arrows pointing from the referrer (source) to the referred (target) resource.

Example 1 In the first example we show the separation of a Web page in components. Figure 3 illustrates the separation of the Web page in three version-controlled resources: (*i*) the page resource (`news.page`), (*ii*) the content resource (`news.xml`), and (*iii*) the formatting resources (`news.xslt`). The first version of the page resource has a reference to first version of the content resource, as well as a reference to the first version of the formatting resource. Versions properties are presented textually. For instance, the `news.xml(1){getcontenttype: text/xml}` string indicates that the first version of the `news.xml` resource has a *getcontenttype* property with "`text/xml`" value. The WebDAV-defined *getcontenttype* property indicates the format of the resource content.

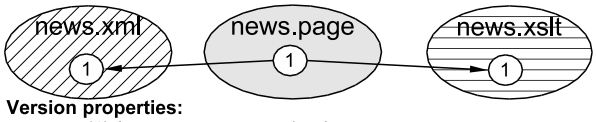

Figure 3: A page decomposed in content and formatting components

Example 2 The target of a reference may be a specific version, as in the example 1, or a version-controlled resource as we will show in this example. The goal of this example is to show the way references to version-controlled resources are resolved and how configured pages are generate due to a user request. Figure 4 illustrates a new version (2) of the page resource. Versions belonging to the same version-controlled resource are related by *derivation relationship*, being denoted by dotted arrows. The sucessor versions are indicated by the dashed arrows. There are two references from the second version of the news page to the version-controlled resources with page content and formatting.

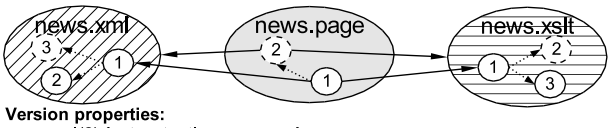

Figure 4: References to version-controlled resources

The content resource now contains two new alternative versions (2 and 3). Suppose that the second version is written in English and the third version is written in Portuguese. The language is indicated by the WebDAV-defined *getcontentlanguage* property. The formatting resource now also contains two new alternative versions (2 and 3). Suppose that the third version is written to translate `news.xml` versions to HTML and the second version is written to translate them to WML. The output format generated by the formatting versions is indicated by the author-defined *outputformat* property.

It is important to notice that when a version-controlled resource is required by an user (for instance, a Web browser), the behavior of a WebDAV server is to reply with the current content (*current versions* are been shown through a dashed border) of the version-controlled resource. However, when a page is required by the user, the configuration process is performed to solve the page references that point to version-controlled resources, in which the default behavior is to select the most recent version. In the Figure 4, for instance, if `news.page` is required, the configuration process will choose the second version of the page once that it is the current one. Then, references of the second version will be solved by selecting the third version (the most recent one) from both `news.xml` and `news.xslt` resources.

After the configuration process is performed, the server will apply the formatting rules specified by `news.xslt` (version 3) to the page content (`news.xml`, version 3), sending the result to the user. Thus, by using the default behavior, an HTML page written in Portuguese is generated.

Example 3 In this example we show how the configuration process may be guided and used to adapt pages. Since not always the default behavior of the configuration process is adequate, for instance, a user (or an adaptation module in the Web server) may request a page written in a language that deviates from that generated by the default behavior.Thus, there must be a way to modify this behavior, guiding the configuration process in selecting suitable versions.

Interventions in the behavior of the process are specified by *version selection criteria*. These criteria are expressions that relate version properties and guide the process in choosing suitable versions for page configuration. Figure 5 shows the version selection using the `getcontentlanguage='en'` criterion. This criterion specifies that versions in English should be used if available, i.e., it specifies a preference in

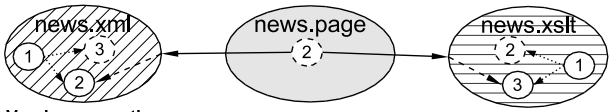

Figure 5: Version selection using the `getcontentlanguage='en'` criterion

version selection. This criterion must be provided to the configuration process by some process that is in charge of adapting Web pages. In this figure, dashed arrows are used to indicate the version that is selected by the configuration process.

