Annals of Software Engineering 13, 231–247, 2002

XML-Based Hypertext Functionalities for Software Engineering

LUCA BOMPANI, PAOLO CIANCARINI and FABIO VITALI {bompani; cianca; vitali}@cs.unibo.it
Department of Computer Science, University of Bologna, Mura A. Zamboni, 7 I-40127 Bologna, Italy

Abstract. Hypertext functionalities represent a form of the distilled wisdom of the hypermedia community. Even if they were introduced and advocated already in the pre-Web era, most of these functionalities are absent in current Web browsers. However, such functionalities can be very useful in some specific applicative fields, like for instance browsing complex software engineering documents, using standard WWW components. We propose to exploit the advent of XML as a basic infrastructure for describing software engineering hypertexts. In fact, we describe XMLC, a prototype of an XML browser that, given its modular architecture and general scope, can be seen as the basis for implementing sophisticated hypertext functionalities for software engineering documentation to be maintained and browsed on the Web.

Keywords: XML, displets, hypertext functionalities, software engineering notations

1. Introduction

The community of researchers of hypertext functionalities was born in order to identify and study systems and functions suitable to create and manage hypertext, and to either verify them or introduce them in other communities such as document management systems, software engineering environments, and worldwide information systems like the Web [Rossi and Ziv (eds.) 1998].

Interestingly, even if it is a very huge hypertext, the World Wide Web has developed according to ways that were very peculiar and difficult to predict for most hypertext researchers. For instance, the WWW community valued the development of standards and protocols more than functionalities. This lead to the creation of some dozens of different languages and protocols that are necessary to master the task of creating satisfactory Web sites.

Such a richness of languages shows on one hand, that there exists the possibility of implementing a large number of interesting functionalities, and on the other hand, that unfortunately the WWW does not enforce or even facilitate them, so that their use depends on the will and awareness of the authors of Web pages and sites. Furthermore, this richness of possibilities is coming to the detriment of simplicity, which was once the real advantage of the World Wide Web over other systems such as Gopher or FTP.

Yet, the XML family is a considerable advancement over previous languages and standards. The possibility offered by XML [Bray *et al.* 1998] to freely define a syntax

(i.e., a Document Type Definition, or DTD) tailored for one's document classes, and to use standard XML tools to create, verify and exchange data is a real bonus.

In our opinion the strength of XML lies beyond its capabilities suitable to define community-specific DTDs. For instance, it is becoming convenient to use it even for application-only data, that is, for objects that are not naturally meant to be displayed to a human user.

One of the long-term goals of our research group at the University of Bologna consists of creating an environment that, while relying on several existing Web languages and protocols, can provide fundamental hypertext functionalities in a streamlined and easy way. In this paper we will concentrate on browsing and displaying hypertext data.

Our approach is particularly useful to make software engineering environments *Web-aware*: most documents of current software processes tend to be composed of several different chunks, some of text, some of formulas in special notations, and some of structured graphical diagrams. Currently it is very difficult to turn these documents into pages that can be made available through a Web browser, since each formula and each diagram need to be converted into a passive image.

In this paper we discuss the design of XMLC, a compiler for XML documents that is the kernel of a very general architecture for providing sophisticated functionalities to documents created in the XML format. While our overall design goals are to create a complete authoring environment for sophisticated hypermedia, in this paper we focus on sophisticated browsing of XML data. Indeed, the architecture described here can be fruitfully used for more than visualization, for it is an extremely general way to associate behaviors to XML elements, and thus to produce active documents that perform computations, enact goals, produce results. We call these documents *declaratively active documents* (or DADs) because of this characteristic.

This paper is structured as follows: in the next section we discuss some of the most important hypertext functionalities on the Web. Then we classify some hypertext functionalities that are natural to single out in normal software engineering documents. In the following section we discuss the current architecture of XMLC, and provide examples of some of the displet classes we have created. Of particular importance in our view are the packages for displaying notations relevant to software engineering, which have constituted for several reasons our main target for the implementation of displets.

2. Hypertext functionalities on the Web

In [Bieber *et al.* 1997] a list of nine fundamental (according to the authors' opinions) hypermedia functionalities were proposed and discussed, with the understanding that few of them, if any at all, were either available on the World Wide Web or exploited to their full potential:

- Typed nodes and links.
- Link attributes and structure-based queries.
- Transclusions, warm and hot links.

- Annotations and public vs. private links.
- Computed personalized links.
- External link databases and link update mechanisms.
- Global and local overviews.
- Trails and guided tours.
- Backtracking and history-based navigation.

These items were selected from a longer list of 25 items assembled at the 2nd HTF workshop in conjunction with the Hypertext 96 conference [Ashman *et al.* 1996].

