Fine-grained, Structured Configuration Management for Web Projects

Tien N. Nguyen
Dept. of EECS
Univ. of Wisconsin, Milwaukee
tien@cs.uwm.edu

Ethan V. Munson
Dept. of EECS
Univ. of Wisconsin, Milwaukee
munson@cs.uwm.edu

Cheng Thao
Dept. of EECS
Univ. of Wisconsin, Milwaukee
chengt@cs.uwm.edu

ABSTRACT

Researchers in Web engineering have regularly noted that existing Web application development environments provide little support for managing the evolution of Web applications. Key limitations of Web development environments include line-oriented change models that inadequately represent Web document semantics and inability to model changes to link structure or the set of objects making up the Web application. Developers may find it difficult to grasp how the overall structure of the Web application has changed over time and may respond by using ad hoc solutions that lead to problems of maintainability, quality and reliability.

Web applications are software artifacts, and as such, can benefit from advanced version control and software configuration management (SCM) technologies from software engineering. We have modified an integrated development environment to manage the evolution and maintenance of Web applications. The resulting environment is distinguished by its fine-grained version control framework, fine-grained Web content change management, and product versioning configuration management, in which a Web project can be organized at the logical level and its structure and components are versioned in a fine-grained manner as well. This paper describes the motivation for this environment as well as its user interfaces, features, and implementation.

Categories and Subject Descriptors

D.2.9 [**Software Engineering**]: Configuration management;
H.5.4 [**Information Interfaces and Presentation**]: Hypertext / Hypermedia

General Terms

Documentation, Management

Keywords

Web engineering, version control, configuration management

1. INTRODUCTION

In only a decade, the World Wide Web has grown to significantly affect all aspects of our lives. Organizations from industry, government, education, entertainment, business, and services all use the Web to improve and enhance their operations. Even traditional information and database systems have migrated to the Web.

Many organizations have successfully developed large and high-quality Web sites, but others have failed or have struggled to avoid major failures. While some of these failures result from a lack of vision or short-sighted goals, others result from a flawed design and development process or poor management of development efforts [21]. Web developers commonly use ad hoc development processes that lack rigor, systematic techniques, sound methodologies, and quality assurance and may pay little attention to issues such as requirements analysis, quality, performance evaluation, configuration management, maintainability, and scalability [22]. In fact, Web development is often seen as being primarily an authoring task rather than an application development task. Therefore, systematic, disciplined approaches are only beginning to be applied to the development of high-quality Web-based applications.

WWW2004, May 17–22, 2004, New York, New York, USA.
ACM 1-58113-844-X/04/0005.

In response, the discipline of *Web engineering* has emerged, advocating a systematic approach to development of high quality Web-based systems [39]. It promotes the establishment and use of sound scientific, engineering and management principles in the development, deployment and maintenance of Web-based systems. To a large extent, Web engineering views Web development as an important variant of software development. As a result, many of the tools and practices already used to produce other software are still relevant, though Web engineering does have distinct practices that arise from the nature of hypermedia and the Web [33, 45].

One task that is important for software and for the Web is the management of the evolution and maintenance of Web-based applications. Dart [10, 11] has argued that Web systems would benefit greatly from the use of techniques from software configuration management (SCM). Web applications are software artifacts, and as such can benefit by making use of advanced version control and software configuration management technologies. But Dart has also pointed out that the evolution of Web applications presents special challenges that are not well-addressed by existing SCM systems. For our purposes, the most important of these challenges are:

Variety of Types: Web systems are built from a wide diversity of objects including documents in any of markup languages, document templates, style sheets, images, streaming media, animations, applets, and scripts. Unlike software engineering, where program source code is seen as the central artifact, in Web systems it is difficult to identify one type of object as most important. So, an SCM system for Web applications must be compatible with a wide variety of file types, including their templates and their corresponding editing tools. A versioning model which corresponds to the logical structure of these objects would also be desirable.

Intermixed Types: Furthermore, in Web applications it is common to intermix different object types in the same file. An example would be an HTML file that contains embedded CSS style commands and JavaScript. A versioning model that understands this would be helpful.

Transparency: The semantics of URLs and browsers places constraints on deployed file locations that are less often seen in traditional software. It can be useful to think about the system as having a different logical structure than what is imposed by these other constraints. Thus, an SCM system that can present logical models of the system structure, in addition to the structure of the deployed files will provide a more transparent interface.

Developers: Web systems are developed by diverse teams ranging from graphic artists to specialized software engineers. Thus, the SCM system's interface needs to be accessible to less technical users or they will resist its use, leaving important portions of the system outside the control of the SCM tools.

Rate of Change: Large Web systems appear to change faster than traditional software systems. Some sites, such as news sites, must show thousands of daily content changes. While substantive structural changes visible to end users may be more rare on these sites, they still appear to happen at a faster rate than in traditional software. Continuous fine-grained evolution is thus a distinguishing characteristic of Web systems [32, 39]. Thus, Web-oriented SCM must help developers understand changes at a very fine-grained level.

In addition, fine-grained change management in SCM tools must take into account the external and internal structures of a Web system [10]. *External structure* (also called *navigational structure*) of a Web system refers to the structure of a collection of Web documents with respect to hyperlinks among them. Web documents in a Web system are logically related and connected to each other via these hyperlinks. To facilitate the management of "dangling" links, the versions of Web documents need to be kept in synchrony. Therefore, the history of networks formed by Web documents and hyperlinks needs to be recorded as a Web application evolves. This navigational structure should be distinguished from the Web project's structure and architecture [29]. SCM tools need to allow developers to organize their Web project at the logical level according to their choice of design methodologies, and to support version control for the project's structure and architecture as well.

A Web document, either static or dynamic, has some *internal structure*. An HTML document has a tree-based syntactic structure, while program source code can be regarded as an abstract syntax tree (AST) of syntactic units. Researchers have been taking advantage of the internal structure of a Web page in order to process, visualize, search and retrieve information [23, 25, 50]. As the Internet moves toward XML [36], it is likely that Web documents will gain structure with semantics of growing importance. These structural semantics are described, at least to some extent, by Document Type Definitions (DTD) or XML Schemas, which are also evolving objects. So, Web content must evolve along with its structural rules. Managing the evolution of structures and contents of a Web document will result in various benefits in processing, visualizing, and retrieving Web content. Traditional SCM tools are not well-suited to this task because they often use a line-oriented model of internal changes that disregards these internal structures of Web documents. An SCM tool that can version data objects and handle change management at the logical level is needed.

To investigate the application of SCM technology to Web engineering, we have modified an extensible integrated development environment, the Software Concordance (SC), to be suitable for Web application development. To meet the requirements and to address the problems mentioned above, SC uses a tree-based fine-grained version control framework and a product versioning SCM system to manage the evolution and maintenance of Web applications. The SC environment has been extended to support many types of Web objects including HTML and XML documents, audio clips, images, graphics and animations in Scalable Vector Graphics (SVG) format, and program source code in Java, Java Applet, and JavaScript. The environment can be further extended to integrate additional editors supporting other types of documents.

A structure-oriented approach is used that represents all Web documents as tree-structured objects that are versioned in a fine-grained manner. On top of this fine-grained, tree-structured versioning framework, a product versioning SCM system, named *Molhado*, has been built that provides configuration management support for Web development projects in which the Web project is versioned as a whole entity. The Web system can be structured and versioned at the logical level, independent of the physical locations of its components on the file system. A simple graphical user interface (GUI) has been designed to help Web developers in their versioning and configuration management tasks. Networks of static Web documents and their links are also versioned in a fine-grained manner. A fine-grained content change management tool has been developed, allowing Web developers to track the history and changes of any structural and logical unit in a Web document.

The next section will discuss related work on applying version control and configuration management to Web engineering. Section 3 describes the SC document representation. Our fine-grained version control framework is presented in Section 4 and its application to Web content change management is discussed in Section 5. Section 6 covers the configuration management system of SC environment while the SC editing system is described in Section 7 and the last section presents our conclusions.

