

# XML-based hypertext functionalities for Software Engineering

Luca Bompani, Paolo Ciancarini, Fabio Vitali

Dept. of Computer Science, University of Bologna

Mura A. Zamboni, 7 I-40127 Bologna (Italy)

e-mail: [bompani | cianca | vitali]@cs.unibo.it

## ABSTRACT

*Hypertext functionalities represent a form of the distilled wisdom of the hypermedia community. Given the peculiar nature of the World Wide Web, it is very difficult to successfully propose functionalities that become widely accepted. XMLC is a prototype of an XML browser that, given its modular architecture and general scope, can be proposed as the basis for implementing sophisticated hypertext functionalities on the Web.*

**KEYWORDS:** XML, displets, hypertext functionalities

## INTRODUCTION

The community of hypertext functionalities was born in order to identify and list the functionalities intrinsic to the idea of hypertext, and to either verify them or introduce them in other communities such as document management systems, the World Wide Web, software engineering [14], etc. In particular, the World Wide Web has developed according to ways that were very peculiar and difficult to predict. For instance, the WWW community valued the development of standards and protocols more than functionalities. This led 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 exist 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. In the near future, and just to mention non-proprietary efforts, any Web author will have to deal with at least ten different and non-trivial languages or protocols, namely HTML, HTTP, CSS, ECMAScript (or any of its proprietary dialects, such as Javascript or Jscript), XML, XSL, XPath, XLink, XML-Schema, RDF, and WebDAV, plus many others that at the moment are starting to catch on. So, while there really exist the possibility, in the languages, to provide sophisticated hypertext functionalities, we have to wait for Web applications to actually provide them in a usable way.

Yet, the XML family is a considerable advancement over previous languages and standards. The possibility given by XML [4] to 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. But in our opinion the strength of XML lies beyond the capabilities 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.

Additionally, XSL [10] provides much to XML in terms of reach and flexibility. XSL includes a mapping language [9] that can be used to transform an XML document into another one. Currently its most important use is to transform an XML document into a format that can be displayed by a browser: thus for instance Microsoft Internet Explorer 5 can accept XML documents of any DTD and use XSL to transform them into an HTML document that can then be properly displayed on a computer screen.

Our long-term purpose is to create 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 "WWW-aware": the documents of the software process 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 past papers we discussed *displets* ([7] and [8]), our proposal to provide flexible support for special rendering needs that authors may have. Displets are software modules (currently they are Java classes) that are associated to each element in an XML document and that provide some rendering behavior for that element. Support for the most common element types is provided (for instance, text elements and paragraphs), but it is possible at any time to add new modules enabling specialized rendering semantics for specific needs. Displets provide a way around the afore-mentioned limitations, allowing the semantically-rich description of formulas and diagrams to be expressed in XML and to be displayed in the browser in their full graphical rendering. Additionally, the possibility of activating in some ways the diagrams, verifying their correctness and providing multiple views seem important and easily possible in a general way with the dispset approach.

In this paper we discuss XMLC, a recent implementation of our dispset architecture (also briefly introduced in [3]). XMLC can be considered as a very general architecture for providing very sophisticated functionalities to documents created in the XML format. While the overall design goals are to create a complete authoring environment for sophisticated hypermedia, in the current paper we concentrate 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. Next we provide a few background information about XML and related noteworthy standards. In the following section we discuss the current architecture of XMLC, and provide examples of some of the dispset 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 dispsets.

## 2 HYPERTEXT FUNCTIONALITIES ON THE WEB

In [2] a list of 9 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 [1].

At the moment, probably, all of these functionalities could be easily implemented on the WWW. 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). The research and 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.

It is indeed our opinion 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 in 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 protocol and languages, is obvious yet not sufficiently understood. No single system can provide added value to the WWW as a whole. Almost no organization (in many cases not even Microsoft or Netscape) 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 (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 can't share our links with our colleagues; there is little point in annotating or transcluding, if we can't 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 can't 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 [16] 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 [12] provides clients with remote writing power, thus making editable browsers a real possibility. XPointer [11] and XLink [13] allow external links, thus making it possible to separate content and link, and to put links into external linkbases. RDF [5] 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.

## **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 focussed 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 [BV97] a list of classes of relationships.

*Schema relationships*: 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.

*Ontological relationships*: 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.

*Occurrence relationships*: 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 exists 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.

*Process relationships*: 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 progress 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.

*Structural relationships*: 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 exist 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 is often given in advance. Thus there exist a structural relationship between each structure and the part of the specification of the software model that requires it.

*Dynamic relationships*: 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 is 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.

*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.).