Most of these functionalities could be easily implemented with current WWW technologies. Server-side CGI, servlets and DBMS applications, as well as client-side plug-ins, Java and Javascript programs allow now a degree of freedom in customizing the WWW unprecedented in any other hypermedia system (even those that did implement some of these functionalities). Both the research and the commercial communities have in fact already explored some of these functionalities in the last few years. Yet, few of them have really caught on with the larger WWW community, or even found a small visibility stand-point through the available commercial applications.

We suspect indeed that no Java applet, CGI application or other custom concoction can possibly produce any relevant change in the way the WWW is used. The reason for this is that these would all be added functionalities to the core sets in servers and browsers, and, unfortunately, the WWW is neither the set of server functionalities, nor the set of browser functionalities. The fact that the WWW is not a system, or a set of interdependent systems, but a set of protocols and languages, is obviously not yet sufficiently understood. No single system can provide added value to the WWW as a whole. Almost no organization (in many cases not even Microsoft) can introduce a new functionality in its products and find out that the WWW as a whole catches on. The WWW must not be improved in the systems, but in the way it actually works: by changing the underlying languages and protocols (namely HTML, HTTP, CGI, etc.).

We can group the above-mentioned functionalities in two larger families: those that add to the active participation of the users in the production of information, and those that add to the exploration of the available information. On the one hand, annotations, private links and computed personalized links (that require external link bases and link update mechanisms to work on a large scale) allow for the active participation of readers to the nodes they read. On the other hand, overviews, trails, guided tours and sophisticated backtracking patterns (that require richer types and attributes for nodes and links) enhance the navigation and the access to the information of the hyperbase. Finally transclusions and links of various temperature provide both a richer expressive means for authors, and a richer exploration means for readers.

Both families share the same problem: they are not functionalities that can be experienced by the single user, i.e., that one enlightened user can adopt for his/her own purposes and be enriched by using them; they are functionalities that have to be shared by a large community in order for them to fully provide their benefits. There is little point

in using an external link database, if we cannot share our links with our colleagues; there is little point in annotating or transcluding, if we cannot publish our notes and transclusions; there is little point in being able to create overviews and guided tours on some collection of documents, if we cannot publish them for our readers. Thus these functionalities must be dictated through the standards and protocols that make up the Web, rather than through any specific application.

More recently, in [Vitali and Bieber 1999] four hypermedia functionalities were further identified:

- Editable browsers.
- Storing document content and link anchors separately.
- External linkbases.
- Displaying link spans, node and link attributes.

In all these cases, actual WWW protocols were cited that could provide the necessary expressive power to implement these functionalities: WebDAV [Goland *et al.* 1999] provides clients with remote writing power, thus make editable browsers a real possibility. XPointer [DeRose and Daniel Jr. 2001] and XLink [DeRose *et al.* 2001] allow external links, thus making it possible to separate content and link, and to put links into external linkbases. RDF [Brickley and Guha 1999] allows arbitrary meta-information to be added to any Web document, and to be used for classification, indexing, and searches.

A shareable long term goal is to identify a single, simple and streamlined architecture to provide all these functionalities using WWW protocols and hiding the complexities behind the protocols used. With XMLC, which will be described in the next section, we are providing a single, easy to use and easy to expand architecture for browsing XML documents. We consider it a first step in that direction.

3. Hypertext functionalities in a software development project

A *software development process* is the description of both the activities and the documents to be produced during the development of a software product. In short, a software process is a method to produce a software product; it prescribes in details all the documents to be produced and all techniques and tools to be used in all development phases starting from the exploration of the concept of a new product and ending when the product is finally retired from operation.

In the last years, software engineering environments have evolved mainly in the explicit support they offer to specific software process models. This means that most research efforts have focused on how a software process is described and how its activities are controlled and enacted by a *process engine*, namely a process-centered programming environment that divides the whole software engineering lifecycle in several subsequent and/or parallel phases, each with its own characteristic input and output documents.

The set of documents related to a software development process are many, of different forms, formats, and specificity. Documents belonging to different phases of the

software process will sometimes have the same topic, but with a different level of detail and granularity. Documents belonging to the same phase will be differentiated in purpose, structure and content. Single documents belonging to one phase may contain parts of differing structure, notation, and purpose: for instance, the same topic may be handled with a free-text description, a formal specification using an appropriate mathematical notation, a graphic depiction of its parts, etc.

A key issue that has become pervasive and obvious in recent years is the support for hypertext functionalities in the data format. In its simplest form, hypertext is the provision of connections and relationships between documents and text chunks, allowing readers to move from a document to another one (*navigation*) following a different order than the one imposed by the linear sequences of the documents themselves. In a complex situation such as the one enacted in the software development process, the relationships could be classified in scope as follows:

- Inter-phase relationships, or the relationships existing among documents belonging to different phases of the software process.
- Intra-phase relationships, or the relationships existing among different documents of the same process phase.
- Inter-part relationships, or the relationships existing, within a single document, between different parts and possibly different notations (free-text, structured text, mathematical notations, graphical depictions).
- Intra-part relationships, or the relationships existing within a single part or notation of a document, between the atomic entities that compose it (for instance, between different elements of the same schematics).

