

Experiments in the Use of XML to Enhance Traceability Between Object-Oriented Design Specifications and Source Code

Dr. Jim Alves-Foss
Center for Secure and
Dependable Software,
University of Idaho,
Moscow, ID 83844-1008
jimaf@cstds.uidaho.edu

Mr. Daniel Conte de Leon
Center for Secure and
Dependable Software,
University of Idaho,
Moscow, ID 83844-1008
danielc@cstds.uidaho.edu

Dr. Paul Oman
Schweitzer Engineering
Laboratory, Inc.,
Pullman, WA 99163-5603
paulom@selinc.com

Abstract

In this paper we explain how we implemented traceability between a UML design specification and its implementing source code using XML technologies. In our linking framework an XMI file represents a detailed-design specification and a JavaML file represents its source code. These XML-derivative representations were linked using another XML file, an Xlink link-base, containing our linking information. This link-base states which portions of the source code implement which portions of a design specification and vice-versa. We also rendered those links to an HTML file using XSL and traversed from our design specification to its implementing source code. This is the first step in our traceability endeavors where we aim to achieve total traceability among software life-cycle deliverables from requirements to source code.

1. Toward a Life-Cycle Based Software Engineering Approach

Most software developed in the commercial world today is designed and constructed without the use of formal verification and traceability aids because market pressures exert a schedule urgency, which encourages ad-hoc programming. While there exist techniques for developing reliable mechanical and electrical components of complex systems, the reliable construction of complex software systems has taken a back seat to market demands for newer functionality. Most of techniques used today to develop reliable computational systems, such as formal methods, structured testing, software process management, software modeling, and software metrics, are primarily centered on a single aspect of the software life-cycle. For example, many techniques and tools focus

on the programming-centric paradigm, where the goal of the methodology is to enhance the ability to write reliable software written in a specific modern programming language. What is needed is a higher-level approach and tools that works above the programming-centric paradigm to view the whole system being developed. The development of the system from a top-down perspective, without any presumed underlying technology, allows the creation of complex reliable systems without needless constraint of programming language or methodology.

Large commercial systems containing millions of lines of code cannot be implemented in a timely fashion if all portions of the system undergo formal verification in addition to unit and integration testing prior to release. The key to more rigorous large-scale software design and implementation is to establish an integrated software engineering environment that combines state-of-the-art technologies in the areas of graphical user interfaces, formal software specification/design, and code verification and validation. The intent is to target complex sections of software needing formal verification and those needing rigorous testing, at the expense of the more mundane subsections of the system requiring less rigor and attention. We have embarked on a three year project, funded by the Army Research Office, to explore the integration of existing software specification and verification techniques (from the formal methods domain) and existing Computer Aided Software Engineering (CASE) tools, with the goal of developing a full-spectrum software development approach based on widely available technologies (as opposed to language specific or process-centric environments). In our environment, a set of widely available and consistent tools are envisioned for formal specification, design, verification, testing, and change management supporting the entire software engineering life-cycle.

One common problem of the software development process is that it generates disparate documents (requirements specifications, design specifications, source code, etc.) written in several different specification and programming languages. In disciplined software engineering we need to link these documents together for traceability, consistency, change management, and evaluation. By creating visual, traceable links between requirements, design structures and code, we can assess the impact of requirement and code changes on the other corresponding work products.

In this paper we describe how we implemented traceability links between UML design specifications and its implementing Java source code using XML-derivates as specification languages and Xlink as linking language. We plan to use XML-derivates to represent all software life-cycle deliverables from requirements to specifications and embed traceability information in all of them. In this way engineers could identify and initiate the appropriate level of verification and validation necessary for each portion of the system during development or maintenance.

2. Using XML to Enhance Traceability Across Life-Cycle Products

One of the goals of our current research project is to add traceability between several disparate documents created by the software engineering process. In particular, we want to be able to trace object-oriented design specifications to source code. Towards this goal we have identified new technologies and designed a framework where traceability is an integral part of software development. In this framework, traceability links (and other desirable software properties) are embedded across life-cycle products by using an XML-derivative language.