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

**Table 1.** Relationships among software process documents

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

## XML AND RELATED STANDARDS

Formed in 1994 by the special interest group on SGML of the World Wide Web Committee, the XML working group set to work to provide an SGML-based generalized markup language that could provide most of the functionalities of SGML, while retaining the same simplicity of HTML for the casual author and the developer of tools. The working group divided the effort in several independent but related languages that cooperate in order to provide a complete and sophisticated markup environment. For our purposes, the most important proposals of the XML family are the XML generalized markup language itself, the XSL stylesheet language, and the DOM document object model, which will be discussed singularly in the following sections.

XML 1.0 (Extensible Markup Language, [4]) is the first and most stable proposal of the XML family. It is a proper subset of SGML, rather than an application of it (like HTML). XML is a generalized markup language, where authors can decide the set of tags they are going to use in writing their documents, and it is used at its best to express the structure of the document and the semantically relevant elements, rather than their typographical properties.

Just like SGML documents, XML documents lack any machine-interpretable semantics. That is, XML dictates about the syntax of the markup, not about its meaning. This means on the one hand that authors are free to assign whatever meaning they want to the elements they choose to employ, but also that there must be a way for programs to make use of these elements in an appropriate way.

The solution taken within the XML family is to *map* (or *rewrite*) the source XML document (containing elements meaningful to the author) into a different XML document containing elements meaningful for the program that has to perform the application. Thus, a document containing elements such as section, titles, formulas, sentences, etc., all meaningful to its human author, will be rewritten into a new document containing elements such as blocks, paragraphs, inline styles, white regions, and so on, all meaningful to a displaying or printing program.

This is done by XSLT [9], the transformation part of XSL (Extensible Stylesheet Language). XSLT is a mapping language for XML elements. Each XSL stylesheet is made of rules composed of a pattern and an action. The pattern identifies an element of the XML source to which the action should apply; a pattern may specify all elements of a given type, all elements contained in a given subtree, all elements having a given attribute, etc. When the most appropriate pattern is found for the current XML element, the action part is considered. XSL actions are simply XML subtrees that are written in the destination document in place of the element being considered.

The Document Object Model [17] is a platform- and language-neutral interface that allows programs and scripts to dynamically access and update the content, structure and style of documents. The Document Object Model provides a standard set of objects for representing HTML and XML documents, a standard model of how these objects can be combined, and a standard interface for accessing and manipulating them. Through DOM it is possible to develop applications that use XML documents in a very general way.

## 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.

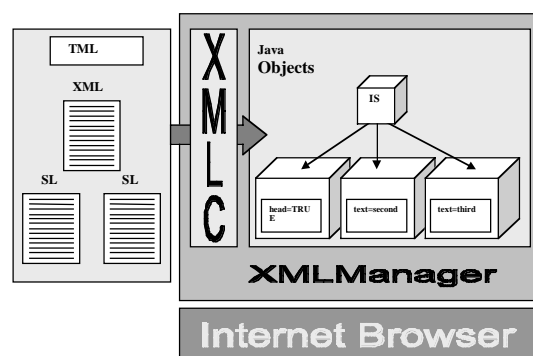


Fig. 1: The architecture of the XMLC application.

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 displet that can be activated. XMLC will finally instantiate all the required displets, creating a tree of runnable objects. Figure 1 shows a schema of the architecture.

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 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.

The current implementation of the XMLC architecture is in Java; a displet can be any sort of Java classes, but using the concept of JavaBeans it is easy to create sophisticated and interoperable displets: the use of JavaBeans Containers and Components, which can be easily organized in hierarchies, nicely fits 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. Parameters of the applet are the XML document 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 JavaBeans 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.

In the following we briefly report on the simple, display-oriented displets that have been implemented.

#### **Text and images**

We have implemented support for text oriented XML elements. The level of support is comparable to that of HTML 1.0 text elements: basic blocks (P, UL, OL and header elements) and inline chunks (like I, B, TT elements, etc.), plus an image tag and a simple inline hypertextual link.

There are three displets taking care of the display of text elements: Paragraph, Word and MultiWord.

- A Paragraph is a container spaced vertically (that is, two or more Paragraphs are put one above the other), with parameterized margins, line height and several other aspects.
- A Word is a component taking care of the display of a single word (separated by variable-width white space). Words are spaced horizontally and can control font, size, style, baseline and a few other parameters of their content.
- A MultiWord is a container for Words that is still spaced horizontally. It is used to group together Words that share a common propriety (for instance, that belong to the same run of bold characters, or to the same hypertext anchor).