A more comprehensive and general definition of hypertext is *relationship management for data-based applications*. An application in our sense is not simply a program, but a structured set of procedures that are concerned with a collection of data elements and interest computer programs, people, processes, plus whatever storage, retrieval, transmission and processing engines are used for their management. The same application can be used with different data, and the same data can undergo different applications. So, according to our definition, hypermedia is all that is concerned with structuring and giving access to the parts composing an application through their interrelationships.

These relationships do not simply have to be among the data elements, or be composed of specific, ad hoc references between two specific objects. We take a broader view of the idea, deriving from [Bieber *et al.* 1997] a list of classes of relationships.

3.1. Schema relationships

Schema relationships are the connections created by the structuring of the data elements in separate objects where the relation is apparent in the structure itself. The schema relationships deal with the structures of the entities of the software process: the modules of software packages, the function points of a complex procedure, the elements of a

complex data type, the methods of an object, etc. Modular design practices (top-down, bottom-up, structured, object oriented, etc.) imply that some entities are defined once and recur in several different places in the same document or across documents.

3.2. Ontological relationships

Ontological relationships are the connections linking data elements, programs, people, process steps or underlying working environments with the parameters and descriptive information that accompany and define them. Ontological relationships provide generic information about the individual entities of a document: the meaning of an acronym or of a specialized term, the numerical values of a quantitative design constraint, the global identifier of a document section, issue, requirement or entity, etc. Such information chunks are not necessarily contained in any other document and can usually be deduced by the definition of the element, by its specification, or by an apposite data dictionary provided for this purpose.

3.3. Occurrence relationships

Occurrence relationships are the connections between all appearances and uses of a data element, program, person, etc., in the application. Sometimes the same entity, requirement, or function appears in several documents, for different purposes and in different detail. Occurrence relationships exist between an entity and all the places in the document where the information is presented, discussed or detailed. Indexes specify all the occurrence of a given term. Specialized indexes may create a list of all the occurrences of only those terms that are deemed interesting. Tables of content, on the other hand, provide a general structure of the definition of terms. By combining these two services, one can easily provide all the occurrence relationships in documents.

3.4. Process relationships

Process relationships are the connections between a data element, program, person, etc. and the application's processes that can or do interact with it. We say that there is a process relationship between an entity and all the tasks in the software processes that deal with the entity. For instance, there is a process relationship between drafts of the same document as it progresses through all the stages of creation, verification and modification. Or, a process relationship exists connecting the test series and their output, because each output is the result of a process on a specific test suite.

3.5. Structural relationships

Structural relationships are the connections that embody a structure of constraints and references. Some information chunks may be seen as a different view of some data items that may or may not be shown autonomously. For instance, if a read-only method of a

class is a simple computation based on other read/write elements, a structural relationship exists between the method and the formula. More abstractly, the choice of a given software process model will impose the creation of a series of given documents, whose structure and layout are often given in advance. Thus there exists a structural relationship between each structure and the part of the specification of the software model that requires it.

3.6. Dynamic relationships

Dynamic relationships are connections that are not intrinsic to the application model or in any other way known in advance, but that become apparent during the life of the application. These relationships may derive from applying some logic and computation to the state of the document, or may derive from external events that happened since the document was written. A dynamic relationship thus is a data-mining discovery about the usage, structure, occurrence, etc. of entities and documents. Generally, these relationships present themselves during the lifecycle of a document, as opposed to the time of its creation. That is to say, they cannot be discovered in advance, during the design of a document, but are the result of computations on the instances of documents. In the case of the software process, the most important computations that can be performed on documents are validation and verification of their correctness, completeness and consistency. Whenever a problem is found out, either automatically or by a human reader, a relationship is created on the relevant documents. Besides actual problems, dynamic relationships automatically discovered by appropriate engines may improve the consistency in terms, interfaces, libraries, etc., which may be extremely important for large and complex software projects.

3.7. Ad hoc relationships

Ad hoc relationships are all the relationships whose existence cannot be determined by a rule, but are created because of an ad hoc decision (usually by a human, but not necessarily). These are the well-known hypertext links in the stricter sense. For instance, this includes all connections between structured elements of the documents and non structured chunks (such as annotations, comments, discussions, etc.). Furthermore, of course, all links and references to documents not produced within the software process are ad hoc. Furthermore, we include in this category all connections that, for several ad hoc reasons, cannot be included in the other categories (such as links to unclassified bug reports, unimplemented constraints, deliberate violations of requirements and specifications, etc.).