2.1 The Extensible Markup Language (XML)

The Extensible Markup Language (XML) [7] is a new markup language developed by the World Wide Web Consortium [10]. XML technology is rapidly gaining widespread use among companies and organizations, and is changing the way they store and use Internet and Intranet based information systems. The rush for moving to XML is due to several advantages of the technology. First, as any other Internet standard, it is open and affordable. Second, XML is a multi-platform and non-vendor dependent technology, which is a plus in a rapidly changing environment. Third, XML is a natural outcome of HTML and Web technologies since it adds the advantage of being flexible enough to adapt to each company's business know-how. Fourth, XML easily conforms to the object-oriented paradigm with its elements, attributes, and hierarchical structure. Fifth, and last, it is a continuation of client-server and three-tier

architectures that allows us to disaggregate content from model, and data from format.

An XML file is a well-formed tagged file, where data is included between tags indicating markup. Both data and markup are included together in an XML file. The markup tells us something about the meaning of the marked up data. An XML file must be well-formed based on a set of grammar rules called a Document Type Definition (DTD) [7]. The extensibility of XML is based on DTDs and Schemas, a new richer language for defining XML document formats. This extensibility allows us to define the format of any document including software design specifications, source code, and any other software life-cycle deliverable. In our particular case, traceability, we are concerned with linking a design specification with its current source code implementation, so a DTD is necessary for the design specification and for the source code specification, in order to be able to represent design documents and source code documents in XML markup.

2.2 SeaBASS: A Test-Platform Java Application

SeaBASS stands for Certificate Based Authorization Simulation System. It is a software application developed at the Center for Secure and Dependable Software. Its purpose is to allow us to explore ways to build distributed authorization into computer systems. SeaBASS was implemented, using the Java programming language, as a distributed object system emulating file management, user management, machine management, and other applications to simulate a certificate based authorization mechanism. The size of the system is about 3000 lines of source code. Several reasons propelled us to use SeaBASS as a test-platform for our software engineering studies. First, the SeaBASS project was developed using object-oriented design and programming methods. Second, it is an internal development application so we have both design and source code specifications. And last, it is a relative small application suitable for the purposes of this work.

Although, SeaBASS is a small application and our final goal is to target large commercial systems, it is being used for the purposes of this work as a proof-of-concept. Our goal for large commercial systems is to be able to differentiate which portions of the system need formal verification and which ones can avoid that verification and be rather subject to informal testing, based on the criticality of the requirements. So we plan to develop an environment where we can differentiate and qualify portions of a small system and in the future port our work to a larger scale.

3. XML Extensions

The use of XML based languages is a fast growing and open technology that has the advantages of being non-platform, non-vendor dependent, and can be applied to the software engineering domain in order to improve software engineering practices by the use of automated tools. We aim to have an XML representation for each of the software life-cycle deliverable from requirements to source code, including links, comments, metrics and any other information necessary for a software project. In order to do this we need an XML-derivate language for each type of document of a software project. In particular we are looking for an XML representation of UML designs and Java source code. Fortunately part of that work is already completed and we currently can use the XMI [14] specification and a recent JavaML [11] specification with this purpose.

Our purpose in using this XML-based design and source code specifications is three fold: First, to have enhanced information about the source code such as software metrics and criticality level. Second, to be able to transform that information in other formats in order to have several points of view of the same source code (i.e.: software visualization). And third, to add traceability links and use them to navigate the source code itself and traverse from source code to design and vice-versa. XML is extensible enough to allow us to implement these objectives.

3.1 Using XMI and JavaML

The Object Management Group [15], lead by IBM Corporation [9], and other companies such as Rational Software Corporation [17], have developed a specification for writing object-oriented design specifications in XML. That language, called XML Metadata Interchange (XMI) [14], allows you to write a marked-up-text based object-oriented software design specification. For example, a Unified Modeling Language (UML) [13] specification can be serialized to an XMI file using Rational Rose [17] or the IBM XMI Toolkit [16]. Moreover, Argo UML [20] and MagicDraw UML design tools use XMI as native file format. The new marked up specification is a text file representing an object-oriented design of the software project. We used the XMI Toolkit to extract a design specification of our SeaBASS project. This design specification is an XML representation of the detailed-design of our software project and it is saved as an XMI file. This XMI file, as we explain later, allows us to add links to the source code since our detailed-design now is represented in an XML-derivative format.