Using this criterion, the process selects version 2 of `news.xml` because this version has the value `'en'` (English) for the *getcontentlanguage* property. However, since none of the versions of `news.xslt` has the value `'en'` for the *getcontentlanguage* property, version 3 (the most recent one) is selected. This means that there are no formatting rules that are specific to the English language. Thus, by using the provided criterion, a page written in English is sent to the user, instead of a page written in Portuguese, as it is the case when the default behavior is used.

In this example we also show that the *content* of a page may be adapted, in this case using the alternative versions of the content resource. In an analogous way, the *formatting* of a page may also be adapted. For example, a page can be seen on a Web browser or on a cellular telephone: depending on the client display device, a suitable version of the formatting rules must be selected.

Example 4 In this example we show that versions of content or formatting resources may contain references to others resources. We also explain how a designer may specify the criteria to select a consistent pair of versions, when a version refers to a version-controlled resource. The formatting versions shown in Figure 6 refer to a logotype image (GIF and WBMP) (some elements that do not change in the transition from one example to the next are omitted for the sake of clarity). It is necessary to intervene in the configuration process in order to assure that the consistent pair of the formatting rules and the logotype image are correctly chosen. The configuration process must choose: (*i*) the third version of formatting (that generates HTML) and the second version of logotype (in GIF format), or (*ii*) the second version of formatting (that generates WML) and the third version of logotype (in WBMP format).

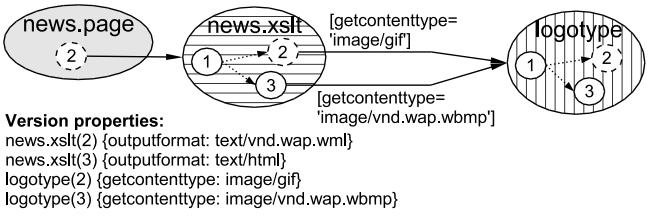

Figure 6: Criteria for solving a single references

In order to specify this constraint, local criteria may be associated to the formatting references. Figure 6 shows an example of local criteria (between square brackets) associated to references. Each criterion specifies that the version chosen as the reference target must have a content corresponding to the referred version of the formatting.

Formatting versions could, of course, refer to specific logotype versions to get consistent pairs. However, if a specific version is referred (as in figure 6) it is used, i.e., configuration process does not try to select the best one. It is not desirable because new improved versions of the logotype could be created in the future and this improvement would not be perceived by clients. Using version selection criteria it is possible to choose the most suitable versions even if they are created later.

Example 5 In this example we explain the use of a global criterion. An designer should specify such criterion in a general way, what requires that the source and target versions of a reference could be denoted. Thus, the language used to express the criteria must support the indication of such resources.

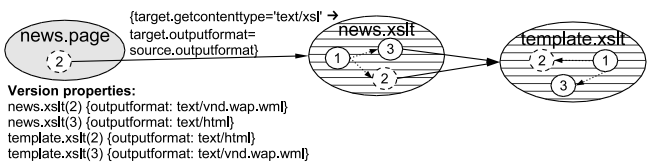

Figure 7: Specifying criteria to resolve all references.

Besides references to images, XSLT stylesheets may have references to other stylesheets, usually references for inclusion, using *xsl:include* or *xsl:import* tags [8]. This is useful when some transformation rules are used in several stylesheets, so they are grouped together in one or more stylesheets to be reused. In figure 7, there is a new formatting resource (`template.xslt`) that serves as a template for generating the page header and footer. It also has two alternative versions, one that generates HTML formatting and other that generates WML formatting.