To every type of relationship corresponds a type of link. With the exception of ad hoc links, which have to be created one by one by skilled people, and dynamic links, which should be created during the lifetime of the document base by ad hoc analysis and data mining applications, all other types of relationships can and should be made available to users without specific human intervention. These two classifications, in

Table 1
Relationships among software process documents.

	Inter-phase relationships	Intra-phase relationships	Inter-part relationships	Intra-part relationships
Schema relationships	Recurring definitions	Use definition	Use definition	Not appropriate
Ontological relationships	Methodology explanation	Terms and objects dictionaries	Terms and objects dictionaries	Object properties
Occurrence relationships	Table of contents and indexes	Table of contents and indexes	Table of contents and indexes	Table of contents and indexes
Process relationships	Not appropriate	Test sets and results	Process descriptions	Process descriptions
Structural relationships	Stub generation	Not appropriate	Not appropriate	Computed class members
Dynamic relationships	Inconsistency reports	Inconsistency reports	Inconsistency reports	Inconsistency reports
Ad hoc relationships	Comments and references to external literature	Comments and references to external literature	Comments and references to external literature	Comments and references to external literature

scope and types, provide us with the grid in table 1, which we filled with the types of relationships that are relevant for the software process.

4. The architecture of XMLC

XMLC (XML Compiler) is an architecture for rendering displets. XMLC relies upon technologies and languages such as XML, XSL and DOM.

The main purpose of XMLC is to read an XML document and to produce a displayable tree of Java objects. This happens in a few steps: first, the XML document is read and transformed by a normal XML parser into an internal tree representation based on DOM. Then one or more layers of XSL stylesheets are applied to the DOM tree through the use of an XSLT processor. This creates a final DOM tree that needs to have an important property: for every element type in the tree there must be an available Java class (a displet) that can be activated. XMLC will finally instantiate all the required displets, creating a tree of runnable objects.

Each element in the DOM tree is transformed into a displet according to the following rules:

- The element's name determines the Java class to be loaded.
- The element's attributes determine the value of the settable properties of the instance of the class.
- The element's content (both sub-elements and text content) is added to the tree as children of the class instance.

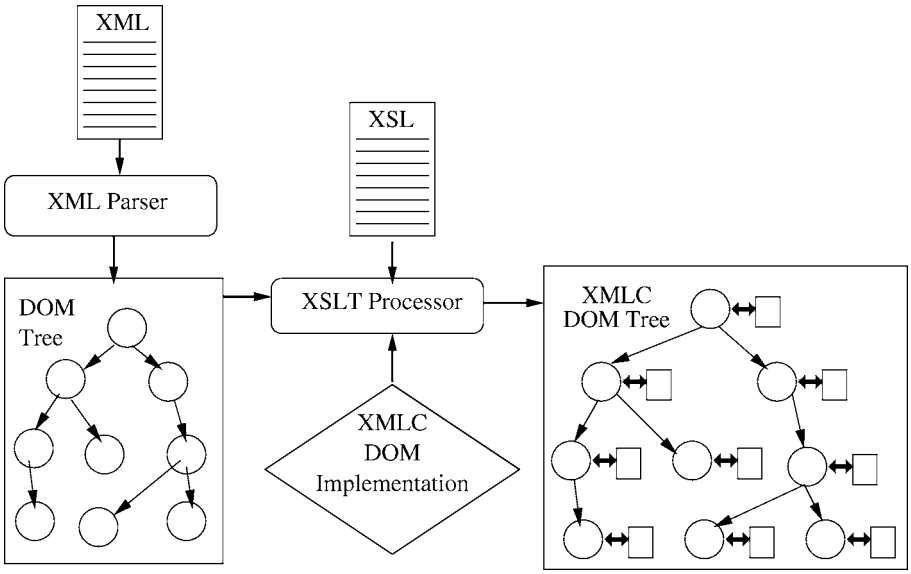

Figure 1. The architecture of the XMLC engine.

The current implementation of the XMLC architecture is in Java; a displet can be any sort of Java classes, but deriving them from the widget defined by the AWT and Swing Java library, it is easy to create sophisticated and interoperable displets. In particular using the Containers and Components base class they can be easily organized in hierarchies, which nicely fit with the hierarchical nature of DOM trees and XML documents.

Currently, our main use of XMLC is wrapped inside an applet within an HTML document, as shown in figure 1. Parameters of the applet are the XML documents to be displayed and the XSL stylesheets to be applied to it. This allows us to display XML documents within well-known Internet browsers.

Furthermore, since XML elements are transformed into Java objects, complex behaviors can be easily added during the lifetime of the visualization, providing support for hypertext jumps, animations, interactions with the reader, and in general all the computational capabilities of the Java language.