2. RELATED WORK

Researchers and vendors in the configuration management area are taking different approaches to SCM for the Web. All have added Web functions to their SCM tools by offering access to some or all SCM functionality through a browser [11]. WebSynergy [60] provides a Web front-end into all of its existing SCM capabilities as well as Web authoring tools. Similar to our approach, MKS's WebIntegrity [59] integrates its version control facilities with an authoring tool, while in Merant's PVCS [35], version control is the core part and is separate from authoring systems. However, both of them version control at the file level. StarTeam [51] is Web-enabled with the intention of tool integration. TrueChange [57] provides content change management along with its version control, but with less focus on configuration management. Serena's eChangeMan [16] is focused on process management and change tracking. Rational's ClearCase [31] provides configuration management via "tagging" all files in the same configuration with the same label for later retrieval. ClearQuest [7] is a change request management tool of ClearCase, coordinating many developers in changing Web documents. Content change management in SourceSafe [49] is line-oriented. Computer Associate's CCC/Harvest [5] pays considerable attention to supporting collaboration among distributed development teams. Perforce [41] is more lightweight than other SCM tools and it has the ability to migrate repositories from other SCM tools such as CVS and PVCS into its internal repository. Although all of these SCM tools have distinguished and valuable features, they are focused on version control of files, rather than on configuration management for a Web project as a whole. None of them supports fine-grained change management at the logical level and all content change management is line-oriented.

On the other hand, some *Web application development environments* (also called *Web authoring environments* or *Web content man-*

agement environments) have realized that SCM practices need to be incorporated into their tools. FrontPage [19], Macromedia's DreamWeaver [15], and ColdFusion [8] have no built-in version control support. Other tools, such as StoryServer [52] and TeamSite [54], are designed to support many aspects of Web development, with particular strength in supporting collaborative work. TeamSite provides visual differencing tools so that two versions of the same content can be examined side by side. Inso's DynaBase [27] is an integrated content management and publishing platform for Web applications. Similar to our approach, it is XML-based, allowing better management and reuse of data. In DynaBase, configuration management use the "tagging" technique also seen in ClearCase. Configurations act like a bill-of-materials for the Web site, enumerating which items are included in the site and which version of each item is in use. ArticleBase [1] integrates content management and version control into an authoring and publishing system. Its version control support is on file basis and no configuration management is provided. None of these systems allows developers to structure their Web project and its objects at the logical level depending on their choice.

OOHDM-Web [44], an environment for the development of Web-based applications, is focused on the hypermedia design methodology defined by the OOHDM model [45]. CGILua [6] is a tool for developing dynamic HTML pages and manipulating input data from forms, which enables the use of embedded code in normal HTML files. RMCase [12] is a CASE tool that supports the complete life cycle of a Web application. Its main goal is to support the process model for developing Web applications. Similarly, the WebComposition process model and its related product Eurovictor [20] focused on open process model that allows for the integration of processes and reuse of components. The JESSICA system [2] provides a XML-based modeling language for the automatic mapping from the design to Web-resources. AriadneTool [37], a design toolkit for hypermedia applications, allows a designer to generate dynamically XML templates. No version control support is provided in these systems.

Many research in *versioned hypermedia* [24, 26, 61] community have focused on version control in the presence of hyperlinks. However, the main goals of their versioned hypermedia systems often do not include supports for Web application development. Therefore, supports for program source code are very limited. The GAIA framework [30] combines *open* hypermedia and versioning paradigms. GAIA builds versioning mechanisms on top of traditional open hypermedia architecture to support versioning for links, documents and anchors. RHYTHM [34] is a prototypical distributed hypertext system that tackled problems arising from distribution and versioning, both from a structural and from a cognitive point of view. It is better suited for hypermedia systems than Web-based applications where program source code is also an important part. Similar to RHYTHM, HyperProp [48] provides versioning supports for hypertext authoring systems without paying much attention to program source code. HyperPro [40] provides versioning supports of program objects as small as Pascal procedures. However, its goal is to map internal structure of a program into a hypertext and no program analysis is supported. In RCS-based HyperWeb [18], the smallest versionable object was a file.

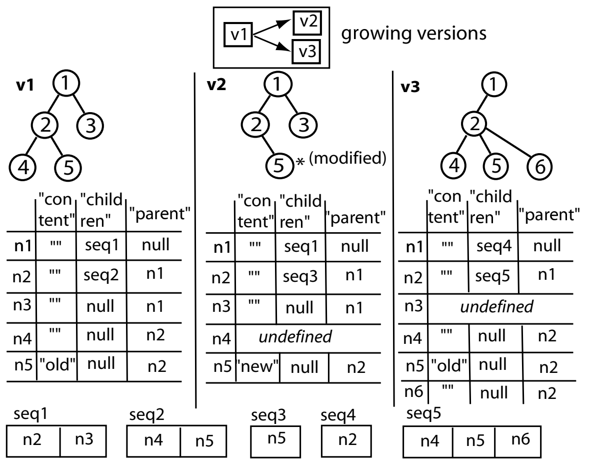
To improve the authoring and browsing features for versioned contents of Web pages, some researchers followed the language-oriented approach. They have attempted to change the Uniform Resource Locator (URL) of a Web page to include a version identifier [42, 46]. They use existing Web infrastructure such as forms, Java applets, and plug-ins to create a user interface for revision control systems on the server. Vitali and Durand proposed VTML (versioned text markup language) [3] to express change operations for HTML documents. For example, they introduced two new tags (INS and DEL) to express insertion and deletion. The WebDAV protocol [62] is an extension of the Hypertext Transfer Protocol (HTTP) to support distributed authoring and versioning. It extends HTTP to include versioning operations for Web pages. WebDAV is designed to meet requirements [47] for distributed authoring and versioning on the Web environment.

Although functionality is not consistent across the tools, most of existing tools have very little or no support for configuration management. Configuration management support is limited to version control of files. The similarities among them include supports for Web and scripting languages, templates and stylesheets, versioning of files, roll-back of complete sites via backup, audit logging, workflow support for collaborative work, commercial database interfaces, and minimal change tracking and management support. None of them cares about the external and internal structure of Web content in the sense described in the previous section. Their content change management is coarse-grained, with differencing done on a line-by-line basis. None of them have ability to allow developers to logically structure their Web projects.

3. REPRESENTING WEB CONTENT

Web content can consist of data objects, code, and component libraries. Examples of data objects are data files, documents, images, audios, and videos. Code can be active controls and scripts that can be embedded into an HTML page. Component libraries are reusable codes such as JavaBeans and Microsoft Foundation Classes. Among Web documents, hyperlinks exist that can point to any page in the Web. Templates such as stylesheets, which are written in some languages, enable separation of content and its presentation style. To provide fine-grained version control and content management for static documents, the SC environment follows the principle that textual information such as HTML, XML documents, and program source code are treated as structured objects and versioned in a fine-grained manner, while binary information such as audio clips, video clips, and component libraries are considered to have no internal structure and are versioned on a file basis.

The SC's document representation uses a *structure-oriented approach* where each Web document is a *document tree* composed of *document nodes* (also referred to as *structural units*). Each document node can be associated with multiple pairs of attribute name and value. The document trees are versioned in a fine-grained manner, which is described later. Structural units in a document carry different logical senses depending on the document's type. Program source code and scripts are represented as ASTs, therefore, in a program, a structural unit is equivalent to a syntactic unit such as a class, method, or statement. On the other hand, in an HTML or XML document, depending on the document's DTD, a document node might represent a section, paragraph, sentence, or phrase. In either case, a document node represents the logical semantics encoded by an XML or HTML element. This document tree representation is logically equivalent to the structure supported by Document Object Model [13], though as described in the next section, the implementation details are quite different. A conversion facility is implemented to allow the import and export of DOM-compatible documents into the SC's representation. The maturity of XML technology [63] allows SC to support a wide variety of information types, including graphics and animation written in SVG [53], Unified Modeling Language (UML) [43] diagrams, Java programs, and HTML documents with embedded scripts. The implementation of this structure-oriented approach uses the Fluid Internal Representation [4], which is described in the next section.