Similar to the consistency problem in the references from `news.xslt` versions to `logotype` (example 4), the consistency between `news.xslt` versions and `template.xslt` can be guaranteed by specifying local criteria. As a stylesheet can include several others, it would be boring to an author to specify the same criterion to each reference. In this case, an author can specify global criteria.

Figure 7 also shows a designer-defined global criterion (between curly brackets) associated to the page. Notice that it is necessary to denote the source and target of the reference. The criterion specifies that: if a target version is a XSLT file, it must generate the same type of formatting that the source version does. This global criterion is a *constraint*; an example of a preference is given in figure 5. This figure also shows that there is no special notation to differentiate constraints and preferences.

Example 6 In this example we show the use of *environment properties* in the specification of version selection criteria. Figure 8 shows a new version-controlled resource (`bargraph`) that is an image of a bar graph with statistical information. It is referred by the versions of `news.xml` and has two alternative versions: one (2) in GIF format and other (3) in WBMP format.

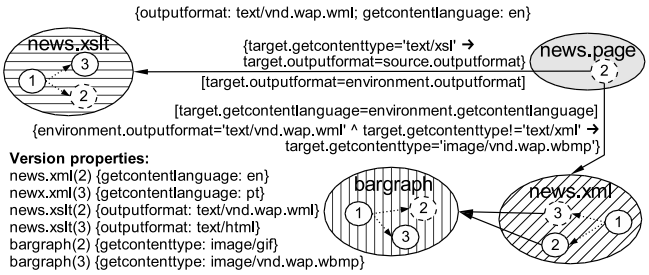

Figure 8: Using environment properties in the criteria specification.

This is an interesting example because the image is referred by versions of the content instead of by versions of the formatting. When the references are resolved, the selected version of the image must have a different format depending on the user device. However, device-dependent formats are usually handled by the formatting part of a page.

In order to ensure the generation of a consistent page, the author can specify criteria using the environment properties. In figure 8, the *outputformat* and *getcontentlanguage* environment properties are shown (between curly brackets on top of the figure). These properties are denoted in the criteria as properties of an *environment resource*, however it actually does not exist. The *getcontentlanguage* property indicate that the user prefer a page with content in English. The *outputformat* property indicate that the user is using a cellular telephone to display the page, thus a page with WML format is required.

A local criterion using the environment property *outputformat* was associated by the author to the reference between `news.page` and `news.xslt` (between square brackets in figure 8). It requires that the chosen XSLT version must generate the same format required by the request context (represented by the environment resource). This criterion is able to adapt the formatting of the page. An analogous criterion was associated to the reference between `news.page` and `news.xml`.

Turning back to the problem of choosing a version of the images referred by the content versions, an author can use the environment properties to specify a suitable criterion. In this case, a global criterion similar to that shown in figure 7 is used. This criterion (in figure 8) is slightly harder than that, however it does the job. It requires that, if the target version of a reference doesn't contain an XML document (i.e., it is an image) and the environment requires a page in WML format, then the target version must contain a WBMP image. A similar criterion can be specified to select GIF images when generating HTML formatting.

In summary, the proposed model supports a page configuration process and is based on references and version selection criteria. Inputs to the process are:

- a URI, indicating the page to be configured;
- a set of environment properties, providing information about the context of the page request;
- two sets of global criteria (one for content and other for formatting), to guide the process in resolving the references.

References to version-controlled resources can have version selection criteria associated with it. Considering its scope, a criterion can be:

- global, associated just to the page; in the configuration process run-time, the global criteria defined by an author are added to the criteria input sets (generated by the server) and used to resolve all references to version-controlled resources;
- local, associated to any reference; it is used to resolve just the reference to what it is associated with.

A criterion can also be classified in:

- constraint, representing a consistency constraint that must be obeyed;
- preference, representing a preference that should be satisfied.