5. Active documents for software engineering

In our research group we have adopted a displet-based approach to building tools for software engineering notations, like for instance XML, Z [Brien and Nicholls 1992] and Petri Nets. We have developed in the last year a number of specialized browsers/editors for these and other well-known notations, which will be described in the following subsections.

Given a formal notation (e.g., Petri Nets diagrams), we have looked for or defined a DTD in order to capture its abstract syntax. Starting from the DTD, we have defined one

or more XSL stylesheets that can be used to manipulate the XML documents. There are at least two purposes that can be enabled by this transformation: we can either display the document or provide some kind of computation on them, such as performing static analysis on them. For instance, with a Petri Net diagram, a possible static analysis consists of looking for loops.

The final step consists usually of enabling the editing and interactive display of the notation inside a Java-enabled browser developing a library of specific displets. We have developed displets for Petri Nets, Z, Statecharts, Data Flow diagrams, Entity–Relationship diagrams, Workflow diagrams, and most UML diagrams. Interactive display is possible when some behavioral semantics is associated to the notations. For instance, the Petri Nets displets can play the token game typical of this notation.

5.1. UML specifications

A key issue is how to define a DTD for a complex software engineering notation. For instance, if an organization uses the UML family of notations and related development process and tools, it is now available XMI (XML Metadata Interchange, by IBM and others), an XML-based metamodel. All UML documents written according to XMI can be displayed by XML-aware browsers and manipulated by XML-based tools to check for some semantics constraints, like consistency. We are applying our approach to XMI as well. A displet has been developed in our group to provide visualization of XMI. An editor called Elmuth (reverse acronym for HyperTextual UML Environment) has been developed. Elmuth is able to create hypertextual and active visualizations of UML documents. Figure 2 shows an instance of MS Internet Explorer including an active document describing (part of) the UML metamodel.

Hypertext multidirectional links among diagrams are managed using our implementation of XLink. The browser includes here four areas: the uppermost left area shows an HTML index useful to navigate the document, the uppermost right area shows a class diagram, the lower right area is a data dictionary, the lower left area shows some code automatically generated from the class diagram.

5.2. The Z notation

A complete support for the Z notation has been implemented. The DTD we use is based on the ZIF Interchange Format [Ciancarini *et al.* 1999], although, through the use of different XSL stylesheets, other syntaxes can be used as well.

The support for Z elements is given by a single displet class, zElement, for all the box types present in Z specifications (e.g., schema, axioms, etc.), and a special downloadable font for all the mathematical glyphs specific of the Z language (e.g., function, subset, the set of integers, etc.).

All other elements of the Z language are mapped onto plain HTML elements such as P, DIV and SPAN. An additional layer of XSL will then transform them into Paragraph and Word objects as needed. In figure 3 we show a small fragment of a Z specification (expressed in ZIF) and in figure 4 its rendering.