v1

	"content"	"children"	"parent"
n1	""	seq1	null
n2	""	seq2	n1
n3	""	null	n1
n4	""	null	n2
n5	"old"	null	n2

v2

	"content"	"children"	"parent"
n1	""	seq1	null
n2	""	seq3	n1
n3	""	null	n1
n4	*undefined*		
n5	"new"	null	n2

v3

	"content"	"children"	"parent"
n1	""	seq4	null
n2	""	seq5	n1
n3	*undefined*		
n4	""	null	n2
n5	"old"	null	n2
n6	""	null	n2

seq1		seq2		seq3	seq4	seq5		
n2	n3	n4	n5	n5	n2	n4	n5	n6

Figure 1: Tree versioning

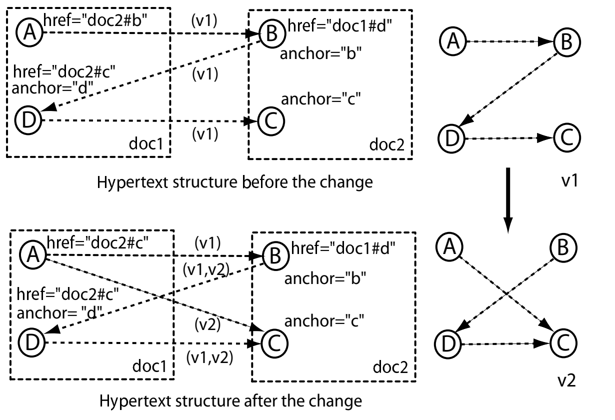


Figure 2: Versioning for hypertext structure

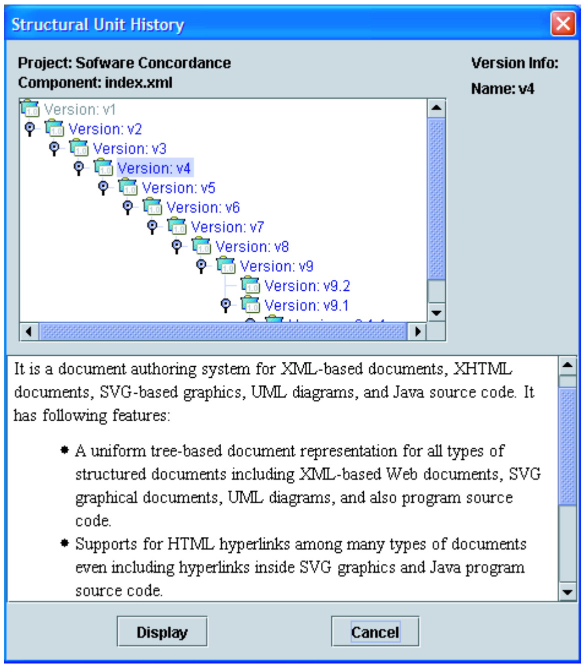
4. FINE-GRAINED VERSION CONTROL

The SC environment makes extensive use of the data and version model supported by the Fluid system [4]. The primitive data model used by Fluid is called the Fluid Internal Representation (IR). The Fluid IR is based on two notions: *nodes* and *slots*. A node is the basic unit of identity and is used to represent objects. A slot is a location that can store a value, possibly a reference to a node. A slot can exist in isolation but more typically slots are attached to nodes, using an *attribute*. An attribute is a mapping from nodes to slots. An attribute may have particular slots for some nodes and map all other nodes to a default slot. The Fluid data model can thus be regarded as an attribute table whose rows correspond to IR nodes and columns correspond to attributes. The cells of the table are slots. Once we add versioning, the table gets a third dimension: the version. There are three kinds of slots. A *constant* slot is immutable; such a slot can only be given a value once, when it is defined. A *simple* slot may be assigned even after it has been defined. The third kind of slot is the *versioned slot*, which may have a different value in different versions. Nodes are used to implement structural units (document nodes) in Web documents, while slots and attributes are used to represent attributes of a document node and their values. For example, an *href* attribute has been defined to represent hyperlinks in an HTML document. Links are created by defining a value for an *href* attribute that is a URL. An additional attribute (*anchor*) is defined for anchors within documents. Similarly, links can also be defined in a Java program since it is represented by an AST. Importantly, this approach to binding hyperlinks to source code does not interfere with program analyses, which ignore these hyperlink attributes. Also, the *href* attribute is defined as versioned slots, therefore, the changes occurring to the *href* attribute can be recorded over time.

In Fluid's version model, a *version* is a point in a tree-structured discrete time abstraction, rather than being a particular state of a system component. From the user's point of view, a uniform global version space is maintained across the entire Fluid IR data model. This is a form of *product versioning* [9], where there is one global version space for the whole Web project, while in file-based versioning systems such as CVS [38] or RCS [56], each file has its own version history. On the other hand, the Fluid version model operates at a much finer internal granularity — the versioned slot level — to effectively store, retrieve, compare, and construct versions. The version model is state-based. However, revisions and variants are not distinguished. The set of versions is organized in a tree, called the *version tree*, with the root of the tree being the initial version of the Fluid IR world. The *current version* is the version designating the current state of the Fluid IR world and any version may be made current. Internally, every time a versioned slot is assigned a (different) value, a new version is produced, derived from and branching off the current version. However, from the users' point of view, a new version is only recorded if users issue an explicit command.

A fine-grained tree-based versioning technique is developed for document trees as well as a Web project's structure, which is often hierarchically organized. Trees are represented via nodes, slots, attributes, and sequences. A *sequence*, which has a unique identifier, is a container with slots of the same data type. Sequences may be fixed or variable in size and share common slots together. A tree is defined with two main attributes: 1) "children" attribute that for each node gives the sequence of children for the node, and 2) "parent" attribute that for each node gives its parent. The details are illustrated via an example in Figure 1. In the example, "content" attribute is also defined to hold string value for each node if any. Assume that there are three versions: *v1*, *v2*, and *v3*. Versions *v2* and *v3* are branching off version *v1*. The shape of the tree at the three versions is shown. Version *v1* has five nodes numbered from 1 to 5. Version *v2* has two differences from the version *v1*: node 4 was deleted and the content of node 5 was changed. Version *v3* has an inserted node (node 6) and node 3 was deleted. The values of versioned slots in the attribute table changed to reflect modifications to the tree in these versions. For example, at the version *v2*, the "content" slot (i.e. the slot defined by the attribute "content") of node 5 contains a new value (the string "new"), and the "children" slot of node 2 contains a reference to a new sequence object (*seq3*). *Seq3* has only one slot, which contains a reference to node 5 since node 4 has been deleted. If there is a request on values of slots associated with node 4 at *v2*, a run-time error will be reported. Note that other attributes might be defined for nodes.

A *hypertext structure* is the network consisting of a set of Web documents and the hyperlinks connecting them in a Web system (external Web pages are excluded). A hypertext structure is versioned in the same manner as versioning for document trees. Suppose that we have two documents: *doc1* and *doc2*, connected as in Figure 2. At version *v1*, *A* links to *B*, *B* links to *D*, and *D* links to *C*. Suppose that now node *A* points to node *C*, and a new version *v2* is created. That is, at the version *v2*, the value of the *href* attribute of node *A* is "doc2#c". It means that the link between *A* and *B* is no longer present at *v2*, while *C* can be reached directly from *A* now. The links between *B* and *D*, and between *D* and *C* are still valid.



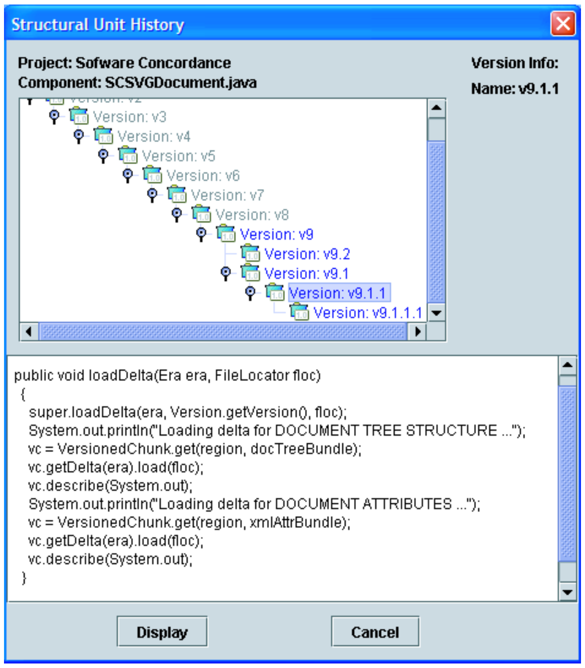
Project: Sofware Concordance
Component: index.xml

Version Info:
Name: v4

It is a document authoring system for XML-based documents, XHTML documents, SVG-based graphics, UML diagrams, and Java source code. It has following features:

- A uniform tree-based document representation for all types of structured documents including XML-based Web documents, SVG graphical documents, UML diagrams, and also program source code.
- Supports for HTML hyperlinks among many types of documents even including hyperlinks inside SVG graphics and Java program source code.