#### **ACTIVE DOCUMENTS FOR SW ENGINEERING**

In our research group we have adopted a displet-based approach to building tools for software engineering notations, like for instance XML, Z [6] 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.

## UML specifications

A key issue is how to define a DTD for a complex sw 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 link 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.

The screenshot shows the Elmuth application running in Microsoft Internet Explorer. The browser window title is "Elmuth - UML Metamodel Example - Microsoft Internet Explorer". The address bar shows the URL "D:\Develop\Elmuth\examples\backbone\dictionary.html".

The application interface is divided into several panes:

- UML Metamodel Tree View:** A tree structure showing the hierarchy of the UML metamodel. The root is "-Document", followed by "-UML Metamodel: Core Package - Backbone", then "-Foundation.Core". Under "-Foundation.Core", there are several elements: "Element", "-ModelElement" (with sub-elements "name" and "visibility"), "+Feature", "+Namespace", "+Parameter", "+Constraint", "+GeneralizableElement", and "Classifier".
- UML Class Diagram:** A class diagram showing the relationships between classes. "Element" is the superclass of "ModelElement" and "Feature". "ModelElement" has two associations: "\* ownedElement" (to "Element") and "1..\* constrainedElement" (to "ModelElement"). "Feature" is a subclass of "ModelElement".
- Code Editor:** Shows the Java code generated from the class diagram. The code defines a package "Foundation.Core" and a class "ModelElement" that extends "Element". It includes attributes "name" and "visibility", and a constructor.
- Data Dictionary:** A table showing the details of the "ModelElement" class. It lists attributes like "name", "id", "visibility", "element type", "abstract", "root", "leaf", and "active".
- Hyperlinks:** A section with buttons for "General", "Relations", "Attributes", "Operations", "Parameters", and "Check".

Fig. 2: The representation of a UML diagram

## The Z notation

A complete support for the Z notation has been implemented [8]. The DTD we use is based on the ZIF Interchange Format [6], although, through the use of different XSL stylesheets, other syntaxes can be used as well.

```
... <schemadef style="vert" purpose="state">
PhoneDB
<decpart>
  <declaration>
    _known: &pset; NAME
  </declaration>
  <declaration>
    phone: NAME &pfun; PHONE
  </declaration>
</decpart>
```

```

<formals> K,L,Z </formals>
<axpart>
  <predicate>
    known = &dom; phone
  </predicate>
</axpart>
</schemadef>
...

```

Table 3: The visualization of the Z specification

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.).

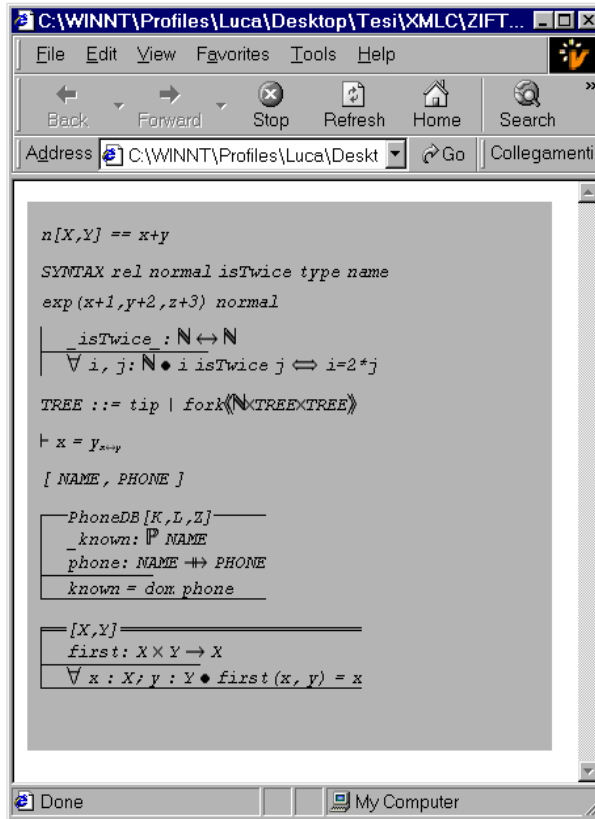


Fig. 3: The visualization of the Z specification

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 table 3 we show a small fragment of a Z specification (expressed in ZIF) and in Fig. 3 its rendering.