In a similar manner, source-code in a software project can be represented using JavaML, a marked up representation of Java source-code developed by Greg

Badros, at the University of Washington [4]. Also, Evan Mamas at the University of Waterloo, Canada [11] developed a newer and different JavaML specification, along with a CppML (C++ markup language) and an OOML (object-oriented markup language). However, neither the XMI specification, nor Badros' JavaML work, nor Mama's set of XML-based OO languages include traceability links or any other information such as criticality level or other software properties we would like to specify in our markup language. Also, Luca Bompani, et al. are using XML to represent software development deliverables such as Z specifications [6]. Other authors such as Nentwich [12] are using Xlink to add links to an XMI file, though not to an XML representation of the source code.

We represented the Java source code of our SeaBASS project using Mamas' JavaML specification. This JavaML is saved as an XML file (cbass.javaml.xml) and allows us to complete both sides of our XML-representation requirements for the detailed-design and source code linking. Now we are in condition of designing and implementing our links between both.

4. Adding Traceability to Software Specifications

Linking dissimilar documents in an efficient and effective way can be achieved by using a common language to represent them. We used XMI, JavaML, and Xlink (all XML derivatives) to establish links between a UML design specification and the Java source code implementing it.

4.1 The XML Linking Language (XLink)

The World Wide Web Consortium is currently developing three language derivatives from XML. They are XLink, XPath, and XPointer. These languages allow us to embed traceability links in our XMI design specification and in our JavaML or CppML source code specification.

In a software project several XML linking files can relate design with source-code in different ways. All those links can reside on a simple file or in several files. The set of all links is a set of XML files, called a link-base, that contains information on how to traverse the project not just from design to source-code as expressed before, but from any software life-cycle deliverable to any other. Most important yet, since a computer can interpret these links, an automated system for generating links and checking project inconsistencies can be developed to do this work. In fact, the Xlinkit tool [12] does this task based in a proprietary rules language.

```
<!ELEMENT software-life-cycle-deliverables-links
  ( requirements-to-high-level-design-links ) ?
  ( high-level-design-to-detailed-design-links ) ?
  ( detailed-design-to-sourcecode-links ) ? >

<!ATTLIST software-life-cycle-deliverables-links
  xmlns:xsl      CDATA #REQUIRED
  xmlns:fo       CDATA #REQUIRED
  xmlns:xlink     CDATA #REQUIRED
  xmlns:relylinks CDATA #REQUIRED >

<!ELEMENT design-to-sourcecode-links
  ( xlink-extended-link ) * >

<!ELEMENT xlink-extended-link
  ( ( resource )+ , ( locator )+ , ( go )+ ) >

<!ATTLIST xlink-extended-link
  xlink:type      CDATA #REQUIRED
  xlink:title     CDATA #REQUIRED >...
```

Figure 1. Link-base DTD excerpt

4.2 Stating Linking Information Using Xlink

We used Xlink [8], a W3C recommendation, to design a link-base, which contains linking information between a UML design specification and its implementing Java source code. Our link-base is currently an XML file with the following structure: The document tag is called *software-life-cycle-deliverables-links*, under this tag different categories of links can be defined using new XML elements such as links between requirements and high-level design or links between detailed-design to source code. A tag called *design-to-sourcecode-links* will contain the elements that define links between our XMI design specification and our JavaML source code specification. These linking elements are Xlink extended links and are added as an element defined by the tag *xlink-extended-link*. The structure of this extended link elements is similar to the examples found in [8].

Figure 1 shows the general structure of a link-base file given by an excerpt of its document type definition (DTD). Each extended link states information on the resources and locators we want to link and the arcs that link them. In the case of our detailed-design to source code links resource is our XMI design specification and our locators are the JavaML files corresponding to each of the Java classes. Arcs define which links are defined from the detailed-design to the source code.