Figure 3: History of a section



Project: Sofware Concordance
Component: SCSVGDocument.java

Version Info:
Name: v9.1.1

```
public void loadDelta(Era era, FileLocator floc)
 {
   super.loadDelta(era, Version.getVersion(), floc);
   System.out.println("Loading delta for DOCUMENT TREE STRUCTURE ...");
   vc = VersionedChunk.get(region, docTreeBundle);
   vc.getDelta(era).load(floc);
   vc.describe(System.out);
   System.out.println("Loading delta for DOCUMENT ATTRIBUTES ...");
   vc = VersionedChunk.get(region, xmlAttrBundle);
   vc.getDelta(era).load(floc);
   vc.describe(System.out);
 }
```

Figure 4: History of a method

Depending on the current version, the correct destination node of a link is implicitly determined. Therefore, the shape of the network is properly exposed in the current version.

5. CONTENT CHANGE MANAGEMENT

Using the versioning scheme described in the previous section, the SC environment provides fine-grained content change management that allows Web developers to track the history of any structural unit (i.e. document node) in a Web document and to compare two arbitrary versions of Web content at levels: 1) structural unit, 2) Web document (except for binary data), and 3) the Web system in both structured and line-oriented manners. In Figure 1, depending on the current version, the shape of a subtree rooted at a document node is exposed. For example, if the current version is set to *v2*, the subtree at node *2* contains only itself and the child node *5* whose "content" slot contains the new value. Based on this mechanism, the SC environment manages the evolution of Web content at a fine granularity. Web developers can select any *structural unit* in any type of structured document (XML, HTML, Java, SVG, UML) displayed in the SC editor and to view its state in different versions. Figure 3 shows a past version of a section in an XML document. When users move the cursor to a different version, the correct content of that section is shown in the lower window. Note that the section has not been created at the version *v1*, therefore, it is "disabled". Figure 4 shows the history of the *loadDelta* method of a Java program. If developers select the root node of a document tree, the history of the document as a whole will be displayed in the same manner. Tracking the history of a Web system as a whole will be discussed in Section 6.

In order to track the fine-grained changes between two arbitrary versions, the SC environment employs a "dirty bit" mechanism in Fluid version engine, called the *Versioned Unit Slot Information* (VUSI). With the VUSI mechanism, SC can tell whether there is a change in one or many attributes of a node between any two arbitrary versions. It can also determine whether the structure of the subtree rooted at a node has been modified between any two arbitrary versions. VUSI has "boolean" versioned slots attached to nodes in a tree. The values of slots are set to false initially, indicating nothing has changed. When the values of attributes associated with a node or the tree structure at the node are changed, the value of the associated VUSI slot of the node will be set to true. Since a VUSI slot is versioned, the mechanism works for any two arbitrary versions (not necessarily predecessor or successor of each other). The VUSI mechanism is also flexible enough to enable the creation of many VUSI slot types for any set of attributes (including children and parent attributes) and to support a variety of behaviors for marking the "dirty bits" for nodes. For example, a VUSI slot type can be defined to track changes occurring to the *href* attribute, while another type keeps an eye on *bgcolor* and *fgcolor* of a table. Another example is that when an attribute of a node is modified, a behavior can be defined such that it marks not only the VUSI slot associated with the node, but also the VUSI slots of its ancestor nodes. This behavior allows for the detection of changes occurring in a subtree under a node.

Based on this mechanism, the SC environment includes a comparison (i.e. *diff*) tool which can show the differences between two arbitrary versions of a document node, of a Web document, and of the Web application as a whole. SC visually displays the differences in both structural and line-oriented manners. Figure 5 shows structural changes in a Java program. A small icon is attached to a document node icon, showing its changing status from version *v7* to version *v8*: either it has been modified (a tree icon), inserted (an "i" icon), deleted (an eraser icon), or moved (a "truck" icon). For example, the methods "getRoot" and "setRoot" have been deleted (see the left window), the methods "numChildren" and "parent" have been added at version *v8* (see the right window), while the body of the method "doAnalysis" has been modified to add a new expression (see the right window). Developers can also select a structural unit and choose an option to display changes of a text node in line-by-line fashion similar to ViewCVS [58]. A structural comparison between two versions of an HTML document is shown

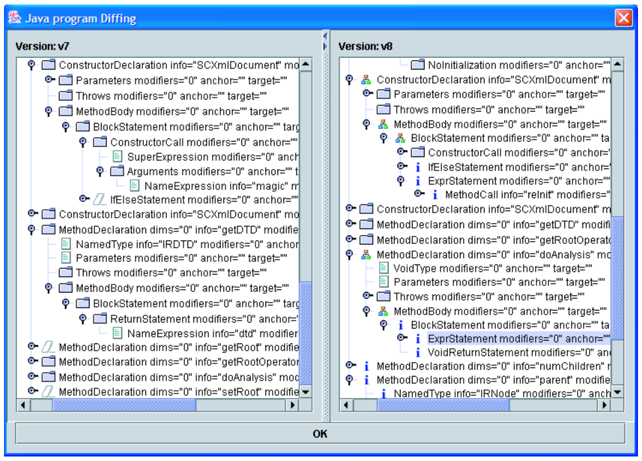

Java program Diffing
Version: v7
ConstructorDeclaration info="SCXmlDocument" mo
Parameters modifiers="0" anchor="" target=""
Throws modifiers="0" anchor="" target=""
MethodBody modifiers="0" anchor="" target=""
BlockStatement modifiers="0" anchor="" targ
ConstructorCall modifiers="0" anchor=""
SuperExpression modifiers="0" anch
Arguments modifiers="0" anchor="" t
NameExpression info="magic" m
IfElseStatement modifiers="0" anchor=""
ConstructorDeclaration info="SCXmlDocument" mo
MethodDeclaration dims="0" info="getDTD" modifie
NamedType info="IRDTD" modifiers="0" anchor
Parameters modifiers="0" anchor="" target=""
Throws modifiers="0" anchor="" target=""
MethodBody modifiers="0" anchor="" target=""
BlockStatement modifiers="0" anchor="" targ
ReturnStatement modifiers="0" anchor="
NameExpression info="dtd" modifier
MethodDeclaration dims="0" info="getRoot" modifie
MethodDeclaration dims="0" info="getRootOperator
MethodDeclaration dims="0" info="doAnalysis" moc
MethodDeclaration dims="0" info="setRoot" modifie
Version: v8
NoInitialization modifiers="0" anchor=""
ConstructorDeclaration info="SCXmlDocument" m
Parameters modifiers="0" anchor="" target=""
Throws modifiers="0" anchor="" target=""
MethodBody modifiers="0" anchor="" target=""
BlockStatement modifiers="0" anchor="" ta
ConstructorCall modifiers="0" anchor="
IfElseStatement modifiers="0" anchor="
ExprStatement modifiers="0" anchor="
MethodCall info="relnit" modifiers="
ConstructorDeclaration info="SCXmlDocument" m
MethodDeclaration dims="0" info="getDTD" modifi
MethodDeclaration dims="0" info="getRootOperat
MethodDeclaration dims="0" info="doAnalysis" mc
VoidType modifiers="0" anchor="" target=""
Parameters modifiers="0" anchor="" target=""
Throws modifiers="0" anchor="" target=""
MethodBody modifiers="0" anchor="" target=""
BlockStatement modifiers="0" anchor="" ta
ExprStatement modifiers="0" anchor="
VoidReturnStatement modifiers="0" an
MethodDeclaration dims="0" info="numChildren" r
MethodDeclaration dims="0" info="parent" modifie
NamedType info="IRNode" modifiers="0" anch
OK