### Finite State Machines

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

```

<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"

```

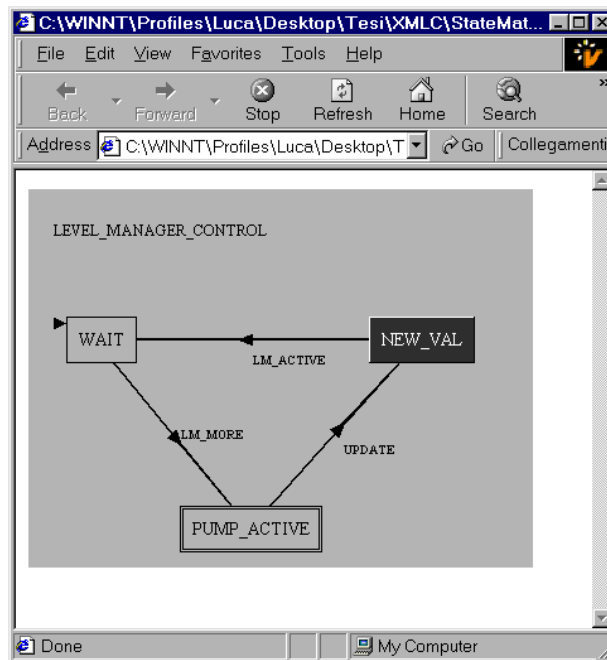
```

origin="100,200">
PUMP_ACTIVE
</State>
</StateMate>

```

**Table 3: A simple finite state machine in XML format,**

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 but 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).



**Fig. 4: The representation of the FSM of table 3**

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 Tab.3 we provide an example of a simple StateMate fragment shown in Fig. 4. 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.

#### **Hypertext links**

W3C is proposing two languages to express hypertext links in XML. XPointer [11] provides a way to express sub-resource addresses within XML documents and other resources, and XLink [13] 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.

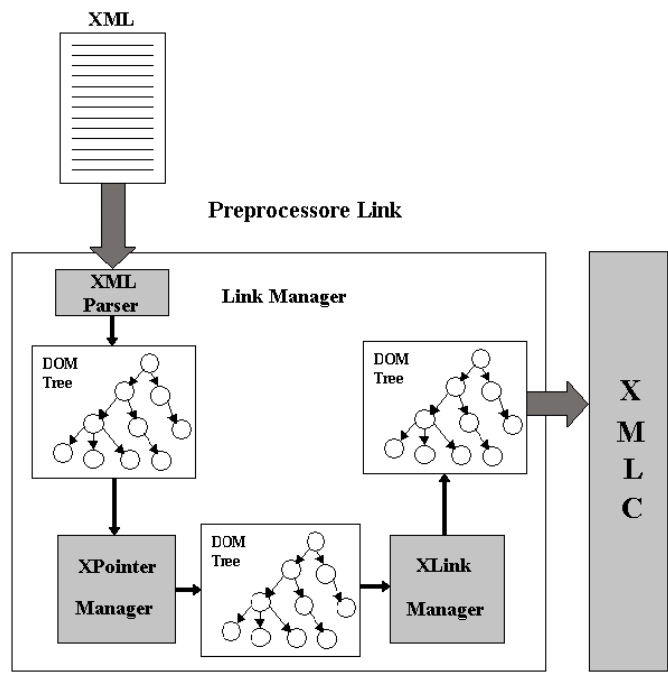


Fig. 5: The XLink-enabled architecture of XMLC

XLinks extends 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;
- 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 Fig. 5.

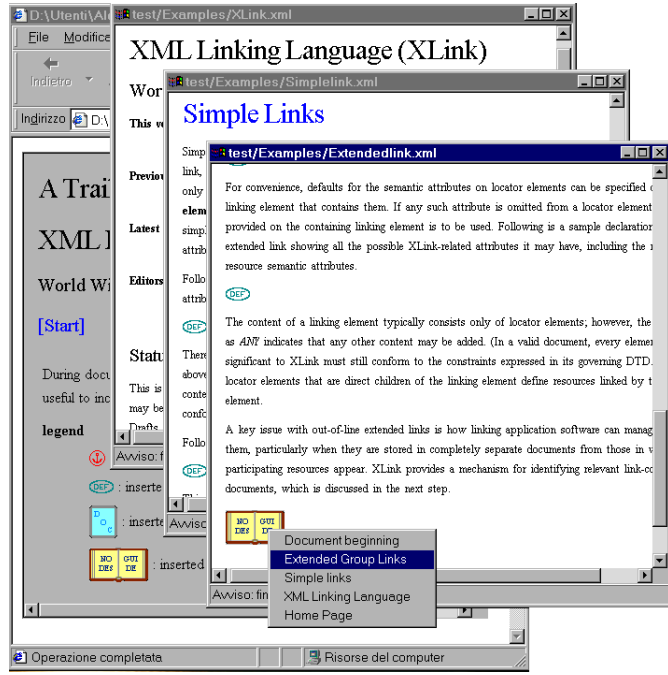


Fig. 6: A simple hyperlinked document group

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 are 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. Fig. 6 shows a sample hyperlinked document group.