Figure 2 shows a sample of a manually developed link-base containing links from our SeaBASS design specification to its implementing Java source code. This link-base indicates what documents belong to the detailed-design specification and what documents belong to the source code specification, and how these resources are linked to indicate what portions of the source code are implementing their corresponding design specification.

```
<software-life-cycle-deliverables-links
  xmlns:xsl=http://www.w3.org/1999/XSL/Transform
  xmlns:fo=http://www.w3.org/1999/XSL/Format
  xmlns:xlink=http://www.w3.org/1999/xlink
  xmlns:relylinks=http://www.einstein.cds.uidaho.edu/Rely/Linking >

  <design-to-sourcecode-links>

    <xlink-extended-link
      xlink:type      e="extended"
      xlink:title     ="DesignToSourceCodeLinks" >
      <!-- Resources are the design specification and
      source code specifications -->
      <resource
        xlink:type      ="resource"
        xlink:label     ="desingSpecification"
        xlink:title     ="Sea BASS Design Specification"
        xlink:href      ="desingspec.xmi.xml">
      </resource> ...
      <!-- There are many target locators
      determined for all the files
      composing the source code specification -->
      <locator
        xlink:type      ="locator"
        xlink:label     ="CBassInterface"
        xlink:title     ="CBassInterface Source Code"
        xlink:href      ="CBassInterface.javaml.xml" >
      </locator> ...
      <locator
        xlink:type      ="locator"
        xlink:label     ="CD"
        xlink:title     ="Class CD Source Code Spec."
        xlink:href      ="CD.javaml.xml">
      </locator>

      <!-- Following arc definitions determine links
      between the design specification and
      each of the files that contains a section of the
      source code for the project or package -->
      <go
        xlink:type      ="arc"
        xlink:from       ="desingSpecification"
        xlink:to         ="CBassInterface"
        xlink:title      ="
        "Go to CBassInterface-classSourceCode"
        xlink:show       ="replace"
        xlink:actuate    ="onRequest" >
      </go>...
      <go
        xlink:type      ="arc"
        xlink:from       ="desingSpecification"
        xlink:to         ="CD"
        xlink:title      ="Go to CD-classSourceCode"
        xlink:show       ="replace"
        xlink:actuate    ="onRequest">
      </go>...

    </xlink-extended-link>

  </design-to-sourcecode-links>

</software-life-cycle-deliverables-links>
```

Figure 2: Sample link-base excerpt

5. Navigating Software Specifications

The W3C, along with previously mentioned XML and Xlink specifications, have developed a transformation language called Extensible Stylesheet Language (XSL) [1]. XSL can be applied to our XML software life-cycle deliverables representation in order to transform documents, including design specifications and source code, into renderable files (i.e.: HyperText Markup Language - HTML or Scalable Vector Graphics - SVG) that can be viewed in different ways by different users.

5.1 Rendering an XMI Specification with Links to the Source Code

We adapted an XSL stylesheet from ArgoUML [20], also present in [15], which reads an XMI specification and generates an HTML table view of the classes and interfaces stated in the XMI specification. We manually added to the stylesheet templates, which render the previously added links in the XMI specification to html links in the JavaML representation of the Java source code. These links are generated in two different ways, first one depending on the name of the class at the detailed-design level and second one depending on the information stated by links in our link-base. Those links are explained in detail below.

The output of this stylesheet is an HTML file with table format similar to the one obtained with the ArgoUML stylesheet with the difference of the added links to the source code.

Table 1: Section of an XSL-rendered XMI design specification showing links to source code for the SeaBASS CD (Certificate Database) class

Class		CD
Attributes:		
Visibility	Type	Name
private	List	certs
Operations:		
Visibility	ReturnType	Name
public	void	CD
public	boolean	addCert
public	boolean	deleteCert
Source Code:		
Go to CD-classSourceCode		

Table 1 shows a section of the HTML output resultant of rendering the SeaBASS XMI file using the previously explained XSL stylesheet. The section of the table, has now two links to the source code specification. Both added links point to the JavaML representation of the Java source code implementing the SeaBASS CD (Certificate Database) class.

In Table 1 the first link (**CD**) corresponds to the class/interface title and stands in the box near the *Class* keyword. This link was generated by the modified stylesheet upon rendering the SeaBASS XMI specification and mapping the name of the class at detailed-design level to a similar name file but using the .javaml.xml extension. Figure 3 shows the portion of the XSL stylesheet that adds the row with the link in the HTML table.

<!-- Adds the name of the class / interface on first row of the class / interface table and creates an hyperlink to the JavaML source code file with same name of the class / interface and ".javaml.xml" extension -->

```
<xsl:element name="td">
  <xsl:attribute name="class">
    <xsl:text>class-name</xsl:text>
  </xsl:attribute>
  <xsl:element name="a">
    <xsl:attribute name="name">
      <xsl:value-of select="$element_name"/>
    </xsl:attribute>
    <xsl:attribute name="href">
      <xsl:value-of select="$element_name"/>
      <xsl:text>.javaml.xml</xsl:text>
    </xsl:attribute>
    <xsl:value-of select="$element_name"/>
  </xsl:element>
</xsl:element>
```

Figure 3. XSL template section, which adds an HTML link from design to source code based on class name

This kind of linking is limited to situations where there is just one source code file for each class/interface like in the Java language. Though, it has the advantage that can be automatically generated without user intervention since the information stated in the XMI specification link is enough to find the corresponding JavaML source code representation. No link-base is needed in this case. We plan to implement this kind of automatic linking functionality into a prototype of a software development environment.