Figure 5: Java program comparison

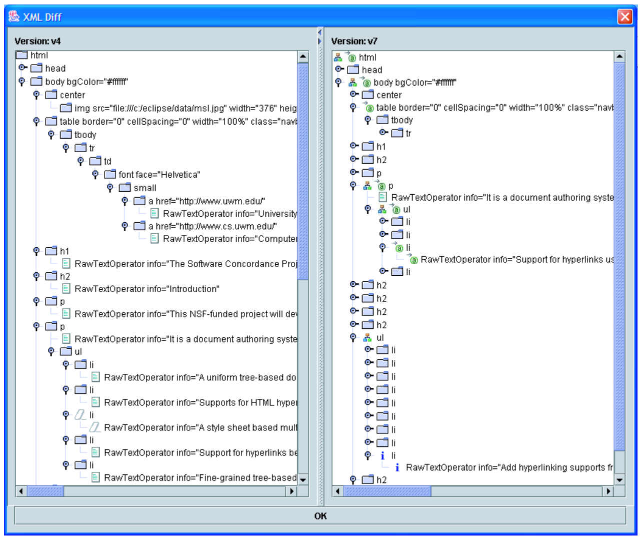

XML Diff
Version: v4
html
head
body bgColor="#ffffff"
center
img src="file:///c:/eclipse/data/msl.jpg" width="376" heig
table border="0" cellSpacing="0" width="100%" class="navb
tbody
tr
td
font face="Helvetica"
small
a href="http://www.uwm.edu/"
RawTextOperator info="University
a href="http://www.cs.uwm.edu/"
RawTextOperator info="Compute
h1
RawTextOperator info="The Software Concordance Proj
h2
RawTextOperator info="Introduction"
p
RawTextOperator info="This NSF-funded project will de
p
RawTextOperator info="It is a document authoring syste
ul
li
RawTextOperator info="A uniform tree-based do
li
RawTextOperator info="Supports for HTML hyper
li
RawTextOperator info="A style sheet based mult
li
RawTextOperator info="Support for hyperlinks be
li
RawTextOperator info="Fine-grained tree-based
Version: v7
html
head
body bgColor="#ffffff"
center
table border="0" cellSpacing="0" width="100%" class="navt
tbody
tr
h1
h2
p
p
RawTextOperator info="It is a document authoring syste
ul
li
li
li
RawTextOperator info="Support for hyperlinks us
li
h2
h2
h2
h2
ul
li
li
li
li
li
li
li
li
li
RawTextOperator info="Add hyperlinking supports fr
h2
OK

Figure 6: HTML document comparison

in Figure 6. Some of document nodes have been modified in terms of both their attributes and structures, for example, the "body" of the document (an "a" icon and a tree icon are both attached to the node icon). Meanwhile, the "table" node icon has only an "a" icon since only the color attribute of the table has been changed.

Figure 7 shows the changes in the structure of a Web project between two versions. As in the document comparison tool, the icon next to a document's entry shows the change in its status from one version to another. With this tool, developers are able to track changes in Web documents and program source code in a fine-grained manner. Therefore, it helps them to maintain better their Web applications. Also, incremental program analyses can take advantages of this fine-grained change tracking mechanism to optimize processes. It is very cumbersome for existing SCM systems that heavily depend on line-by-line comparison between versions to build this sort of fine-grained Web content change management.

6. CONFIGURATION MANAGEMENT

Since Web documents are interrelated and connected to each other via hyperlinks to form a Web application, version control

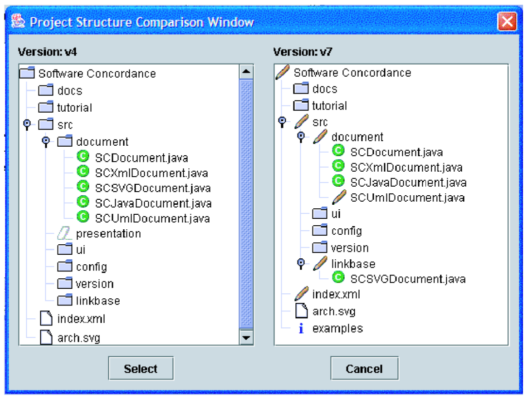


Figure 7: Project structure comparison

for Web content as individuals is not sufficient. A configuration management system is needed to manage the evolution of a Web application as a whole. The SCM system of the SC environment, Molhado, is built on top the Fluid version model to provide configuration management services for developers to manage their Web projects. This section describes the Molhado SCM system and how a Web project can be logically structured and versioned.

6.1 Web project structuring and versioning

6.1.1 *Important abstractions*

Molhado's three main abstractions are *components*, *projects*, and *configurations*. A *component* is a logical unit that is named, can be saved and loaded, and exists within the version space of a Web application project (Web project for short). A component may be a document, a class, a package, a module, a file, or a directory depending on the development framework being used. In SC, the Web documents described earlier are components whose internal structure is versioned at a fine granularity. A *project* is a named entity that represents the version history of a Web project. It is not a version of the project, but rather is used to retrieve the correct project version (including both structure and components). The project contains within itself a tree to represent the project structure. Each node in the tree is associated with a slot that contains a reference to a component. The project structure is versioned using the same tree-based versioning scheme described in Section 4. A *configuration* is a particular version of a project.

6.1.2 *Web project structuring*

Since a slot associated with a node in a project tree can contain a reference to any component, the Web project and its components can be organized logically. Figure 8 shows an example of a logical structure of a Web project. Each node in the project's tree has a slot referencing to a component. In this example (the University of Wisconsin (UW)'s Web site), there are directory components, Java class components, and HTML document components. The "root" directory component of this Web project consists of directory component "code" and two HTML document components "index.html" and "UW.html". Nodes in the subtree at the "code" directory component are associated with slots containing references to Java class components *A*,*B*, *C*, *D*, and *E*. This forms a class hierarchy under the "code" directory component. On the right hand side part of Figure 8, the Web pages of this Web project are logically structured according to the UW's campuses in the first level and then to colleges and schools in the second level.

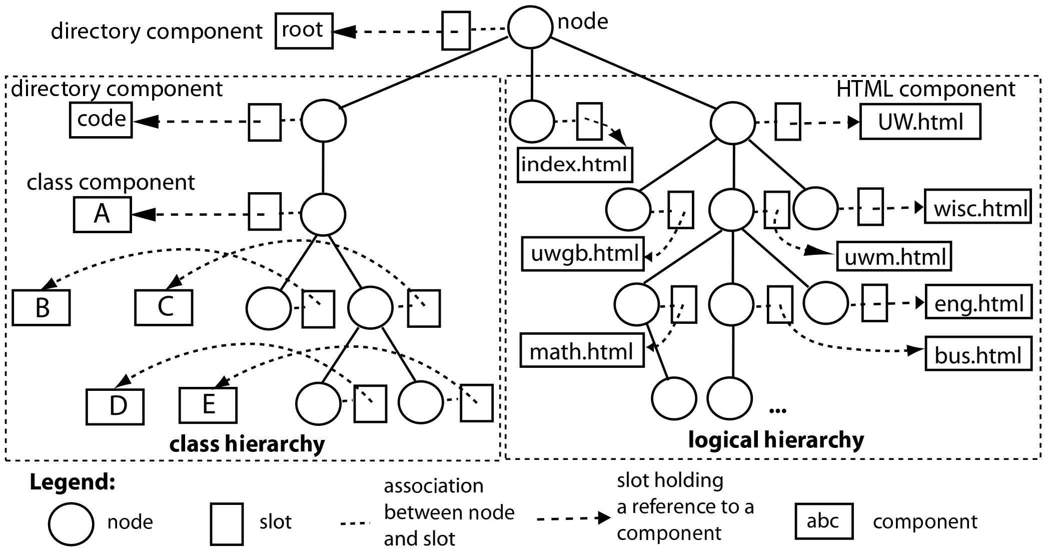


Figure 8: Logical organizations in a Web project

The directory-document hierarchy and components themselves are all logical. This means that directories and documents do not necessarily correspond to any directories and files on a file system. Therefore, in Figure 8, the *class* component "A" (not a *document* or a *file*) can be contained within directory component "code". This physical-independent organization of Web documents allows for accessing to Web content without having to know the actual physical address of the files. This ability is very important for large-scale Web applications whose documents may be distributed across many servers. Figure 8 also shows that Molhado has the ability to support multiple logical organizations of a single Web project. For example, program source code can be structured according to the developers' choice of design and implementation methodologies. Meanwhile, HTML documents can be organized into either a normal directory-document hierarchy based on a file system, or a logical hierarchy depending on the chosen development model such as OOHDM [45] or RMM [28]. Figure 9 shows the structure of the Web project of our research in which Java programs are structured as a package-class hierarchy and HTML documents are organized into a directory-document hierarchy. New component types can be easily added into Molhado to support different logical structures.