5. RELATED WORK

The importance of managing versions of hyperdocuments was recognized since the eighties when Halasz [16] advocated the use of versioning as an important requirement of hyperdocument systems. From then on, several version models for hyperdocuments have been proposed. In the version model for hyperdocument context, one notable published model was that of HyperPro [23], a hypertext system concerned to reduce the cognitive problems and disorientation aggravated by version proliferation of hyperdocument components. CoVer [14, 15](*Contextual Version Server*) is another much cited model that maintains context information with the versions that guides version creation helping in its identification. CoVer has an interesting task concept (composed of versions) that represents coordinated modifications to some objects (nodes and links) in order to implement a modification to the document. An interesting feature of NCM [26] (*Nested Context Model*) is the separation of content and formatting of hyperdocuments. Formatting (presentation specification in NCM) is associated to links and composite nodes and refers to the target node of links or components of composite nodes. GDOC [13, 22, 24] (*Document Management*) is a system for authoring and storage of structured documents to be used in an Intranet. GDOC does not just separate content and formatting but also schema (composition structure) of documents. Formatting can be shared by many documents, however, like NCM, GDOC does not keep formatting under version-control.

As shown, some models do not separate content and formatting of hyperdocuments. Others do it, but do not maintain versions of formatting components, which is important in page maintenance. We believe that our model is more general than those mentioned above as it separately treats content from formatting besides of providing a version management in order to distinguish stable versions from versions under development.

Recently, some XML-specific version models emerged, for instance RBVM [5] and SPaR [6, 7], in which the main focus is in storage and search of data represented as XML documents. Also, Xyleme [20] considers the issues of version management support for a Web Warehouse of XML Data. Our version model also adopts the same style concerning XML documents treatment, but, in contrast to the

approaches above, the best achievement of our model is to separate content and formatting in to distinct components.

In order to represent hyperdocument components, a version model must support a configuration process with information for selection of suitable versions of page components. There are many software configuration management systems that employ alternative approaches to handle versioning and configuration. A survey of configuration management issues can be found in [10, 29]. Configuration can be manual, as in COOP/Orm [19] or through an automatic process as it is done in software process. This process can select versions using some default criteria, for instance selection of the most recent version (e.g. RCS [27]). Otherwise, the user can specify features that selected versions are required to present, for instance Adele [11] and ICE [30, 31]. Návrat's method for version selection [21] is not a complete configuration process, but a way of selecting a version given a version set and a list of criteria. The configuration process adopted by our version model is based on Návrats method, that allows generating the dynamic pages during Web page adaptation process. However, some modifications to that method were made to adequate it to the proposed model. For instance, heuristic functions were replaced by expressions (criteria) and several constraints are allowed instead of just one. This simplifies the criteria specification because a complex criterion can be broken in to many ones. A constraint is not required, so the versions can be selected based just on preferences. After the configuration process have processed the page references, it has completely defined content and formatting of the page. Then the Web server (or an adaptation module) may use a XSLT processor to apply the configured formatting rules to the configured content and may send the configured page to the user. Thus, the dynamic page generation is the main contribution related to the configuration mechanisms presented in this paper.

6. CONCLUSION AND FUTURE WORK

This paper describes a version model that allows alternative versions, thus supporting adaptation and personalization of page content and formatting rules. The proposed model is consistent with the accepted standard WebDAV, simplifying its the implementation. A configuration process is also described and it is responsible for generating dynamic pages. Besides, the model provides resource properties and version selection criteria leading to an automatic configuration process. The main contribution of this paper is to provide a Web server infrastructure to accomplish version control and to support adaptation of Web pages in a way that is transparent to the user.

The proposed model differs from the others by separating Web content and formatting. In this aspect, the proposed model is very attractive since it can be used in maintenance of large Web sites, in which one of the major needs considering to automate site maintenance is to be able to independently adapt and personalize the Web page content and formatting.