The second link, which appears in table 1 at the last row, under the *Source Code* section, includes richer information. It is generated as a result of parsing the SeaBASS XMI specification along with the SeaBASS link-base (an XML file named class.linkbase.xml for our case). In this case the link is represented by a name taken from the xlink:title attribute of an arc element, which appeared in the link-base.

```

<!-- Organization : CSDS - University of Idaho
Purpose : Reads arcs stated in the linkbase and
adds links using hrefs from locators/resources. -->

<xsl:template name="read-arcs">
  <xsl:param name="class-name">
    <xsl:for-each select="document('cbass.linkbase.xml')/
      software-life-cycle-deliverables-links/
      design-to-sourcecode-links/
      xlink-extended-link/go">
      <xsl:if test="@xlink:type='arc'">
        <xsl:variable name="arc-from"
          select="@xlink:from"/>
        <xsl:variable name="arc-to"
          select="@xlink:to"/>
        <xsl:variable name="arc-title"
          select="@xlink:title"/>
        <xsl:for-each
          select="document('cbass.linkbase.xml')/
            software-life-cycle-deliverables-links/
            design-to-sourcecode-links/
            xlink-extended-link/resource">
          <xsl:variable name="resource-label"
            select="@xlink:label"/>
          <xsl:variable name="resource-title"
            select="@xlink:title"/>
          <xsl:variable name="resource-href"
            select="@xlink:href"/>
          <xsl:for-each
            select="document('cbass.linkbase.xml')/
              software-life-cycle-deliverables-links/
              design-to-sourcecode-links/
              xlink-extended-link/locator">
            <xsl:variable name="locator-label"
              select="@xlink:label"/>
            <xsl:variable name="locator-title"
              select="@xlink:title"/>
            <xsl:variable name="locator-href"
              select="@xlink:href"/>
            <xsl:if test="$locator-label=$arc-to">
              <xsl:if test="$resource-label=$arc-from">
                <xsl:if test="$locator-label=$class-name">
                  <xsl:element name="tr">
                    <xsl:element name="td">
                      <xsl:attribute name="class">
                        <xsl:text>feature-detail</xsl:text>
                      </xsl:attribute>
                    </xsl:element>
                    <xsl:element name="a">
                      <xsl:attribute name="href">
                        <xsl:value-of
                          select="$locator-href"/>
                      </xsl:attribute>
                      <xsl:value-of
                        select="$arc-title"/>
                      </xsl:element>
                    </xsl:element>
                  </xsl:element>
                </xsl:if>
              </xsl:if>
            </xsl:if>
          </xsl:for-each>
        </xsl:for-each>
      </xsl:if>
    </xsl:for-each>
  </xsl:template>

```

Figure 4: XSL Template to render extended links to the corresponding HTML class table.

In this case all arcs (see *go* elements in Figure 2) in the link-base are parsed and placed under this section if they belong to the corresponding defined resource (see *resource* elements in Figure 2) and locator (see *locator* elements in Figure 2). Traversing any of these links transfers us to the JavaML specification of the corresponding Java source code.

Figure 4, shows the XSL template that reads a link-base and an XMI specification, based on that information adds a row, in the *Source Code* section of the HTML table, for each arc defined in the link-base for the corresponding resource and locator.

This latest links are the most powerful and they solely depend on the information stated in the link-base. Also, several links can be added to the same class/interface just by adding more arcs to the link-base with the corresponding resource and locator names.

6. Software Specifications with XML

Using XML to represent and link object-oriented design specifications and source code is our first step towards being able to represent all software life-cycle deliverables using a common language (XML) that allows us to include linking, as well as any other information needed to apply formal verification or other software engineering techniques, such as metrics or specification of non-functional requirements. Neither UML specifications nor source code today includes the rigorous and necessary information we need to conduct formal verification and apply other software engineering techniques to the software piece as a whole. We need not just much more information embedded in each specification at all stages of software development; we also need to link all documents to explicitly state relationships between all life-cycle deliverables, and we need to be able to interpret all this new information using our today's computing resources.