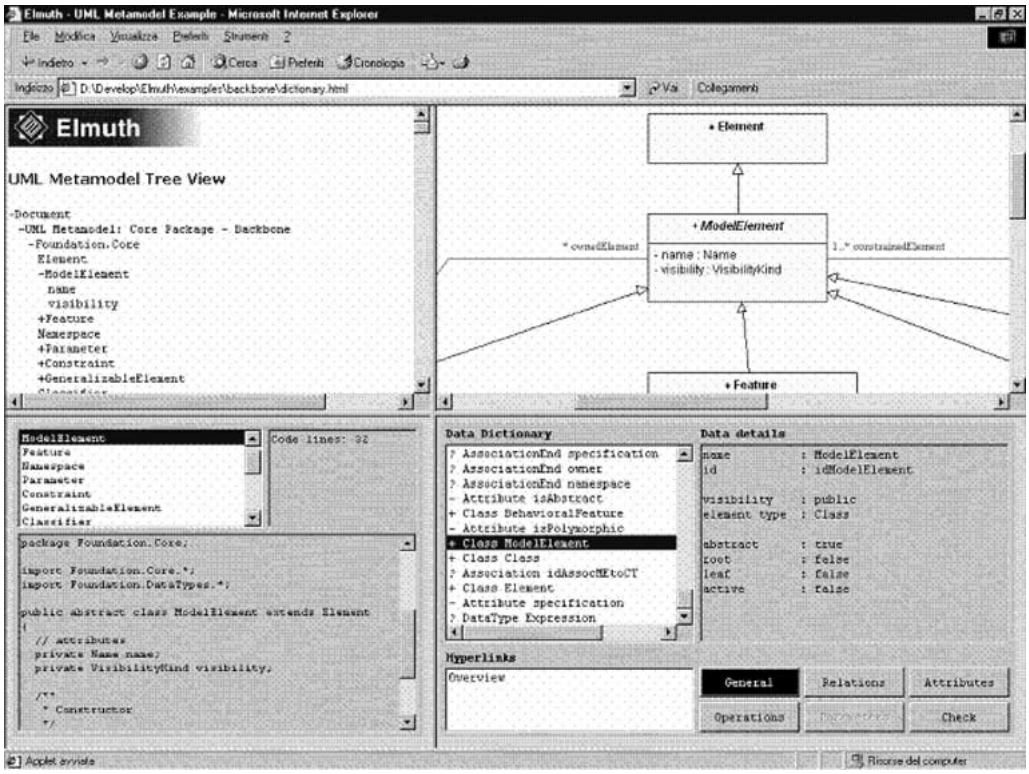

Figure 2. The representation of a UML diagram.

5.3. Finite State Machines

A very simple notation for finite state machines has been implemented.

Lacking any agreed-upon DTD we have created our own, which composed of just three elements: StateMate (the general container), State (representing a State in the Finite State Machine, shown as a rounded-rect box in the display) and Arc (representing a transition in the Finite State Machine, and shown as an arrow in the display). Each state has a position and a label, while the arcs have a label that start from and arrive to the center of the state box. The labels are the content of the State and Arc elements, and can be of any kind (that is, one can use any other displet for them, including HTML elements or other notations as needed).

Each state has an identifier, which is used by the arcs to identify their origin and destination. States can be initial or final. The author must specify the position of the states, while labels are automatically drawn in the correct position. In figure 6 we provide an example of a simple StateMate fragment shown in figure 5. StateMate schemas are an example of active displets, since both states and arcs are active. The active state is highlighted, and by clicking either on a transition or on a destination state, it is possible to traverse the available transitions and execute the finite state machine. Non-reachable states and transitions cannot be activated.

```
<schemadef style="vert" purpose="state">
  PhoneDB
  <decpart>
    <declaration>
      _known: &pset; NAME
    </declaration>
    <declaration>
      phone: NAME &fpfun; PHONE
    </declaration>
  </decpart>
  <formals> K,L,Z </formals>
  <axpart>
    <predicate>
      known = &dom; phone
    </predicate>
  </axpart>
</schemadef>
```

Figure 3. An XML fragment with a Z specification.

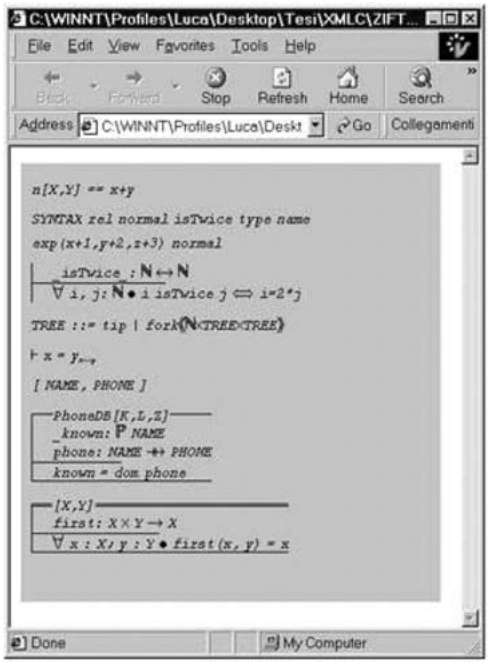

Figure 4. The visualization of the Z specification.

```
<StateMate>
 LEVEL_MANAGER_CONTROL
 <Arc from="state2" to="state1">
  LM_ACTIVE
 </Arc>
 <Arc from="state1" to="state3">
  LM_MORE
 </Arc>
 <Arc from="state3" to="state2">
  UPDATE
 </Arc>
 <State id="state1" start="true"
  origin="0,50">
  WAIT
 </State>
 <State id="state2" origin="250,50">
  NEW_VAL
 </State>
 <State id="state3" end="true"
  origin="100,200">
  PUMP_ACTIVE
 </State>
</StateMate>
```

Figure 5. An XML fregment with a finite state machine.

5.4. Hypertext links

W3C is proposing two languages to express hypertext links in XML. XPointer [DeRose and Daniel Jr. 2001] provides a way to express sub-resource addresses within XML documents and other resources, and XLink [DeRose *et al.* 2001] defines a syntax for hypertextual links between XML documents.

XPointers can specify locations within XML documents by collecting progressively detailed location specifiers. This makes it possible to specify an arbitrarily small location without marking it with a tag as in HTML.

XLinks extend HTML links by introducing several new features:

- Links can refer to multiple end-points.
- Links can be multi-directional.
- Links can be stored externally to the resources they link.