We have partially implemented a prototype Web server to show the feasibility of our model and in the process have come up with our own version model and automatic configuration process. The prototype used just standards as WebDAV, XML, XSLT, XPath. However, we noticed a poor performance in the configuration process due to specific features of the implementation. Although it could be improved, the process relies on the XSLT processor that has a poor performance by nature. Then, it is important to investigate and evaluate alternative ways to improve the Web server performance. Initial ideas involve request predicting [25], a *priori* configuration and caching.

Ongoing work is on an extension of the presented version model in two directions. First, we are considering the generation of a schema or summary of the structure of a document in order to allow constructing a XSLT stylesheet that transforms correctly any XML document according to that schema. However, the proposed model does not require that XML content conforms to a DTD. We consider that this is not a practical requirement once that the structure of the documents undergo frequent changes, demanding that the DTD should also be update frequently. Then, our goal is to generate a schema from versions of an XML document, in a similar approach to that of *Data Guides*[12]. Second, we intend to allow the configuration process to include additional formatting fragments to the configured formatting, even if it is not directly referred by the formatting fragments. We expect that this can be done based on the structure of the configured content. Thus, it could be used to format specific parts of the content. For instance, if a specific fragment appears in the configured fragment, an stylesheet could be included in the configured formatting. In addition, it could be an alternative way to construct (configure) the formatting rules. Also, and not less important, we intend to introduce a formal semantics in order to precisely define the versioning and configuration process. This is required in order to investigate the version model issues rigorously.

7. REFERENCES

[1] T. F. Adbelzaher and N. Bhatti. Web content adaptation to improve server overload behavior. *Computer Networks*, 31(11-16):1563–1577, may 1999.

[2] M. Bieliková and P. Návrat. Modelling versioned hypertext documents. In B. Magnusson, editor, *Proceedings of the 8th System Configuration Management Symposium*, volume 1439 of *LNCS*, pages 188–197, Brussels, Belgium, july 1998. Springer.

[3] C. A. Brooks, J. Cooke, and J. Vassileva. Versioning of learning objects. In *IEEE Intl. Conf. on Advanced Learning Technologies*, pages 296–297, Athens, Greece, july 2003. IEEE Computer Society.

[4] P. Brusilovsky. Methods and techniques of adaptive hypermedia. *User Modeling and User Adapted Interaction*, 6(2-3):87–129, 1996.

[5] S.-Y. Chien, V. J. Tsotras, and C. Zaniolo. Efficient management of multiversion documents by object referencing. In *Intl. Conf. on Very Large Data Bases*, pages 291–300, Roma, Italy, Sept. 2001. Morgan Kaufmann.

[6] S.-Y. Chien, V. J. Tsotras, C. Zaniolo, and D. Zhang. Storing and querying multiversion XML documents using durable node numbers. In *Intl. Conf. on Web Information Systems Engineering*, pages 232–244, Kyoto, Japan, dec. 2001. IEEE Computer Society.

[7] S.-Y. Chien, V. J. Tsotras, C. Zaniolo, and D. Zhang. Efficient complex query support for multiversion XML documents. In *Conf. on Extending Database Technology (EDBT 2001)*, volume 2287 of *LNCS*, pages 161–178, Prague, Czech Republic, mar. 2002. Springer.

[8] J. Clark. XSL transformations (XSLT) version 1.0 - W3C Recommendation, nov 1999. Disponvel em <http://www.w3.org/TR/xslt>. Acesso em outubro de 2001.

[9] G. Clemm, J. Amsden, T. Ellison, C. Kaler, and J. WhiteHead. RFC-3253: Versioning extensions to WebDAV (Web distributed authoring and versioning), Mar. 2002. (<http://www.ietf.org/rfc/rfc3253.txt>).

[10] R. Conradi and B. Westfechtel. Version models for software configuration management. *ACM Computing Surveys (CSUR)*, 30(2):232–282, june 1998.

[11] J. Estublier and R. Casallas. *The Adele Configuration Manager*, chapter 4, pages 99–139. Number 2 in Trends in Software. John Wiley & Sons, England, first edition, jan. 1995.