Research work is currently being done to develop new XML-based formal specification languages. For example, Bompani et al. [6] developed an XML-derivate for expressing Z language requirement specifications; also Barnett and Schulte at Microsoft Research developed an executable specification language called AsmL [5] based on abstract state machines. We plan to use and extend these languages with linking, metrics, and all necessary information to conduct verification along all life-cycle deliverables.

7. Conclusions

We successfully used Xlink to design and implement links from a detailed-design specification to its implementing source code for our SeaBASS project. An

XMI file represented this detailed-design specification and a JavaML file did it for our SeaBASS source code. These XML-derivative representations were linked using an Xlink link-base containing our linking information. This link-base states which portions of our SeaBASS Java source code implement which portions of our SeaBASS design specification.

We modified a version of an XSL stylesheet that transforms an XMI design into an HTML table format. The new stylesheet transforms the XMI file and, at the same time, parses the links stated in our link-base. The result of this new stylesheet is an HTML file showing the design of our SeaBASS project in the same previous table format but now containing links to each of the source code files implementing the class at design level.

The main advantage in using XML is that it allows us to automate creation and management of links. In this case-study links in our XMI specification and templates in our XSL stylesheet were manually added. We plan to develop a prototype of a software development environment where links between object-oriented design and source code would be automatically created and maintained by the system based on implicit and explicit information given by the software engineer. Using such a tool would allow engineers to develop large software systems while maintaining consistency between object-oriented design specifications and its implementing source code.

We plan to use XML-derivates not just for detailed-design and source code, though for all software life-cycle deliverables, and this work is a proof-of concept that we can implement traceability among those disparate documents by means of XML and Xlink. Using XML for representing software documentation enable us to:

- Use languages and tools that already exist reducing the need for new tool development. For example for this project we used Xerces [19], Xalan [18], and StylusStudio, all tools previously developed to work with XML files.
- Implement traceability among life-cycle deliverables as done in this work using the Xlink proposed recommendation.
- View documents such as design and source code using several user-oriented views as the table view of the design with links to the source code explained in this paper.
- Answer questions with a query language similar to SQL and do data mining on specification files including combinations of requirements, design, and source code files.
- Distribute a software specification across the organization and implement Internet based software engineering information systems to aid the software engineering process.

- Adapt to different software platforms, development paradigms, and different programming languages.

This work may lead to a new open standard for software life-cycle deliverables representation and management, where life-cycle information can be communicated, viewed, traced, and transformed in ways not yet designed. All these characteristics are much desired in a software engineering environment. The representation of life-cycle deliverables with XML based languages allows us to implement tools to achieve these and other functionalities with the advantage of being a well supported, non-platform dependent, non-vendor dependent environment.

8. Future Work

Our current view of links stated in a link-base is using simple HTML links. More work is needed to add different behavior to the links and not to limit to today browsers' simple links behavior. For example add behavior to a link in a way that when we right-click on a class, in a graphical UML design, options to see/edit the source code appear or to navigate to the requirements that this class is implementing. This can be done by means of XSL and/or by means of an integrated software development environment.

In our linking example we were able to traverse from a detailed-design specification to its implementing source code. We are looking for forward and backward linking abilities; moreover our XML-based approach allows us to add links into any life-cycle deliverable specification. This work can be easily extended to include linking from source code to detailed-design as long as we develop a stylesheet for doing so.

The advantages of XSL as a formal language transformation engine do not limit to rendering design specifications to add navigable links from an XMI design specification to a JavaML source code representation using the HTML table representation we used here. An XMI file can be rendered to an SVG-based UML diagram. ArgoUML [20] uses this approach to save and show a UML diagram. Future work is needed to add graphically represented links in a UML diagram to its implementing source code and vice-versa.

Software visualization can be improved, by means of using XSL, adding graphical representations, of any software life-cycle deliverable, from requirements to source code. Using XSL we could show and visually link different specifications or different views of the same software system. Different fonts, colors, sound, etc. can be used to enhance the way we see software today. In essence, a combination of XML-derivates, XSL,

browsers, and applications can be used to implement multimedia software development environments.

9. Acknowledgment

We would like to thank Jie Dai, Mohan Muppallanni, Carol Taylor, and Kevin Twitchell, members of the RELY [2] project research group, for their contributions to the project where this work came from. We would also like to thank the U.S. Army Research Laboratory and the U.S. Army Research Office for funding our project under grant DDAD19-99-1-0088 and making this work possible.