6.1.3 *Web project versioning*

Our project versioning approach, where a Web project is versioned as a whole, implies that versioning of a component is subsidiary to that of the project to which the component belongs. This is in contrast to the composition model [17] in most of existing SCM systems, where the project version (i.e. configuration) is dependent on the versions of each of its components. This project versioning approach always assures the construction of a consistent configuration since when a project version is chosen as the current, the project's tree will be correctly retrieved and versioned slots associated with nodes in the tree will refer to appropriate components at the current version as well. Then, the internal structure of each component and contents of slots will also be determined at the current version as mentioned in Section 4. This approach also avoids the complexity of using version selection rules in composition-based SCM systems and traditional versioned hypermedia systems [61], which are used to select the correct versions of components to be included in a version of a Web project.

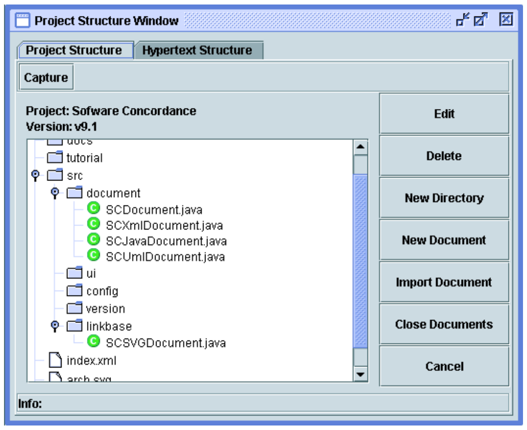


Figure 9: Project structure window

6.2 The operational model

This section discusses the operational model for a user during a transaction. Via GUIs, the user can open an existing Web project. The user can view the history of the Web project in a project history window. After selecting the working version, the SC system displays the project's structure and its components in a project structure window (see Figure 9). The version that is initially displayed in the project structure window is called the *base version*.

Via this window, the user can manipulate the project structure. If any modification is made to the project structure or its components at this base version, a new internal version in Fluid is temporarily created as a branch from the base version (the word "modified" will be attached to the base version's name), and the project structure window will now show information about this derived version. The user can choose to discard any derived version (i.e. any changes to the base version), or to issue a command to *capture* the project's state in a particular version. Intermediate internal versions from the base version to the newly captured version are discarded. A unique name within the Web project version space is assigned to the newly captured version either by the user or by the system. Bookkeeping information such as dates, authors, and descriptions can be attached to the new version for later retrieval. The captured version plays the role of a checkpoint version which the user can retrieve and refer to and becomes the new base version of the project structure window. However, no data is saved after a capture.

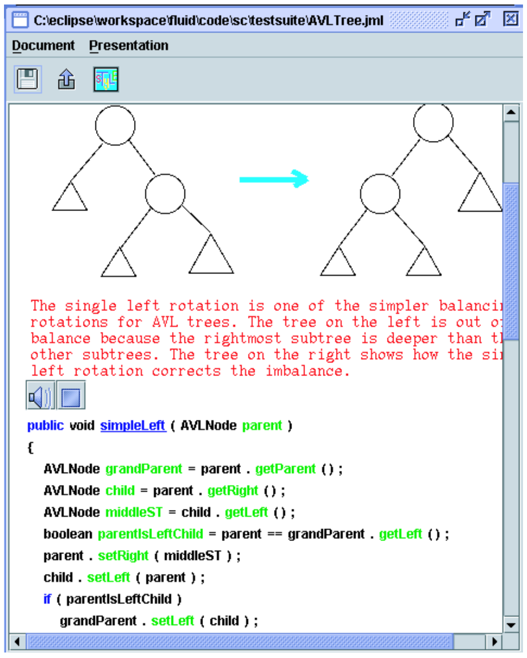
While working on one version of a Web project, the user can always switch to work on any other version. Switching to work or to view a different version can be done explicitly or implicitly, whether or not the current working version has been captured. If the user moves the mouse focus to a component editing window, or to a project structure window, the working version is automatically set to the version that the window is displaying. The user can also explicitly select a different version from the project history window and open it. Any windows showing the old version are still available should the user want to do additional work on that version. It implies that there may exist many windows displaying different versions (captured or uncaptured) in the same editing session. The user is not only allowed to view, but also to modify the new working version. In this case, an additional derived version will be branched off from the new working version. This switching feature allows the user to work on many versions at the same time during one session.

The user may *commit* changes to the project at any time. *Commit* is the command with which the user saves all changes made during an editing session since the previous commitment. Upon issuing this command, the user is asked which uncaptured versions should be saved and the chosen versions are then saved to a file system. Only the differences are stored. The user may also save complete version snapshots, which can improve version access time. In current system, each user can store his/her own data files for a Web project anywhere in a file system. Each user does not see changes from others. Therefore, no locking mechanism is needed. Users can share the data files and using merging tools to collaborate. A better collaboration mechanism is being implemented using a central versioning repository similar to CVS [38] with an "official" version graph. To branch from a version, the user just needs to copy data files for that version to his/her own workspace. The user can create many "private" versions. Finally, the user can copy meaningful versions back to the central repository.

7. SC EDITING SYSTEM

This section describes the details of the SC editing system for Web content. The editing system is an integration of a structured document editor for XML, HTML, and plain text documents, a syntax-recognizing Java program editor, an SVG graphic and animation editor, and a UML editor. The SC environment is compatible with XML-based document editing environments since it supports the integration of editors for new document types whose internal representation is XML-compatible. For example, the Thorn UML editor [55] and DrawSWF SVG editor [14] were easily integrated into the SC editor because their document representations are XML-based. Integration requires only that a new editor follows a simple plug-in protocol. The SC environment uses a Document Object Model (DOM) [13] parser to import XML-based documents, converts the DOM trees into the SC document tree representation, and then version controls them as described earlier.

All Web documents and their versions mentioned in Section 3 are stored according to the Fluid's persistence model [4]. In the Fluid's persistence model, a *persistent entity* is the basic unit of information that can be stored and then loaded. The persistence model uses the *forward direct delta* technique to store differences between versions. The data files for versioned persistence are immutable and may be freely duplicated or distributed using any file



The single left rotation is one of the simpler balanci
rotations for AVL trees. The tree on the left is out o
balance because the rightmost subtree is deeper than t
other subtrees. The tree on the right shows how the si
left rotation corrects the imbalance.

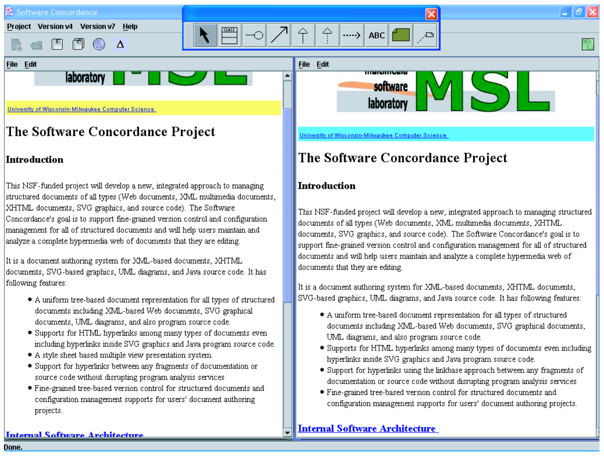
```
public void simpleLeft ( AVLNode parent )
{
   AVLNode grandParent = parent . getParent () ;
   AVLNode child = parent . getRight () ;
   AVLNode middleST = child . getLeft () ;
   boolean parentIsLeftChild = parent == grandParent . getLeft () ;
   parent . setRight ( middleST ) ;
   child . setLeft ( parent ) ;
   if ( parentIsLeftChild )
      grandParent . setLeft ( child ) ;
```

Figure 10: Intermixing object types

sharing protocol. At any version, developers can import and export a Web project's documents (stored in the Fluid persistence representation) including program source code, UML diagrams, SVG images, animations, and graphics from and to XML format. This feature is obviously important since it helps users work with tools outside the SC environment.