[12] R. Goldman and J. Widom. Dataguides: Enabling query formulation and optimization in semistructured databases. In *Intl. Conf. on Very Large Data Bases*, pages 436–445, Athens, Greece, aug. 1997. Morgan Kaufmann.

[13] A. Grala and C. A. Heuser. GDOC: A system for storage and authoring of documents through Web browsers. In *Intl. Conf. of the Chilean Computer Science Society*, pages 115–124, Valpariso, Chile, nov. 1997. IEEE Computer Society.

[14] A. Haake. CoVer: A contextual version server for hypertext applications. In *European Conf. on Hypertext Technology*, pages 43–52, 1992.

[15] A. Haake. Under CoVer: The implementation of a contextual version server for hypertext applications. In *European Conf. on Hypertext Technology*, pages 81–93, Edinburgh, Scotland, sep. 1994. ACM Press.

[16] F. G. Halasz. Reflections on notecards: Seven issues for the next generation of hypermedia systems. *Communications of the ACM*, 31(7):836–852, july 1988.

[17] S. Iksal and S. Garlatti. Revisiting and versioning in virtual special reports. In *Workshop on Adaptive Hypermedia*, volume 2266 of *LNCS*, pages 264–279, Aarhus, Denmark, aug. 2002. Springer.

[18] M. Kay. *XSLT Programmer's Reference*. Wrox Press, UK, first edition, June 2000.

[19] B. Magnusson and U. Asklund. Fine grained version control of configurations in COOP/Orm. In *System Configuration Management*, volume 1167 of *LNCS*, pages 31–48, Berlin, Germany, 1996. Springer.

[20] A. Marian, S. Abiteboul, G. Cobena, and L. Mignet:. Change-centric management of versions in an xml warehouse. In *27th International Conference on Very Large Data Bases*, pages 581–590, Roma, Italy, sept 2001. Morgan Kaufmann.

[21] P. Návrat and M. Bieliková. Knowledge-controlled version selection in software configuration management. *Software - Concepts and Tools*, 17(1):40–48, mar 1996.

[22] M. Noronha, L. G. Golendziner, and C. S. dos Santos. Compartilhamento de componentes com verses em documentos estruturados. In *Braziliam Simposium on Database*, pages 319–333, oct 1998. (in Portuguese).

[23] K. Østerbye. Structural and cognitive problems in providing version control for hypertext. In *European Conf. on Hypertext Technology*, pages 33–42, 1992.

[24] C. P. Santos and C. S. dos Santos. Configuration of versioned documents. In *Intl. Conf. of the Chilean Computer Science Society*, pages 53–61, Talca, Chile, nov 1999. IEEE Computer Society.

[25] S. Schechter, M. Krishnan, and M. D. Smith. Using path profiles to predict HTTP requests. *Computer Networks*, 30(1-7):457–467, apr 1998.

[26] L. F. G. Soares, N. L. R. Rodrigues, and M. A. Casanova. Nested composite nodes and version control in an open hypermedia system. *Information Systems*, 20(6):501–519, sep 1995.

[27] W. F. Tichy. RCS - a system for version control. *Software - Practice and Experience*, 15(7):637–654, jul 1985.

[28] B. Westfechtel. A graph-based system for managing configurations of engineering design documents. *Intl. Journal of Software Engineering and Knowledge Engineering*, 6(4):549–583, dec 1996.

[29] B. Westfechtel, B. P. Munch, and R. Conradi. A layered architecture for uniform version management. *IEEE Transactions on Software Engineering*, 27(12):1111–1133, dec 2001.

[30] A. Zeller. A unified version model for configuration management. *ACM SIGSOFT Software Engineering Notes*, 20(4):151–160, oct. 1995.

[31] A. Zeller and G. Snelting. Unified versioning through feature logic. *ACM Transactions on Software Engineering and Methodology*, 6(4):398–441, oct. 1997.