10. References

- [1] S. Adler, A. Berglund, J. Caruso, S. Deach, P. Grosso, E. Gutentag, A. Milowski, S. Parnell, J. Richman, and S. Zilles: Editors. "Extensible Stylesheet Language (XSL) Version 1.0," W3C Proposed Recommendation, August 28, 2001, W3C 2001. Web Page: <http://www.w3.org/TR/xsl/>.
- [2] J. Alves-Foss, and P. Oman, "RELY: A Life-Cycle Based Software Engineering Approach to the Design and Implementation of Reliable Software Systems," Research Proposal, Center for Secure and Dependable Software, 1999. Web Site: <http://www.csd.su.umd.edu/RELY.html>.
- [3] Apache, "The Apache Software Foundation", Web Site: <http://www.apache.org/>.
- [4] G. Badros, "JavaML: A Markup Language for Java Source Code," University of Washington, 2000. Web Site: <http://www.cs.washington.edu/homes/gjb/papers/javaml/javaml.html>.
- [5] M. Barnett and W. Schulte, "The ABCs of Specification: AsmL, Behavior, and Components," June 15, 2001. Microsoft Research, Redmond, WA, 2001. <http://research.microsoft.com/foundations/#AsmL>.
- [6] L. Bompani, P. Ciancarini, F. Vitali, "Software Engineering and The Internet: A Roadmap," Department of Computer Science, University of Bologna, 2000. (To be published).
- [7] T. Bray, J. Paoli, C.M. Sperberg-McQueen, and E. Maler: Editors, "Extensible Markup Language (XML) 1.0 (Second Edition)," W3C Recommendation, October 6, 2000, W3C 2000. Web Page: <http://www.w3.org/TR/REC-xml/>.
- [8] S. DeRose, E. Maler, and D. Orchard: Editors, "XML Linking Language (XLink) Version 1.0," W3C Recommendation, June 27, 2001, W3C 2000. Web Page: <http://www.w3.org/TR/xlink/>.
- [9] IBM, "International Business Machines Corporation", Web Site: <http://www.ibm.com/>.
- [10] I. Jacobs, "About the World Wide Web Consortium (W3C)", The World Wide Web Consortium, March 2001. Web Site: <http://www.w3.org/>.
- [11] E. Mamas, and K. Kontogiannis, "Towards Portable Source Code Representation Using XML," Department of Electrical and Computer Engineering, University of Waterloo, Canada, 2001. (To be published).
- [12] C. Nentwich, L. Capra, W. Emmerich, and A. Finkelstein, "Xlinkit: A Consistency Checking and Smart Link Generation Service," (Research Note,) Department of Computer Science - University College London, London, UK 2000. Web Site: <http://www.xlinkit.com/>.
- [13] Object Management Group, "OMG Unified Modeling Language Specification", Version 1.4, September 07, 2001, OMG 2001. Web Page: <http://www.omg.org/technology/documents/formal/uml.htm>.
- [14] Object Management Group, "OMG XML Metadata Interchange Specification (XMI)", Version 1.1, November 02, 2000, OMG 2001. Web Page: <http://www.omg.org/cgi-bin/doc?formal/2000-11-02>.
- [15] OMG, "Object Management Group", Web Site: <http://www.omg.org/>.
- [16] K. Poole, S. Brodsky, G. Doney, A. Gleboy, M. Golding, T. Grose, H. Huang, C. Rich, C. Tung, and S. Wang, "The XMI Toolkit," April 25, 2000, IBM Alphaworks 2000. Web Site: <http://www.alphaworks.ibm.com/aw.nsf/techmain/xmitoolkit>.
- [17] Rational, "Rational Software Corporation", Web Site: <http://www.rational.com/>.
- [18] The Apache XML Project, "The Xalan XSLT Processor," The Apache Software Foundation, 2001. Web Site: <http://xml.apache.org/xalan-j/>.
- [19] The Apache XML Project, "The Xerces XML Parser for Java," The Apache Software Foundation, 2001. Web Site: <http://xml.apache.org/xerces-j/>.
- [20] University of California, "ArgoUML: An Object Oriented UML Design Tool", Tigris Open Source 2001. Web Site: <http://argouml.tigris.org/>.