## 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. Furthermore the Web, born as an exchange medium for professionals, has clearly become a graphic, impact-oriented one-way medium for the presentation of corporate truths or inflated egos.

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 XLinks and XPointers. 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/>

## ACKNOWLEDGMENTS

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 class of 1999 at the University of Bologna.

## REFERENCES

- [1] H. Ashman, V. Balasubramanian, M. Bieber, H. Oinas-Kukkonen (Eds.), *Proceedings of the 2nd International Workshop on Incorporating Hypertext Functionality into Software Systems (HTFII)*, Hypertext '96 Conference, Washington, 1996, <http://www.cs.nott.ac.uk/~hla/HTF/HTFII/Proceedings.html>
- [2] M. Bieber, F. Vitali, H. Ashman, V. Balasubramanian, H. Oinas-Kukkonen, 'Fourth Generation Hypertext: Some Missing Links for the World Wide Web', *International Journal of Human-Computer Studies* 47, Academic Press, 1997, p. 31-65.
- [3] L. Bompani, P. Ciancarini, F. Vitali, 'Active Documents in XML', *ACM SigWeb Newsletter* 8 (1), 1999, p. 27-32.
- [4] T. Bray, J. Paoli, C. M. Sperberg-McQueen, *Extensible Markup Language, (XML) 1.0*, W3C Recommendation 10 February 1998, <http://www.w3.org/TR/REC-xml>
- [5] D. Brickley, R.V. Guha, *Resource Description Framework (RDF) Schema Specification*, W3C Proposed Recommendation, 3 March 1999, <http://www.w3.org/TR/PR-rdf-schema>
- [6] S. Brien and J. Nicholls, *Z Base Standard*, Programming Research Group, Oxford, UK, 1992.
- [7] P. Ciancarini, A. Rizzi and F. Vitali, "An Extensible Rendering Engine for XML and HTML", *Computer Networks and ISDN Systems*, 30(1-7):225-238, 1998.

- [8] P. Ciancarini, F. Vitali, C. Mascolo, 'Managing complex documents over the WWW: a case study for XML', *IEEE Transactions on Knowledge and Data Engineering* 11 (4), 1999, p. 629-638.
- [9] J. Clark, *XSL Transformation (XSLT) 1.0*, W3C Draft 13 August 1999,  
<http://www.w3.org/TR/1998/WD-xslt>
- [10] D. Deach, *Extensible Stylesheet Language (XSL) 1.0*, W3C Draft 18 August 1998,  
<http://www.w3.org/TR/1998/WD-xsl>
- [11] S. DeRose, R. Daniel Jr., *XML Pointer Language (XPointer)*, W3C Working Draft, 9 July 1999,  
<http://www.w3.org/TR/WD-xptr>
- [12] Y. Goland, E. Whitehead, A. Faizi, S. Carter, D. Jensen, HTTP Extensions for Distributed Authoring -- WEBDAV, *IETF RFC 2518*, February 1999,  
<http://www.ietf.org/rfc/rfc2518.txt>
- [13] D. Orchard, S. DeRose, B. Trafford, *XML Linking Language (XLink)*, World Wide Web Consortium Working Draft 26 July 1998,  
<http://www.w3.org/TR/WD-xlink>
- [14] G. Rossi, H. Ziv (eds.), *Proceedings of the Fifth International Workshop on Engineering Hypertext Functionality into Future Information Systems (HTF5)*, ICSE 98 Conference, Kyoto, 1998,  
<http://www.ics.uci.edu/pub/kanderso/htf5/papers/>
- [15] F. Vitali, C. Watters (eds.), *Proceedings of the Fourth International Workshop on Hypertext functionality and the WWW (HTF4)*, 7th Int. WWW Conference, Brisbane, 1998,  
<http://dragon.acadiau.ca/~cwatters/htf4/>
- [16] F. Vitali, M. Bieber, 'Hypermedia on the Web: What Will It Take?', *ACM Computing Survey*, in print.
- [17] L. Wood, V. Apparao, S. Byrne, M. Champion, S. Isaacs, I. Jacobs, A. Le Hors, G. Nicol, J. Robie, T. Research, R. Sutor, C. Wilson and L. Wood, *Document Object Model (DOM) 1.0*, W3C Recommendation, 1 October 1998,  
<http://www.w3.org/TR/REC-DOM-Level-1>