A user edits XML, HTML and Java programs in the same manner. The user interacts with the system using a menu bar, a tool bar, and contextual pop up menus. When a user selects a Web document from the project structure window, an appropriate editor is invoked for the document. To display a document, a default CSS-like style sheet is selected by the system for the document unless it has one. The presentation of the document is built based on the document tree and the style information. The user can choose to open a document with any appropriate style sheet. To edit a document, the user moves the mouse and selects any structural unit of the document that needs to be edited. Then, via the commands in the pop up menu, the user can choose to edit the content of that structural unit presented in the selected portion of the presentation, or to edit the documentation associated with that unit. The SC editor invokes the *node editor*, which is a simple ASCII text editor. The editor unparses the node and displays the resulting textual representation of the node to be edited. The user edits the text and returns to the document window. Depending on the type of the document that is being edited, the editor invokes either an XML, HTML parser or a Java parser to incrementally parse the modified text, and then creates new nodes and attaches them to the document tree. If there exists any errors in the modified text, error messages are displayed and the user can fix them.

The SC editing environment also allows a user to edit image or graphic documentation, and then to associate them with any structural unit in a Java program. This feature allows for intermixing different object types in the same document, which is very common in Web content. Figure 10 shows a Java program intermixing with graphics, texts, and audio clips. The user invokes an image and graphic editor. When the user finishes with his images, the SC



University of Wisconsin-Milwaukee Computer Science

The Software Concordance Project

Introduction

This NSF-funded project will develop a new, integrated approach to managing structured documents of all types (Web documents, XML multimedia documents, XHTML documents, SVG graphics, and source code). The Software Concordance's goal is to support fine-grained version control and configuration management for all of structured documents and will help users maintain and analyze a complete hypermedia web of documents that they are editing.

It is a document authoring system for XML-based documents, XHTML documents, SVG-based graphics, UML diagrams, and Java source code. It has following features:

- A uniform tree-based document representation for all types of structured documents including XML-based Web documents, SVG graphical documents, UML diagrams, and also program source code.
- Supports for HTML hyperlinks among many types of documents even including hyperlinks inside SVG graphics and Java program source code.
- A style sheet based multiple view presentation system.
- Support for hyperlinks between any fragments of documentation or source code without disrupting program analysis services
- Fine-grained tree-based version control for structured documents and configuration management supports for users' document authoring projects.

Internal Software Architecture

University of Wisconsin-Milwaukee Computer Science

The Software Concordance Project

Introduction

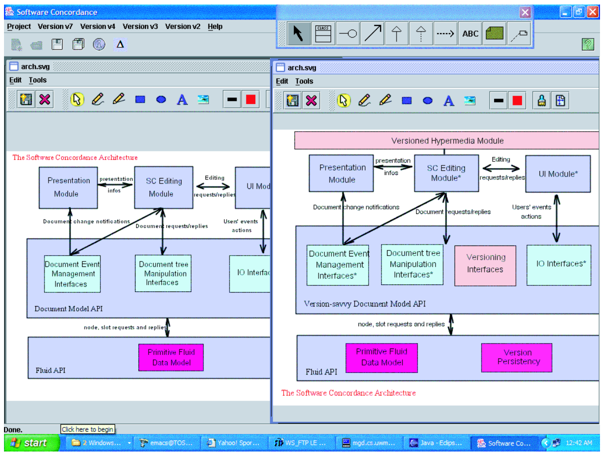
This NSF-funded project will develop a new, integrated approach to managing structured documents of all types (Web documents, XML multimedia documents, XHTML documents, SVG graphics, and source code). The Software Concordance's goal is to support fine-grained version control and configuration management for all of structured documents and will help users maintain and analyze a complete hypermedia web of documents that they are editing.

It is a document authoring system for XML-based documents, XHTML documents, SVG-based graphics, UML diagrams, and Java source code. It has following features:

- A uniform tree-based document representation for all types of structured documents including XML-based Web documents, SVG graphical documents, UML diagrams, and also program source code.
- Supports for HTML hyperlinks among many types of documents even including hyperlinks inside SVG graphics and Java program source code.
- Support for hyperlinks using the linkbase approach between any fragments of documentation or source code without disrupting program analysis services
- Fine-grained tree-based version control for structured documents and configuration management supports for users' document authoring projects.

Internal Software Architecture

Figure 11: HTML document editor



Software Concordance
Project Version v7 Version v4 Version v3 Version v2 Help
arch.svg
Edit Tools
The Software Concordance Architecture
Presentation Module
presentation infos
SC Editing Module
Editing requests/replies
UI Mod
Document change notifications
Document requests/replies
Users' events actions
Document Event Management Interfaces
Document tree Manipulation Interfaces
IO Interfa
Document Model API
node, slot requests and replies
Primitive Fluid Data Model
Fluid API
arch.svg
Edit Tools
Versioned Hypermedia Module
Presentation Module
presentation infos
SC Editing Module*
Editing requests/replies
UI Module*
Document change notifications
Document requests/replies
Users' events actions
Document Event Management Interfaces*
Document tree Manipulation Interfaces*
Versioning Interfaces
IO Interfaces*
Version-savvy Document Model API
node, slot requests and replies
Primitive Fluid Data Model
Version Persistency
Fluid API
The Software Concordance Architecture
Done.
Click here to begin
start
2 Windows...
emacs@TOS...
Yahoo! Spor...
WS_FTP LE ...
mgd.cs.uwm...
Java - Eclips...
Software Co...
12:42 AM

Figure 12: SVG graphic and animation editor

editor adds the resulting images into the document window. To associate audio documentation with a structural unit, the user invokes an audio selection dialog to choose an audio file. Image, audio, and graphic documentation file names are contained in special attributes of document nodes. In addition, the user can create, edit a hyperlink, and attach it to any structural unit in a Java program using steps similar to the ones needed to edit the documentation. The user can enter a URL or choose a file from a selection dialog. Similar to HTML documents, hyperlinks in programs are supported via the *href* attribute as described earlier. This approach to binding multimedia annotations and hyperlinks to source code does not interfere with program compilation process since it ignores the slots that support multimedia documentation and hyperlinks.

For an HTML document, the user is also able to preview the document's appearance as if it were displayed via a Web browser. Figure 11 displays the contents of an HTML document for two different versions. Similarly, Figure 12 shows two versions of the SC's architecture description document in SVG format. All graphic, image, and animation objects in SVG format are versioned in the same manner as XML documents. To edit a UML diagram, the user invokes the UML diagram editor, which is a specialized graphic editor for class diagrams, use case diagrams, sequence diagrams, and activity diagrams.

8. CONCLUSION

Systematic approaches to Web engineering are becoming increasingly necessary as Web applications grow and have longer lifetimes. The research presented here is based in the belief that managing the evolution of Web applications is a task that requires sophisticated configuration management tools. As recently as the year 2000, it was asserted that very few companies with mission-critical Web system were using SCM tools [11], and that others were developing their own CM tools and techniques because they were not fully aware of the state of the art in SCM. At the same time, researchers in the SCM area have paid little attention to bringing advanced technologies to Web engineering. Most existing SCM and version control tools that are used are focused on version control of individual files with limited supports for content change and configuration management of a Web project.

We have described a fine-grained version control and structured configuration management system (Molhado) and discussed how it is well-suited to managing the evolution of a Web application project. Molhado is a core part of the Software Concordance environment, which was modified by this research to support Web development. Its versioning scheme supports content change management that helps Web authors manage the evolution of their Web project at the logical level and at fine granularity. The configuration management services help Web developers to logically structure and manage a Web project and its components. Multiple logical organizations of a Web project can exist in the same version. The SC editor prototype provides a GUI-based environment for browsing and editing versions of a project and its components. A user can work on multiple versions simultaneously. SC is flexible and extensible to support many other types of Web content. An experimental study is being conducted to evaluate the performance, efficiency, and usability of the system.