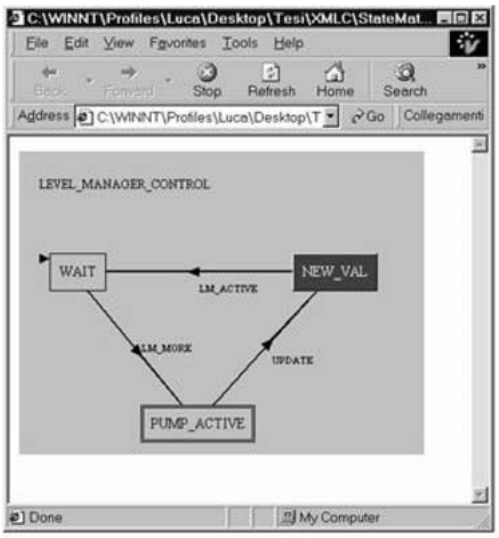

Figure 6. The visualization of the finite state machine.

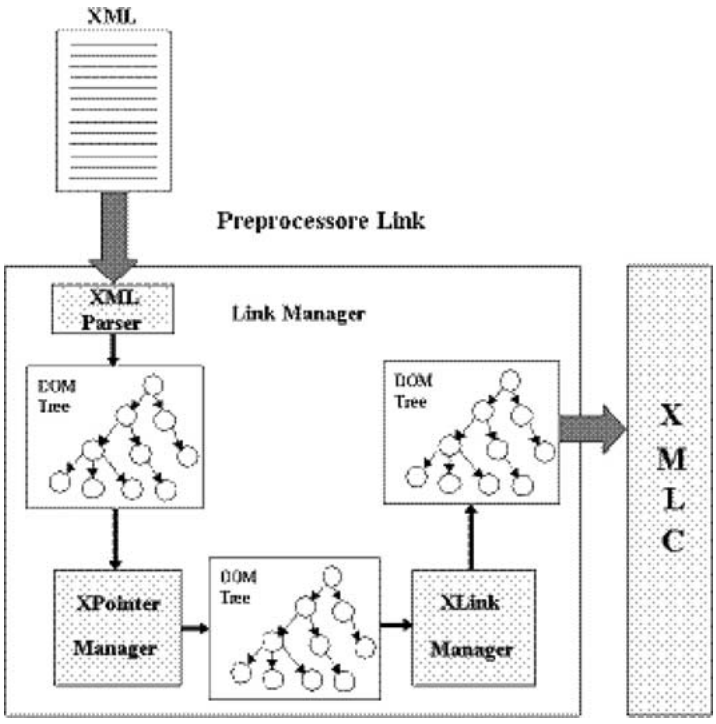

Figure 7. The XLink-enabled architecture of XMLC.

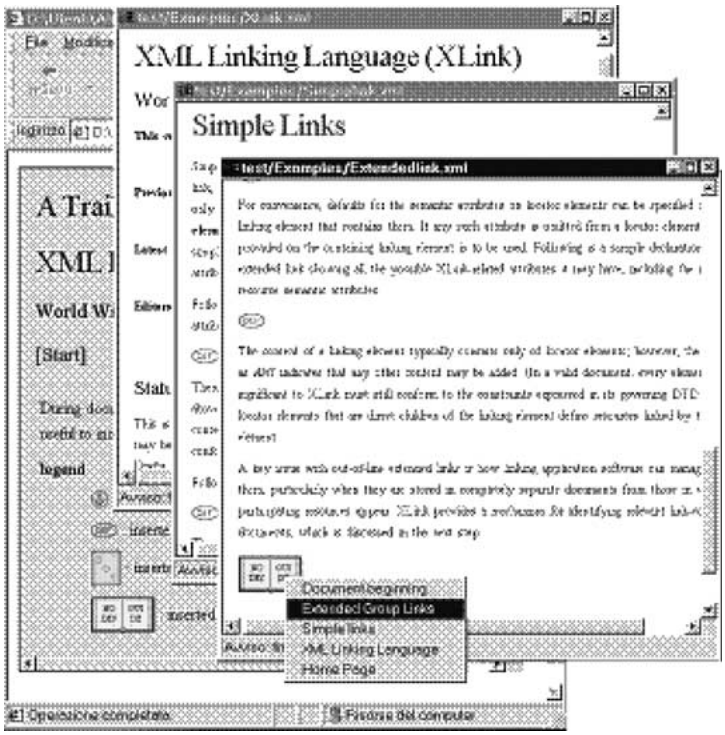

Figure 8. A simple hyperlinked document group.

- Links can be activated in a variety of ways (they may open a new window, substitute the current content, or expand within the current content, etc.).
- Links can create groups of related documents to be loaded together.

We have provided a complete implementation of XLink based on displets for our XMLC architecture. This has added a few steps to the sequence of transformations of the XMLC application, as shown in figure 7.

After parsing the XML document, all link elements are identified and added to a list. Then, an identifier is added to all the addressable elements of the document, since after the application of the XSL stylesheets the structure of the document can become arbitrarily different from the original one, and it is necessary to provide a way to identify the elements that can be located through XPointers. The document then is subjected to the usual XSL transformations. Before displaying, though, additional wrapper classes are added around the document elements that are starting points of links, to provide the most appropriate jumping functionality. When the user clicks on one such element, the class reacts, consults the list of destinations, and activates the jump.

The implemented management of document groups is quite sophisticated and takes into consideration whether the destination document will replace the current one, it will be created in a new window, or it will integrate with the current document. Figure 8 shows a sample hyperlinked document group.

6. Conclusions

Hypertext functionalities will be slow in implementation, and even slower in acceptance. It is just too difficult to take care of them by non-professionals.

The kind of ideas and functionalities presented here and in the literature on hypermedia functionalities present important characteristics anyway, that we presume will become more and more important as the public gets acquainted with the possibilities of the new medium. Yet, in order to provide easily sophisticated functionalities as the ones mentioned, the current architecture of the clients and the servers needs to be rethought. In particular, fewer and more powerful protocols and standards need to be used.

The XML family is an important step in that direction. XML and its cohort can actually let users and authors express their data and wishes in a sophisticated, customizable and expandable way. But a new software architecture needs to be implemented to take advantage of the generality of these languages.

XMLC is a customizable and expandable architecture for displaying XML documents. Being expandable, it has been easy to add support for several sophisticated hypertext functionalities, such as the ones allowed by XLink and XPointer. Work is under way to add more of them to future implementations. XMLC is a working prototype, and can be examined, downloaded and used. We gladly point the interested reader to the URL: `http://www.cs.unibo.it/projects/displets/`.

Acknowledgements

This paper has been partially sponsored by an Italian MURST 40% project contract "SALADIN", and by a grant from Microsoft Research Europe. We would like to acknowledge here the contribution of all the people that have worked on this architecture: Michael Bieber, Chao-Min Chiu, Cecilia Mascolo, Stefano Pancaldi, Alfredo Rizzi, Alessandro Rocca, Alessandro Ronchi, Silvia Villa, and all our students of the Software Engineering classes at the University of Bologna.

References

Ashman, H., V. Balasubramanian, M. Bieber, and H. Oinas-Kukkonen, Eds. (1996), *Proceedings of the 2nd International Workshop on Incorporating Hypertext Functionality into Software Systems* (*HTFII*)*, Hypertext 96 Conference, Washington*, `http://www.cs.nott.ac.uk/hla/HTF/HTFII/Proceedings.html`.

Bieber, M., F. Vitali, H. Ashman, V. Balasubramanian, and H. Oinas-Kukkonen (1997), "Fourth Generation Hypertext: Some Missing Links for the World Wide Web," *International Journal of Human-Computer Studies 47*, 31–65.

Bray, T., J. Paoli, and C.M. Sperberg-McQueen (1998), "Extensible Markup Language, (XML) 1.0," 10 February 1998, `http://www.w3.org/TR/REC-xml`.

Brickley, D. and R. Guha (1999), "Resource Description Framework (RDF) Schema Specification," 3 March 1999, `http://www.w3.org/TR/REC-rdf-schema`.

Brien, S. and J. Nicholls (1992), "Z Base Standard, Programming," Programming Research Group, Oxford.

Ciancarini, P., F. Vitali, and C. Mascolo (1999), "Managing Complex Documents over the WWW: A Case Study for XML," *IEEE Transactions on Knowledge and Data Engineering 11*, 4, 629–638.

DeRose, S. and R. Daniel Jr. (2001), "XML Pointer Language (XPointer), W3C Proposed Recommendation," 27 June 2001, `http://www.w3.org/TR/xptr`.

DeRose, S., E. Maler, and D. Orchard (2001), "XML Linking Language (XLink), W3C Recommendation," 27 June 2001, `http://www.w3.org/TR/xlink`.

Goland, Y., E. Whitehead, S.C.A. Faizi, and D. Jensen (1999), "HTTP Extensions for Distributed Authoring – WEBDAV, IETF RFC 2518," February 1999, `http://www.ietf.org/rfc/rfc2518.txt`.

Rossi, G. and H. Ziv, Eds. (1998), *Proceedings of the Fifth International Workshop on Engineering Hypertext Functionality into Future Information Systems* (*HTF5*)*, ICSE'98 Conference*, Kyoto, `http://www.ics.uci.edu/pub/kanderso/htf5/papers`.

Vitali, F. and M. Bieber (1999), "Hypermedia on the Web: What Will It Take?" *ACM Computing Survey*, in print.