9. REFERENCES

[1] ArticleBase. http://www.runningstart.com/.
[2] R. Barta and M. Schranz. JESSICA: an object-oriented hypermedia publishing processor. *Computer Networks and ISDN Systems*, 30:239–249, 1998.
[3] L. Bendix and F. Vitali. VTML for Fine-grained Change tracking in Editing Structured Documents. In *Proceedings of the Software Configuration Management Workshop*. Springer Verlag, 1999.
[4] J. Boyland, A. Greenhouse, and W. L. Scherlis. The Fluid IR: An internal representation for a software engineering environment. In preparation. For information see `http://www.fluid.cs.cmu.edu`.
[5] CCC/Harvest. http://www3.ca.com/.
[6] Cgilua 3.2. http://www.tecgraf.puc-rio.br/cgilua/.
[7] ClearQuest. www.rational.com/products/clearquest/index.jsp.
[8] ColdFusion. http://www.allaire.com/.
[9] R. Conradi and B. Westfechtel. Version models for software configuration management. *ACM Computing Surveys (CSUR)*, 30(2):232–282, 1998.
[10] S. Dart. Content Change Management: Problems for Web Systems. In *Proceedings of the Software Configuration Management Workshop, SCM-9*. Springer Verlag, 1999.
[11] S. Dart. *Configuration Management: the missing link in Web engineering*. Artech House, 2000.
[12] A. Diaz, T. Isakowitz, V. Maiora, and G. G. RMC: A tool to design WWW applications. *The World Wide Web*, 1995.

[13] Document Object Model. http://www.w3.org/dom/.

[14] DrawSWF. drawswf.sourceforge.net.

[15] Macromedia DreamWeaver. http://www.dreamweaver.com/.

[16] eChangeMan. http://www.serena.com/.

[17] P. Feiler. Configuration management models in commercial environments. Technical Report CMU/SEI-91-TR-7, Software Engineering Institute, 1991.

[18] J. C. Ferrans, D. W. Hurst, M. A. Sennett, B. M. Covnot, W. Ji, P. Kajka, and W. Ouyang. HyperWeb: a framework for hypermedia-based environments. In *Proceedings of the Symposium on Software Development Environments*, pages 1–10. ACM Press, 1992.

[19] Microsoft FrontPage. http://www.microsoft.com/.

[20] M. Gaedke and G. Graf. Development and Evolution of Web-applications Using the WebComposition Process Model. In *Proceedings of 2nd Web Engineering Workshop at the 9th International World Wide Web Conference (WWW9-WebE)*, 2000.

[21] A. Ginige and S. Murugesan. The Essence of Web Engineering. *IEEE Multimedia*, 8(2):22–25, April 2001.

[22] A. Ginige and S. Murugesan. Web Engineering: An Introduction. *IEEE Multimedia*, 8(2):14–18, April 2001.

[23] E. J. Glover, K. Tsioutsiouliklis, S. Lawrence, D. M. Pennock, and G. W. Flake. Using Web Structure for Classifying and Describing Web Pages. In *WWW11: 11th International World Wide Web Conference*, 2002.

[24] J. Griffiths, D. Millard, H. Davis, D. Michaelides, and M. Weal. Reconciling versioning and context in hypermedia structure servers. In *Proceedings of the 1st International Metainformatics Symposium*, 2002.

[25] S. Gupta, G. Kaiser, D. Neistadt, and P. Grimm. DOM-based Content Extraction of HTML Documents. In *WWW12: 12th International World Wide Web Conference*, 2003.

[26] D. L. Hicks, J. J. Leggett, P. J. Nurnberg, and J. L. Schnase. A hypermedia version control framework. *ACM Transactions on Information Systems (TOIS)*, 16(2):127–160, 1998.

[27] Dynabase content management. http://www.rbii.com/products/dynabase/.

[28] T. Isakowitz, E. Stohr, and P. Blasubramaninan. RMM: A Methodology for Structured Hypermedia Design. *Communications of the ACM*, 38(8):34–44, 1995.

[29] M. D. Jacyntho, D. Schwabe, and G. Rossi. A Software Architecture for Structuring Complex Web Applications. *Journal of Web Engineering*, 1(1):37–60, 2002.

[30] T. Kejser and K. Gronbak. The GAIA Framework: Version Support In Web Based Open Hypermedia. In *proceedings of IADIS International Conference on WWW/Internet*, 2003.

[31] D. Leblang. The CM challenge: Configuration management that works. *Configuration Management*, 2, 1994.

[32] D. Lowe. Web Engineering or Web Gardening. *WebNet Journal*, 1999.

[33] D. Lowe and J. Eklund. Client Needs and the Design Process in Web Projects. *Journal of Web Engineering*, 1(1):23–36, 2002.

[34] C. Maioli, S. Sola, and F. Vitali. Versioning for Distributed Hypertext Systems. In *Proceedings of ACM Conference on Hypertext and Hypermedia*, 1994.

[35] PVCS. http://www.merant.com/.

[36] L. Mignet, D. Barbosa, and P. Veltri. The XML Web: a First Study. In *WWW12: 12th International World Wide Web Conference*, 2003.

[37] S. Montero, P. Daz, and I. Aedo. A design toolkit for hypermedia applications. In *Proceedings of the International Conference on Web Engineering-ICWE*, 2003.

[38] T. Morse. CVS. *Linux Journal*, 1996(21es):3, 1996.

[39] S. Murugesan, Y. Deshpande, S. Hansen, and A. Ginige. Web Engineering: A new discipline for Web-Based System Development. In *Web Engineering: Managing Diversity and Complexity of Web Application Development (LNCS 2016)*. Springer Verlag, 2001.

[40] K. Østerbye. Structural and cognitive problems in providing version control for hypertext. In *Proceedings of the ACM Conference on Hypertext*, pages 33–42, 1992.

[41] Perforce. http://www.perforce.com/.

[42] R. Pettengill and G. Arango. Four lessons learned from managing WWW digital libraries. In *Proceedings of Conference on Theory and Practice of Digital Libraries*, 1995.

[43] J. Rumbaugh, I. Jacobson, and G. Booch. *The Unified Modeling Language Reference Manual*. Object Technology Series. Addison-Wesley, 1998.

[44] D. Schwabe and R. de Almeida Pontes. A Method-Based Web Application Development Environment. In *Proceedings of the Web Engineering Workshop, WWW8 Conference*, 1999.

[45] D. Schwabe and G. Rossi. An Object Oriented Approach to Web-based Application Design. *Theory and Practice of Object Systems*, 4(4), 1998.

[46] J. Simonson, D. Berleant, X. Zhang, M. Xie, and H. Vo. Version augmented URIs for reference permanence via an Apache module design. In *Proceedings of the WWW7 Conference, Computer Networks and ISDN Systems*, 1998.

[47] J. A. Slein, F. Vitali, E. J. Whitehead, Jr., and D. G. Durand. Requirements for distributed authoring and versioning on the World Wide Web. *StandardView*, 5(1):17–24, 1997.

[48] L. Soares, G. S. Filho, R. Rodrigues, and D. Muchaluat. Versioning support in HyperProp system. *Multimedia Tools and Applications*, 8(3):325–339, 1999.

[49] Microsoft Visual SoureSafe. http://msdn.microsoft.com/ssafe/prodinfo/overview.asp.

[50] E. Spertus and L. A. Stein. Squeal: A Structured Query Language for the Web. In *WWW9: 9th International World Wide Web Conference*, 2000.

[51] StarTeam. http://www.borland.com/starteam/.

[52] StoryServer. http://www.vignette.com/.

[53] Scalable vector graphics. http://www.w3c.org/Graphics/SVG.

[54] TeamSite. http://www.interwoven.com/.

[55] Thorn UML editor. http://thorn.sphereuslabs.com/.

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

[57] TrueChange. http://www.truesoft.com/.

[58] Viewing CVS Repositories. viewcvs.sourceforge.net/.

[59] WebIntegrity. http://www.mks.com/.

[60] WebSynergy. http://www.continuus.com/.

[61] E. J. Whitehead, Jr. *An Analysis of the Hypertext Versioning Domain*. PhD thesis, University of California – Irvine, 2000.

[62] E. J. Whitehead, Jr. WebDAV and DeltaV: collaborative authoring, versioning, and configuration management for the Web. In *Proceedings of the ACM conference on Hypertext and Hypermedia*, pages 259–260. ACM Press, 2001.

[63] W3C XML. http://www.w3c.org/XML.